



acontis technologies GmbH

SOFTWARE

EC-Simulator

User Manual

Version 3.2

Edition: February 22, 2024

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	8
1.1	What is EtherCAT?	8
1.2	EC-Simulator - Features	8
1.3	Editions (HiL / SiL)	10
1.3.1	Hardware-in-Loop (HiL) Simulation	10
1.3.2	Software-in-the-Loop (SiL) Simulation	10
1.4	Protected version	11
1.4.1	Licensing procedure for Development Licenses	11
1.4.2	Licensing procedure for Runtime Licenses	12
1.5	Known restrictions	13
1.6	License	13
1.6.1	EC-Simulator license	13
1.6.2	Free Open Source Software contained in EC-Simulator	13
1.6.3	Free Open Source Software supported by EC-Simulator	14
2	Getting Started	15
2.1	EC-Simulator Architecture	15
2.2	Extended EtherCAT Network Configuration (EXI)	15
2.3	HiL Simulation Software architecture	16
2.4	SiL Simulation Software architecture	16
2.5	Mixed mode of real slaves and simulated slaves	17
2.6	Operating system configuration	18
2.7	Running EcSimulatorHilDemo	19
2.7.1	Setting up and running EcSimulatorHilDemo	19
2.7.2	Command line parameters	20
2.8	Using emllSimulator (SiL)	29
2.8.1	Simulator Ethernet Driver parameters	30
2.9	EC-Simulator Software Development Kit (SDK)	31
2.9.1	OS Compiler settings	32
2.10	Real-time Ethernet Driver configuration for EcSimulatorHilDemo	32
3	Software Integration	33
3.1	EcSimulatorHilDemo	33
3.1.1	File reference	33
3.1.2	Simulator lifecycle	33
3.1.3	Logging	34
3.2	Process data update and synchronization	34
3.2.1	Forced Data, Firmware Data, DPRAM	35
3.2.2	EC-Simulator stack as process data memory provider	36
3.3	Accessing process data in the application	38
3.3.1	Process Data Access Functions selection	38
3.3.2	Process variables' offset and size	39
3.4	Hot Connect	41
3.4.1	Configured Station Alias	41
3.5	Error detection and diagnosis	42
3.6	RAS-Server for EC-Lyser and EC-Engineer	42
3.6.1	Integration Requirements	42
3.6.2	Pseudo Code	43
3.6.3	Required API Calls	44
3.7	EC-Simulator stack Source Code	50
3.8	EcSimulatorHilDemoMotion	51
3.8.1	Command line parameters	51
4	Platform and Operating Systems (OS)	52

4.1	tenAsys INtime	53
4.1.1	Setting up and running EcSimulatorHilDemo	53
4.1.2	OS Compiler settings	53
4.2	Linux	54
4.2.1	OS optimizations	54
4.2.2	atemsys kernel module	55
4.2.3	Unbind Ethernet Driver instance	56
4.2.4	Docker	57
4.2.5	Setting up and running EcSimulatorHilDemo	58
4.2.6	OS Compiler settings	59
4.3	QNX Neutrino	60
4.3.1	Thread priority	60
4.3.2	Unbind Ethernet Driver instance	60
4.3.3	IOMMU/SMMU support	60
4.3.4	Setting up and running EcSimulatorHilDemo	60
4.3.5	OS Compiler settings	61
4.4	Windriver VxWorks	62
4.4.1	VxWorks native	62
4.4.2	SNARF Ethernet Driver	63
4.4.3	Setting up and running EcSimulatorHilDemo	63
4.4.4	OS Compiler settings	64
4.5	Microsoft Windows	65
4.5.1	EcSimulatorHilDemo - Microsoft Windows	65
4.5.2	OS Compiler settings	65
4.5.3	EcatDrv for Kernel Timer	66
4.5.4	EcatDrv for Real-time Ethernet Driver	76
4.6	Xenomai	83
4.6.1	Setting up and running EcSimulatorHilDemo	83
4.6.2	OS compiler settings	84
5	Real-time Ethernet Driver	85
5.1	Real-time Ethernet Driver initialization	85
5.1.1	Real-time Ethernet Driver instance selection via PCI location	88
5.2	Intel Pro/1000 - emllI8254x	88
5.3	Intel Pro/1000 - emllIntelGbe	89
5.4	Intel Pro/100 - emllI8255x	90
5.5	Broadcom BcmGenet - emllBcmGenet	91
5.6	Broadcom BcmNetXtreme - emllBcmNetXtreme	91
5.7	Beckhoff CCAT - emllCCAT	91
5.8	Texas Instruments CPSW - emllCPSW	92
5.8.1	CPSW usage under Linux	93
5.9	DW3504 - emllDW3504	94
5.10	Freescale TSEC / eTSEC - emllETSEC	95
5.10.1	ETSEC supported MAC's	97
5.10.2	Shared MII bus	97
5.10.3	Locking	97
5.10.4	Link check	98
5.10.5	Fixed Link	98
5.11	Freescale FslFec - emllFslFec	98
5.12	Xilinx Zynq-7000/Ultrascale (GEM) - emllGEM	100
5.13	Texas Instruments ICSS - emllICSS	101
5.13.1	TTS Feature	102
5.13.2	TI AM335x ICEV2	103
5.13.3	TI AM57xx IDK	103
5.13.4	AM5728 IDK and AM5718 IDK boards and Technical Limitations	103
5.14	Windows NDIS - emllNdis	103
5.15	Windows WinPcap - emllPcap	104
5.15.1	WinPcap, Npcap support	105

5.16	RDC R6040 - emllR6040	106
5.17	emllRemote	106
5.18	Realtek RTL8169 - emllRTL8169	107
5.18.1	RTL8169 usage under Linux	107
5.19	Renesas SHEth - emllSHEth	108
5.19.1	SHEth link status update	109
5.19.2	SHEth usage under Linux	109
5.20	VxWorks SNARF - emllSNARF	110
5.21	Linux SockRaw - emllSockRaw	110
5.22	Windows TAP - emllTap	111
6	Application programming interface, reference	112
6.1	Generic API return status values	112
6.2	Multiple EtherCAT Network Support	112
6.2.1	Overview	112
6.2.2	Licensing	113
6.3	General functions	113
6.3.1	esInitSimulator	113
6.3.2	esDeinitSimulator	117
6.3.3	esGetSimulatorParms	117
6.3.4	esSetSimulatorParms	117
6.3.5	esGetSrcMacAddress	118
6.3.6	esSetLicenseKey (HiL)	118
6.3.7	esSetOemKey (HiL)	119
6.3.8	esConfigureNetwork	119
6.3.9	esSetSlaveSscApplication	121
6.3.10	esRegisterClient	121
6.3.11	esUnregisterClient	122
6.3.12	esExecJob	123
6.3.13	esGetMasterState	125
6.3.14	esIoControl	125
6.3.15	esIoControl - EC_IOCTL_GET_PDMEMORYSIZE	126
6.3.16	esIoControl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB	126
6.3.17	esIoControl - EC_IOCTL_ISLINK_CONNECTED	127
6.3.18	esIoControl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO	127
6.3.19	esIoControl - EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES	128
6.3.20	esIoControl - EC_IOCTL_SIMULATOR_SET_MBX_PROCESS_CTL	128
6.3.21	esIoControl - EC_IOCTL_SIMULATOR_GET_MBX_PROCESS_CTL	129
6.3.22	esIoControl - EC_LINKIOCTL_XXXX	129
6.3.23	esIoControl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS	129
6.3.24	esIoControl - EC_LINKIOCTL_GET_SPEED	130
6.3.25	esGetVersion	130
6.3.26	esGetText	130
6.3.27	esGetMemoryUsage	131
6.3.28	esLogFrameEnable	131
6.3.29	esLogFrameDisable	132
6.4	Process Data Access Functions	132
6.4.1	esGetProcessImageInputPtr	132
6.4.2	esGetProcessImageOutputPtr	133
6.4.3	EC_COPYBITS	134
6.4.4	EC_GET_FRM_WORD	135
6.4.5	EC_GET_FRM_DWORD	135
6.4.6	EC_GET_FRM_QWORD	136
6.4.7	EC_SET_FRM_WORD	136
6.4.8	EC_SET_FRM_DWORD	136
6.4.9	EC_SET_FRM_QWORD	137
6.4.10	EC_GETBITS	137
6.4.11	EC_SETBITS	137

6.4.12	esGetProcessData	138
6.4.13	esSetProcessData	139
6.4.14	esSetProcessDataBits	139
6.4.15	esGetProcessDataBits	140
6.5	Notifications	140
6.6	esNotifyApp	141
6.7	Network operation functions	142
6.7.1	esConnectPorts	142
6.7.2	esDisconnectPort	142
6.7.3	esPowerSlave	143
6.8	Error simulation functions	143
6.8.1	esSetErrorAtSlavePort	143
6.8.2	esSetErrorGenerationAtSlavePort	144
6.8.3	esResetErrorGenerationAtSlavePorts	144
6.8.4	esSetLinkDownAtSlavePort	145
6.8.5	esSetLinkDownGenerationAtSlavePort	145
6.8.6	esResetLinkDownGenerationAtSlavePorts	146
6.8.7	esLogFrameEnableAtSlavePort	146
6.8.8	esLogFrameDisableAtSlavePort	146
6.8.9	esSendSlaveCoeEmergency	147
6.9	Slave control and status functions	147
6.9.1	esGetNumConfiguredSlaves	147
6.9.2	esGetNumConnectedSlaves	148
6.9.3	esGetSlaveId	148
6.9.4	esGetSlaveIdAtPosition	148
6.9.5	esGetSlaveState	149
6.9.6	esSetSlaveState	149
6.9.7	esIsSlavePresent	150
6.9.8	esGetSlaveProp	151
6.9.9	esGetSlavePortState	151
6.9.10	esGetSlaveInpVarInfoNumOf	152
6.9.11	esGetSlaveOutpVarInfoNumOf	152
6.9.12	esGetSlaveInpVarInfo	153
6.9.13	esGetSlaveInpVarInfoEx	154
6.9.14	esGetSlaveOutpVarInfo	155
6.9.15	esGetSlaveOutpVarInfoEx	156
6.9.16	esGetSlaveInpVarByObjectEx	156
6.9.17	esGetSlaveOutpVarByObjectEx	157
6.9.18	esFindInpVarByName	157
6.9.19	esFindInpVarByNameEx	158
6.9.20	esFindOutpVarByName	158
6.9.21	esFindOutpVarByNameEx	158
6.9.22	EC_NOTIFY_SLAVE_PRESENCE	159
6.9.23	EC_NOTIFY_SLAVE_STATECHANGED	159
6.9.24	EC_NOTIFY_HC_TOPOCHGDONE	160
6.9.25	esWriteSlaveRegister	160
6.9.26	esReadSlaveRegister	161
6.9.27	esReadSlaveEEPROM	162
6.9.28	esWriteSlaveEEPROM	163
6.9.29	esGetCfgSlaveInfo	163
6.9.30	esGetSimSlaveInfo	167
6.9.31	esGetBusSlaveInfo	169
6.9.32	esReadSlaveIdentification	172
6.9.33	esGetSlaveStatistics	173
6.9.34	esClearSlaveStatistics	173
6.10	ADS over EtherCAT (AoE)	174
6.10.1	esAoeGetSlaveNetId	174
6.10.2	esAoeRead	174

6.10.3	esAoeWrite	175
6.10.4	esAoeReadWrite	176
6.10.5	esSetSlaveAoeObjectTransferCallbacks	177
6.10.6	AoE Simulator and Master Example	177
6.11	CAN application protocol over EtherCAT (CoE)	183
6.11.1	esExtendSlaveCoeObjectDictionary	183
6.11.2	esDeleteSlaveCoeObject	183
6.11.3	esClearSlaveCoeObjectDictionary	184
6.11.4	esResetSlaveCoeObjectDictionary	184
6.11.5	esSetSlaveCoeObjectTransferCallbacks	184
6.11.6	esCoeSdoDownload	187
6.11.7	esCoeSdoUpload	188
6.11.8	esCoeGetODListReq	188
6.11.9	esCoeGetObjectDescReq	189
6.11.10	esCoeGetEntryDescReq	190
6.11.11	CoE transfer Simulator and Master Example	191
6.12	Ethernet over EtherCAT (EoE)	195
6.12.1	esEoeSendFrame	195
6.12.2	esGetCfgSlaveEoeInfo	195
6.12.3	EoE Ping Example	198
6.13	File access over EtherCAT (FoE)	202
6.14	Vendor specific access over EtherCAT (VoE)	206
6.14.1	esVoeSend	206
6.14.2	esSetVoeReceiveCallback	206
6.14.3	VoE Receive Example	207
6.15	Distributed Clocks (DC)	208
7	Error Codes	209
7.1	Groups	209
7.2	Generic Error Codes	209
7.3	DCM Error Codes	220
7.4	ADS over EtherCAT (AoE) Error Codes	222
7.5	CAN application protocol over EtherCAT (CoE) SDO Error Codes	224
7.6	File Transfer over EtherCAT (FoE) Error Codes	227
7.7	Servo Drive Profil over EtherCAT (SoE) Error Codes	229
7.8	Remote API Error Codes	233

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 Master Classes

Feature name	Short description	Implemented	Feature ID
Service Commands	Support of all commands	x	101
IRQ field in datagram	Use IRQ information from Slave in datagram header	x	102
Slaves with Device Emulation	Support Slaves with and without application controller	x	103
EtherCAT State Machine	Support of ESM special behavior	x	104
Error Handling	Checking of network or slave 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 Master	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

continues on next page

Table 1 – continued from previous page

Feature name	Short description	Implemented	Feature ID
Explicit Device identification	Identification used for Hot Connect and prevention against cable swapping	x	303
Station Alias Addressing	Support configured station alias in slave, 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
Multiple Mailbox channels		x	403
Mailbox polling	Polling Mailbox state in slaves	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 compensation	Continuous Calculation of the propagation delay	–	1102

continues on next page

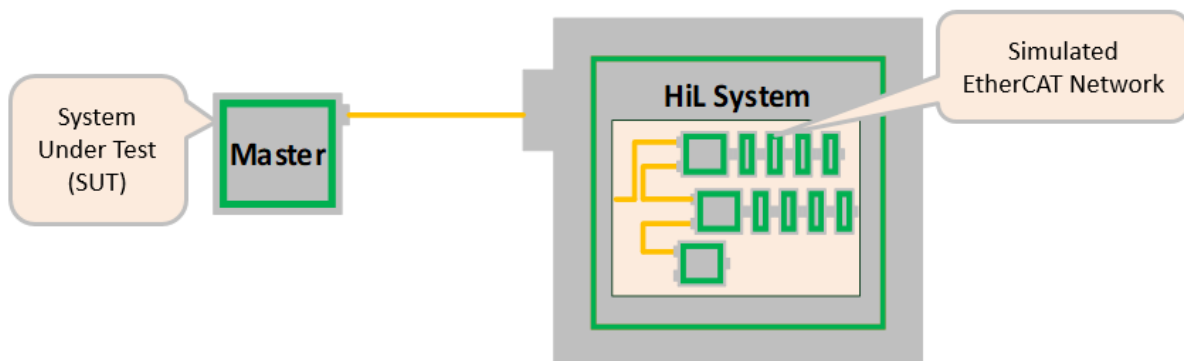
Table 1 – continued from previous page

Feature name	Short description	Implemented	Feature ID
Sync window monitoring	Continuous monitoring of the S ynchronization dif-ference in the slaves	–	1103
via Master	Information is given in ENI file or can be part of any other network confi-guration Copying of the data can be handled by mas-ter stack or master's application	x	1201

1.3 Editions (HiL / SiL)

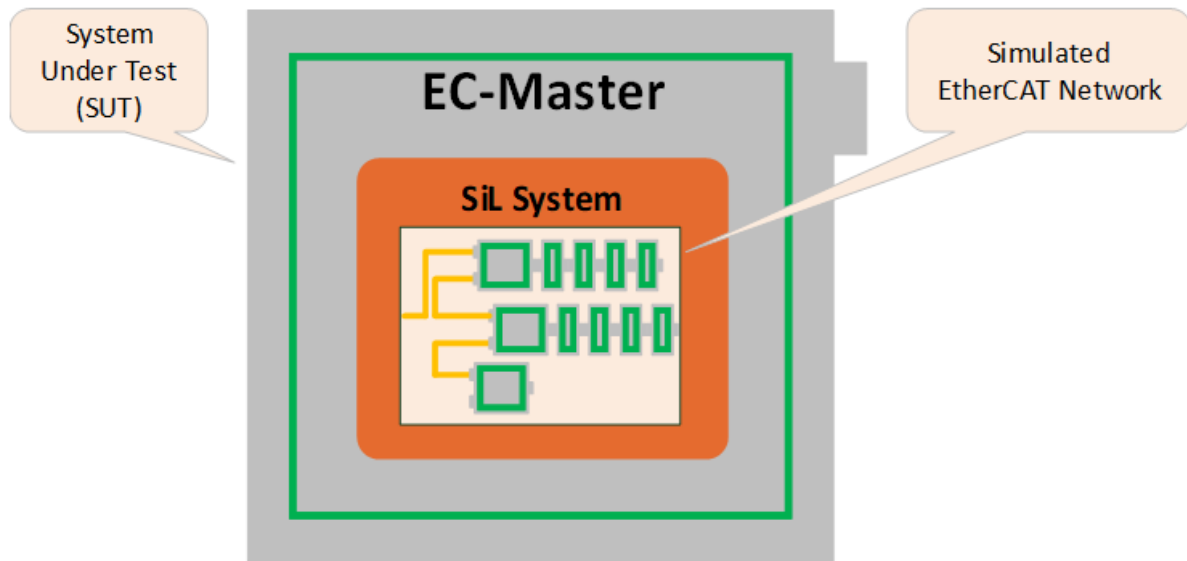
1.3.1 Hardware-in-Loop (HiL) Simulation

The System-Under-Test (SUT) is communicating via an EtherCAT cable with the EC-Simulator software running on an external hardware, the HiL System. In this setup the unchanged application together with the EtherCAT master 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 EtherCAT 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).

Source

Binary and source code without protection

Unrestricted:

Binary without MAC protection

The protected version is limited to 5 slaves 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 slaves require extended information to be simulated.

See also:

Extended EtherCAT Network Configuration (EXI)

The “<ExtendedInfo>”-Tag in EXI are 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@wacławek.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
```

(continues on next page)

(continued from previous page)

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.

WinPCap

The WinPCap library is supported, but not shipped with EC-Simulator.

Npcap

The Npcap library is supported, but not shipped with EC-Simulator.

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 interfaces are C language interfaces, thus the stack can be used in ANSI-C as well as in C++ environments.

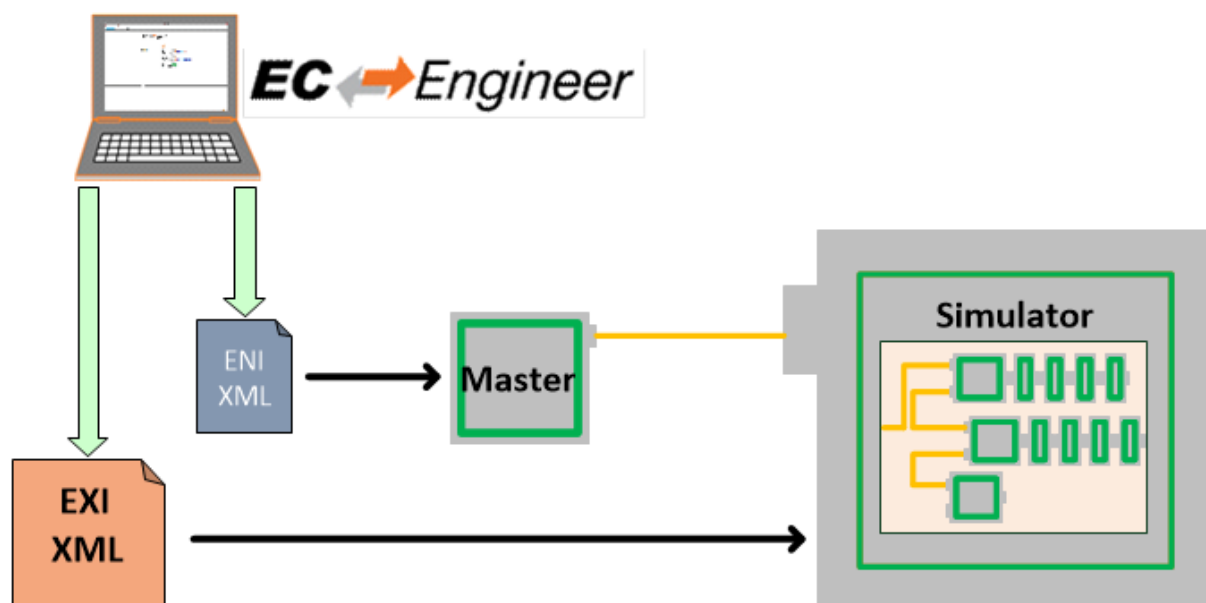
There are two different architecture editions as described below. The architectures contain the following:

- **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 a 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 master 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 slaves. 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 slave vendors, from master stack vendors and from EtherCAT configuration tools. Thus inter-operability between those vendors is guaranteed.

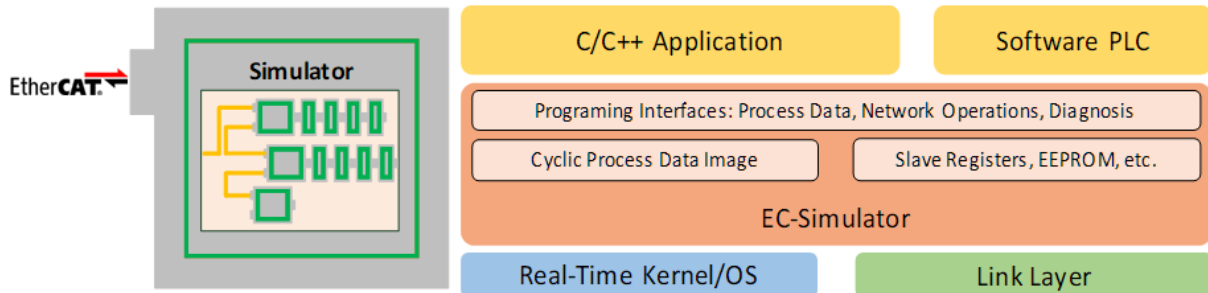
Because the ENI file does not include information about the slave hardware, the EXI (EtherCAT Extended Information) file includes EEPROM content and Slave Object Dictionary from ESI and ESC register values from real network is typically needed. The EXI file can be exported using EC-Engineer.



2.3 HiL Simulation Software architecture

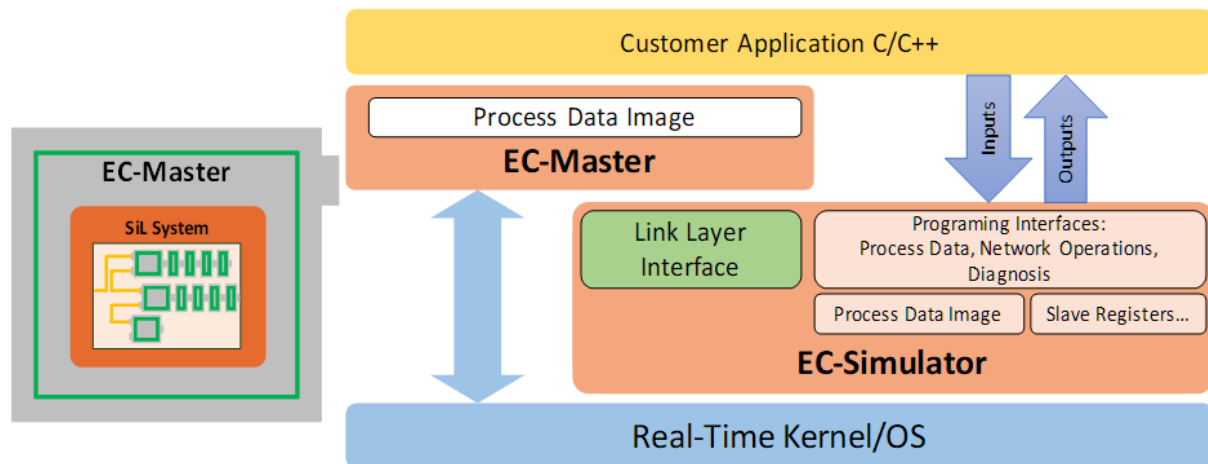
The EC-Simulator software is ready-to-run on various operating systems, e. g. Linux and Windows, and makes use of the Real-time Ethernet Drivers.

The EC-Simulator offers a “C” language API for integration into a Software PLC (LogicLab, logi.cals) or into professional simulation software like ISG Virtuos, etc.



2.4 SiL Simulation Software architecture

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



See also:

`EC_T_SIMULATOR_INIT_PARMS::apLinkParms` at `esInitSimulator()`.

2.6 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)*.

2.7 Running EcSimulatorHilDemo

The EcSimulatorHilDemo is an EC-Simulator example application that handles the following tasks:

- Showing basic EtherCAT communication
- Stack initialization
- Process Data operations for e.g. Beckhoff EL2004, EL1004 and EL4132
- Periodic diagnosis task
- Periodic Job Task
- Logging to EcSimulator0.log, error0.log

The EcSimulatorHilDemo is available “out of the box” for different operating systems. The operating system must be prepared for running EtherCAT® applications,

See also:

Operating system configuration

2.7.1 Setting up and running EcSimulatorHilDemo

Linux (EcSimulatorHilDemo)

1. Operating system configuration: See the section Operating system configuration for how to prepare the operating system
2. Starting EcSimulatorHilDemo

```
$ cd /opt/EC-Simulator/Bin/Linux/x64
$ LD_LIBRARY_PATH=. ./EcSimulatorHilDemo -f exi.xml -intelgbe 2 1 -perf
```

Microsoft Windows (EcSimulatorHilDemo)

1. Windows configuration: See the section Operating system configuration for how to prepare the operating system
2. Determine the network interface: Using the command line option the network interface card used by the example application can be determined. For example the option `-ndis 192.168.157.2 0` will be using the network adapter card with the IP address 192.168.157.2 .
3. Simulating EtherCAT modules: The simulator has to be connected with the target system using an Ethernet switch or a patch cable. Local IT infrastructure should not be mixed with EtherCAT modules at the same switch as the master will send many broadcast packets! EtherCAT® requires a 100Mbit/s connection. Gigabit-Ethernet connections between simulator and master can be typically used, too.
4. Copy all of the example application files into one directory: The application EcSimulatorHilDemo.exe together with the stack DLL EcSimulator.dll, the Ethernet Driver DLL emllPcap.dll and the EXI file have to be copied into one directory.
5. Run the example application: EcSimulatorHilDemo.exe has to be executed. The file name of the EXI file has to be given as a command line parameter as well as the appropriate Real-time Ethernet Driver settings.

```
$ EcSimulatorHilDemo.exe -ndis 192.168.157.2 0 -f exi.xml -t 0 -v 3
```

2.7.2 Command line parameters

```
EcSimulatorHilDemo <LinkLayer> [-f ENI-FileName] [-t cycle time] [-b time] [-a affinity] [-v level] [-perf [level]] [-log prefix [msg cnt]] [-lic key] [-oem key] [-flash address] [-sp [port]] [-rec [prefix [frame cnt]]]
```

The parameters are as follows:

- f** <configFileName>
Path to ENI file
- t** <time>
Running duration in msec. When the time expires the demo application exits completely.
<time>
Time in msec, 0 = forever (default = 120000)
- b** <cycle time>
Specifies the bus cycle time. Defaults to 1000 μ s (1 ms).
<cycle time>
Bus cycle time in μ sec
- a** <affinity>
The CPU affinity specifies which CPU the demo application ought to use.
<affinity>
0 = first CPU, 1 = second, ...
- v** <level>
The verbosity level specifies how much console output messages will be generated by the demo application. A high verbosity level leads to more messages.
<level>
Verbosity level: 0=off (default), 1..n=more messages
- perf** [<level>]
Enable max. and average time measurement in μ s for all EtherCAT jobs (e.g. ProcessAllRxFrames).
<level>
Depending on level the performance histogram can be activated as well.
- log** <prefix> [<msg cnt>]
Use given file name prefix for log files.
<prefix>

<msg cnt>
Messages count for log buffer allocation
- lic** <key>
Use License key.
<key>
26 characters long license key.
- oem** <key>
Use OEM key
<key>
64 bit OEM key.
- flash** <address>
Flash outputs
<address>
0=all, >0 = slave station address

-sp [<port>]
 If platform has support for IP Sockets, this command-line option enables the Remote API Server to be started. The Remote API Server is going to listen on TCP Port 6000 (or port parameter if given) and is available for connecting Remote API Clients.

<port>
 RAS server port

-rec [<prefix> [<frame cnt>]]
 Packet capture file recording

<prefix>
 File name prefix

<frame cnt>
 Frame count for log buffer allocation

Link Layer

Using one of the following demo application Link Layer options, the EC-Simulator will dynamically load the network driver for the specified network adapter card and use the appropriate network driver to access the Ethernet adapter for EtherCAT®. `ShowSyntaxLinkLayer()` in `Common/EcSelectLinkLayer.cpp` is called automatically if the Demo application is started without parameters and lists the possibilities.

Note: Not all link layers are available on all operating systems or architectures. A detailed view in the form of a matrix can be found in the [developer center](#).

-alteratse <instance> <mode>

<instance>
 Device instance 1 = first, 2 = second, ...

<mode>
 0 = Interrupt mode | 1 = Polling mode

-antaios
 Device instance fixed to 2

Mode fixed to 1 = Polling mode

-bcmgenet <instance> <mode>

Hardware: Broadcom BcmGenet

<instance>
 Device instance 1 = first, 2 = second, ...

<mode>
 0 = Interrupt mode | 1 = Polling mode

-bcmnetxtreme <instance> <mode>

Hardware: Broadcom NetXtreme

<instance>
 Device instance 1 = first, 2 = second, ...

<mode>
 0 = Interrupt mode | 1 = Polling mode

-bpf <instance> <mode> <interface>

Hardware: Berkeley Packet Filter, Hardware independent

<instance>
BPF instance (0=first), results to e.g. /dev/bpf0

<mode>
0 = Interrupt mode | 1 = Polling mode

<interface>
Name of Ethernet Interface, e.g. wm0

Optional:

<prefix>
Prefix of the BPF instance path, e.g. /alt

-ccat <instance> <mode>

Hardware: Beckhoff CCAT

<instance>
Device instance 1 = first, 2 = second, ...

<mode>
0 = Interrupt mode | 1 = Polling mode

-cpsw <instance> <mode> <portpriority> <masterflag> <refboard>

Hardware: TI CPSW

<instance>
Device instance 1 = first, 2 = second, ...

<mode>
0 = Interrupt mode, 1 = Polling mode

<portpriority>
Low priority (0) or high priority (1)

<masterflag>

(m) Master (Initialize Switch), (s) Slave

<RefBoard>
bone | am3359-icev2 | am437x-idk | am572x-idk | 387X_evm | custom | osdriver

If custom:

<CpswType>
am33XX | am437X | am57X | am387X

<PhyAddress>
0 ... 31

<PhyInterface>
rmii | gmii | rgmii | osdriver

<NotUseDmaBuffers>
0 = FALSE | 1 = TRUE

-dw3504 <instance> <mode> <RefBoard>

Hardware: Synopsys DesignWare 3504-0 Universal 10/100/1000 Ethernet MAC (DW3504)

<instance>

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode

<RefBoard>

Reference Board: intel_atom | lces1 | rd55up06 | r12ccpu | rzn1 | socrates | stm32mp157a-dk1 | custom

If custom:**<DW3504Type>**

intel_atom | cycloneV | lces1 | stm32mp157a-dk1

<PhyInterface>

fixed | mii | rmii | gmii | sgmii | rgmii | osdriver

<PhyAddress>

0 ... 31 (don't use if osdriver)

-eg20t <instance> <mode>**Hardware: Intel EG20T Gigabit Ethernet Controller****<instance>**

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode

-emac <instance> <mode> <refboard>**Hardware: Xilinx LogiCORE IP XPS EMAC****<instance>**

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode

<RefBoard>

MC2002E | custom

If custom:**<RegisterBase>**

Register base address as hex value

<RegisterLength>

Register length as hex value

<NotUseDmaBuffers>

0 = FALSE | 1 = TRUE

-fs1etsec <instance> <mode> <refboard>**Hardware: Freescale TSEC / eTSEC V1 / eTSEC V2 (VeTSEC)****<instance>**

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode

<RefBoard>
p2020rdb | twrp1025 | istmpc8548 | xj_epu20c | twrls1021a | tqmls_ls102xa | custom

If custom:

<PhyAddress>

0 ... 31

<RxIrq>

Default depending on ETSEC type

<NotUseDmaBuffers>

0 = FALSE | 1 = TRUE

-fslfec <instance> <mode> <refboard>

Hardware: Freescale FEC/ENET

<instance>

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode

<RefBoard>

mars | sabrelite | sabresd | imx28evk | topaz | imxceetul2 | mimxrt1064-evk | imx93evk | custom

If custom:

<FecType>

imx25 | imx28 | imx53 | imx6 | vf6 | imx7 | imx8 | imx8m | imxrt1064 | imx9

<PhyInterface>

fixed | mii | rmii | rmii50Mhz | gmii | sgmmii | rgmmii | osdriver

<PhyAddress>

0 ... 31, default 0 (don't use if osdriver)

Optional:

<nopinmuxing>

no pin muxing

<nomacaddr>

don't read MAC address

-gem <instance> <mode> <refboard>

Hardware: Xilinx Zynq-7000/Ultrascale (GEM)

<instance>

Device instance for GEM, 1 = GEM0,...

<mode>

0 = Interrupt mode | 1 = Polling mode

<RefBoard>

zc702 | zedboard | microzed | zcu102 | zcu104 | KR260 | custom

If custom:

<PhyAddress>

0 ... 31

<PhyConnectionMode>

MIO (0) or EMIO (1)

<UseGmiiToRgmii>

Use Xilinx GmiiToRgmii converter TRUE (1) or FALSE (0)

<GmiiToRgmiiPort>

GmiiToRgmii converter PHY address 0 ... 31

<GemType>

zynq7000 or ultrascale

Optional:**<PhyInterface>**

osdriver

<ClkDivType_K26>

Clock divisor

<Pnopinmuxing>

Don't use pin muxing

-i8254x <instance> <mode>**Hardware: Intel Pro/1000 network adapter card****<instance>**

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode

-i8255x <instance> <mode>**Hardware: Intel Pro/100 network adapter card****<instance>**

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode

-

icss <instance> <mode> <masterflag> <refboard> [<PhyInterface> <PhyAddress>] [<NoPhyReset> cleTime TtsSendOffset]

Hardware: Texas Instruments Board with PRUSS**<instance>**

ICSS Port (100 Mbit/s) 1 ... 4

<mode>

0 = Interrupt mode | 1 = Polling mode

<MasterFlag>

(m) Master (Initialize board, mdio, both phy) or (s) Slave

<RefBoard>

am572x-idk | am571x-idk | am3359-icev2 | am574x

Optional:

<PhyInterface>
mii | osdriver

<PhyAddress>
0 ... 31 (only for mii)

Optional:

<NoPhyReset>
NoPhyReset

Optional:

<TtsEnable>
tts

<TtsCycleTime>
TTS cycle time (usec)

<TtsSendOffset>
TTS cyclic frame send offset from cycle start (usec)

-icssg <instance> <mode> <masterflag> <refboard>

Hardware: Texas Instruments AARCH64 Board with Gigabit PRUSS

<instance>
ICSSG Port 1 ... 6

<mode>
0 = Interrupt mode | 1 = Polling mode

<MasterFlag>

(m) Master (Initialize board, mdio, both phy) or (s) Slave

<RefBoard>
am654x-idk

-19218i <instance> <mode>

Hardware: SMSC LAN9218i/LAN9221

<instance>
Device instance 1 = first, 2 = second, ...

<mode>
0 = Interrupt mode | 1 = Polling mode

-lan743x <instance> <mode>

Hardware: Microchip LAN743x

<instance>
Device instance 1 = first, 2 = second, ...

<mode>
1 = Polling mode

-ndis <IpAddress> <mode>

Hardware: Hardware independent, only available for Windows.

<IpAddress>

IP address of network adapter card, e.g. 192.168.157.2 or 0.0.0.0 if name given

<mode>

0 = Interrupt mode | 1 = Polling mode

Optional:**--name**

Adapter name. Service name from HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkCards

<DisablePromiscuousMode>

Disable promiscuous mode

<DisableForceBroadcast>

-multiplier <instance> <mode> [--type <type>] --port <port> --link <link parms>

Hardware: Beckhoff CUxxxx Ethernet-Port-Multiplier**<instance>**

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode, for now only polling mode is supported

<port>

used CU2508 downlink port 0 = X1, 1 = X2, ...

<link parms>

link parms of network adapter connected to the uplink port e.g. -intelgbe ...

Optional:**<type>**

cu2508 = CU2508 Ethernet-Port-Multiplier

-r6040 <instance> <mode>

Hardware: RDC R6040**<instance>**

Device instance 1 = first, 2 = second, ...

<mode>

1 = Polling mode

-rin32m3 <instance> <mode>

Hardware: Renesas R-IN32M3-EC**<instance>**

Device instance 1 = first, 2 = second, ...

<mode>

1 = Polling mode

-rt18139 <instance> <mode>

Hardware: Realtek RTL8139

<instance>

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode

-rtl8169 <instance> <mode>

Hardware: Realtek RTL8168 / RTL8169 / RTL8111

<instance>

Device instance 1 = first, 2 = second, ...

<mode>

0 = Interrupt mode | 1 = Polling mode

-rzt1 <instance>

Hardware: Renesas RZ/T1

<instance>

Device instance 1 = Port 0 | 2 = Port 1

-sheth <instance> <mode> <RefBoard>

Hardware: Renesas RZG1 or Armadillo-800 EVA

<instance>

Device instance 1 = first, 2 = second, ...

<mode>

1 = Polling mode

<RefBoard>

rzg1e | a800eva

-snarf <adapterName>

Hardware: Hardware independent, only available for VxWorks

<adapterName>

Adapter name, e.g. fei0

-sockraw <device>

Hardware: Hardware independent, only available for Linux.

<device>

Network device, e.g. eth1

Optional:

<mode>

0 = Interrupt mode | 1 = Polling mode

--nommaprx

Disable PACKET_MMAP for receive

-stm32eth <instance> <mode>

Hardware: STM32H7 Ethernet

<instance>

Device instance 1=first, 2=second, ...

<mode>
1 = Polling mode

-tap <instance> <mode> <adapterName>

OpenVPN's Windows TAP

<instance>
Device instance 1=first, 2=second, ...

<mode>
0 = Interrupt mode | 1 = Polling mode

<adapterName>
Adapter name

-winpcap <ipAddress> <mode>

Hardware: Hardware independent, only available for Windows.

<ipAddress>
IP address of network adapter card, e.g. 192.168.157.2

<mode>
0 = Interrupt mode | 1 = Polling mode

2.8 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

2.8.1 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`

struct *_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 **dwBusCycleTimeUsec**

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 `emIISimulator`

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>]
```

2.9 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.**
 -SDKINC: header files to be included with the application
 -SDKLIB: libraries to be linked with the application
- ...SourcesCommon: 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 two directories:

1. <InstallPath>Examples<ExampleName> (with e.g. <ExampleName>=EcSimulatorHilDemo)
2. <InstallPath>ExamplesCommon<OS> (where <OS> is a placeholder for the operating system)
3. <InstallPath>ExamplesCommon
4. <InstallPath>SDKINC<OS> (where <OS> is a placeholder for the operating system)
5. <InstallPath>SDKINC
6. <InstallPath>SourcesCommon

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>SDKLIB<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).

2.9.1 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`):

`armv4t-eabi`, `armv6-vfp-eabi`, `x64` (aka `amd64`), `x86` (aka `i686`), `PPC` (with “`-te500v2`”)

`armv4t-eabi` and `armv6-vfp-eabi` are incompatible with each other. A potentially `armv6-vfp-eabi`-compatible 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`

2.10 Real-time Ethernet Driver configuration for EcSimulatorHilDemo

The different Real-time Ethernet Driver modules are selected and parametered by a common structure (shared by all Real-time Ethernet Driver) and a Real-time Ethernet Driver specific structure, pointed to by an element within the common structure. This parameter set is given to the EC-Simulator stack with the call of `esInitSimulator()`.

See also:

`esInitSimulator()`

3 Software Integration

3.1 EcSimulatorHilDemo

The example application EcSimulatorHilDemo will handle the following tasks:

- Stack initialization
- Flash process data
- **Out of the box solution for different operating systems:**
 - Windows
 - Linux
 - LxWin
 - ...
- Example implementation for polled mode or interrupt 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	Entrypoint for the different operating systems
EcDemoPlatform.h	Operating system specific settings (task priorities, timer settings)
EcDemoApp.cpp	Initialize, start and terminate the EC-Simulator stack
EcDemoApp.h	Application specific settings for EcSimulatorHilDemo
EcDemoParms.cpp	Parsing of command line parameters and generic helper functions
EcDemoParms.h	Basic configuration parameters
EcSelectLin- kLayer.cpp	Common Functions which abstract the command line parsing into Real-time Ethernet Driver parameter
EcNotification.cpp	Slave monitoring and error detection (function <code>emNotify()</code>)
EcSlaveInfo.cpp	Slave information services
EcLogging.cpp	Message logging functions
EcTimer.cpp	Start and monitor timeouts

3.1.2 Simulator lifecycle

This chapter gives brief information about the starting and stopping of the EC-Simulator.

Basically the operation of the EC-Simulator stack is wrapped between the functions

- `esInitSimulator()`
- `esConfigureNetwork()`
- `esDeinitSimulator()`

3.1.3 Logging

The Example programs cpp demonstrates how messages can be processed by the application, see ExamplesCommonEcLogging.cpp.

The messages handled by EcLogging.cpp are of different type, e.g. EC-Simulator 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 EcSimulatorHilDemoMain.cpp. Logging is configured on initialization, see the structure documentation of EC_T_SIMULATOR_INIT_PARMS::LogParms in *esInitSimulator()*. The EC-Simulator automatically filters log messages according to EC_T_SIMULATOR_INIT_PARMS::LogParms.dwLogLevel. Log messages which are less relevant according 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, EcSimulatorHilDemoMain.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.

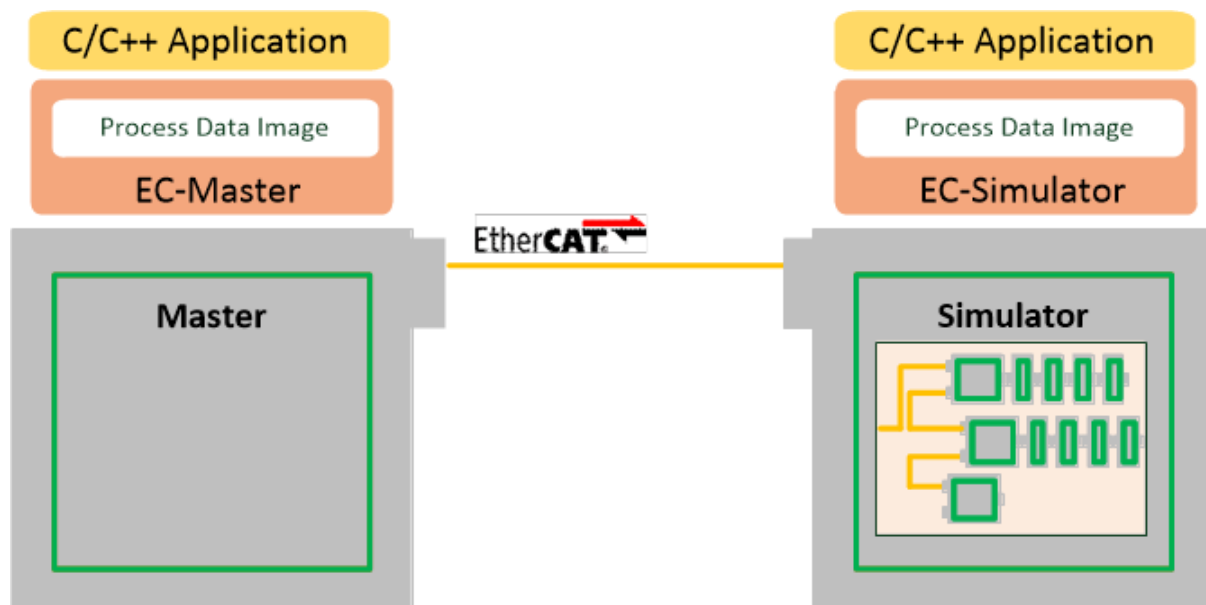
Important: The callback is typically called from the Job Task’s context and should return as fast as possible.

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.

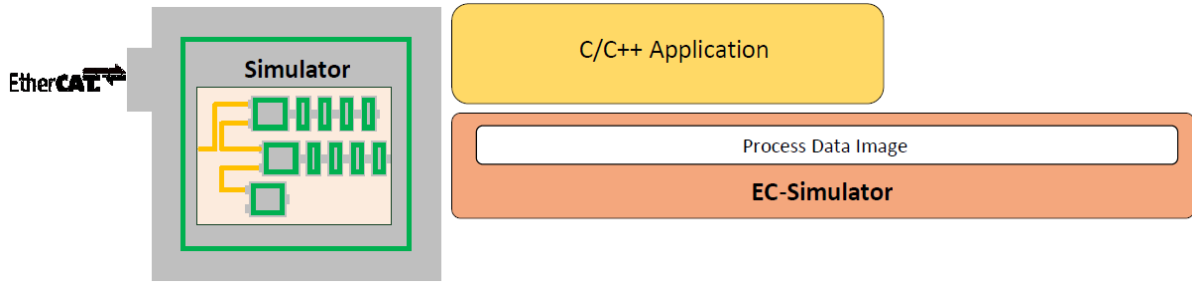
Identical messages are skipped automatically by default.

3.2 Process data update and synchronization

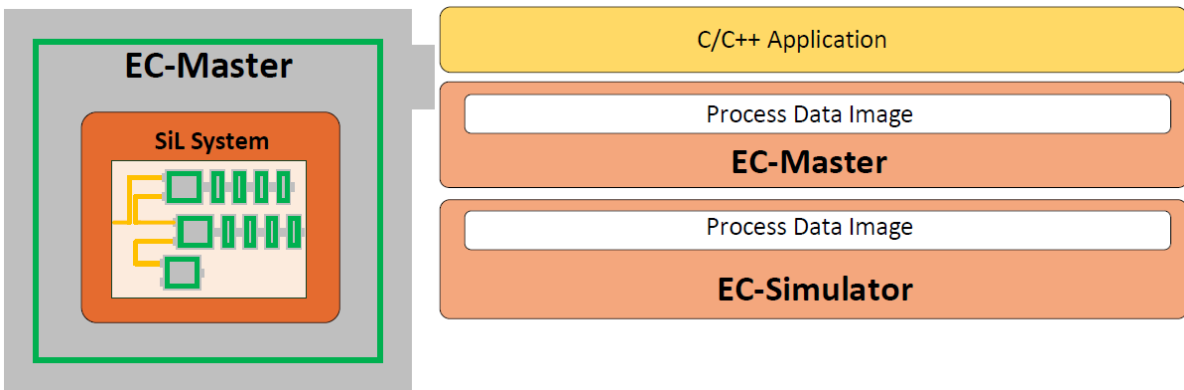
The EtherCAT® Master cyclically sends frames containing the outputs of the Process Data Image to the EtherCAT® network and the Simulator synchronizes the Process Data Image with the frames from the Master and sends the response back to the Master. The Master cyclically receives the inputs of the Process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



The HiL Application directly accesses the Process Data Image at Simulator:

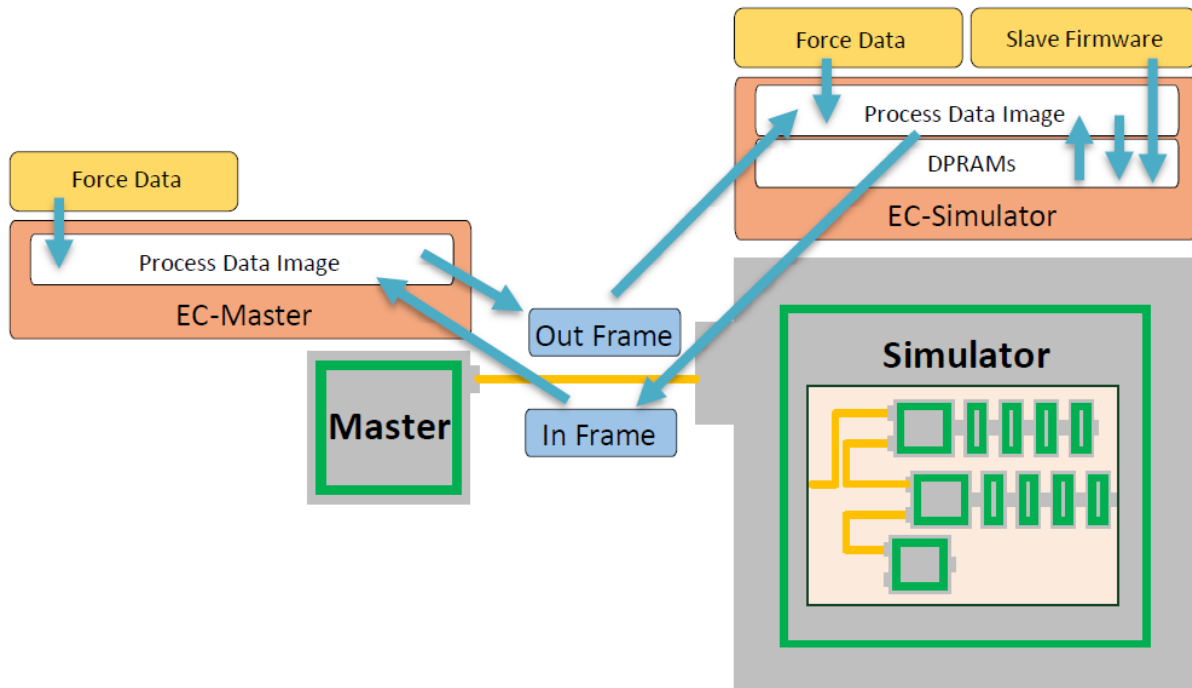


The SiL Application directly accesses the Process Data Image at Master and Simulator:



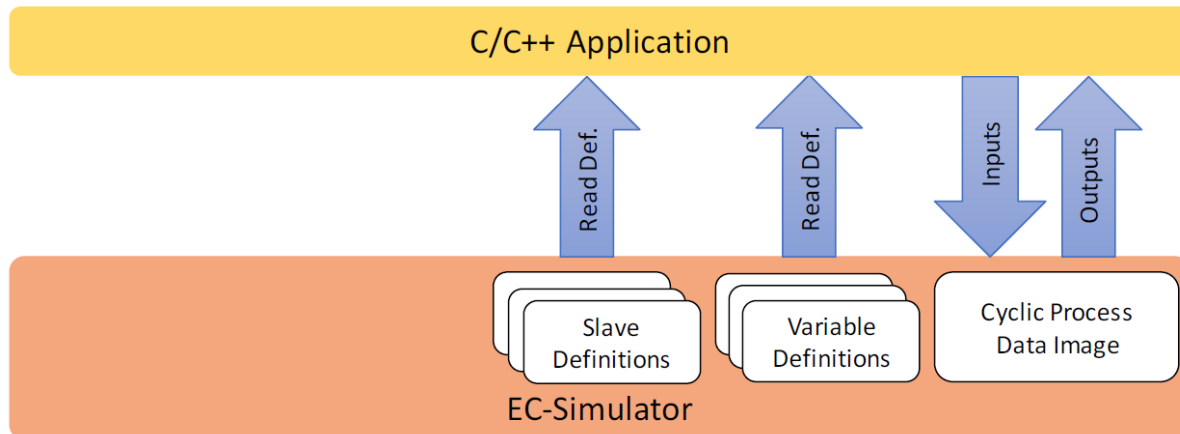
3.2.1 Forced Data, Firmware Data, DPRAM

The EC-Simulator stack synchronizes the outputs and inputs from the Process Data Image and the slave 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:

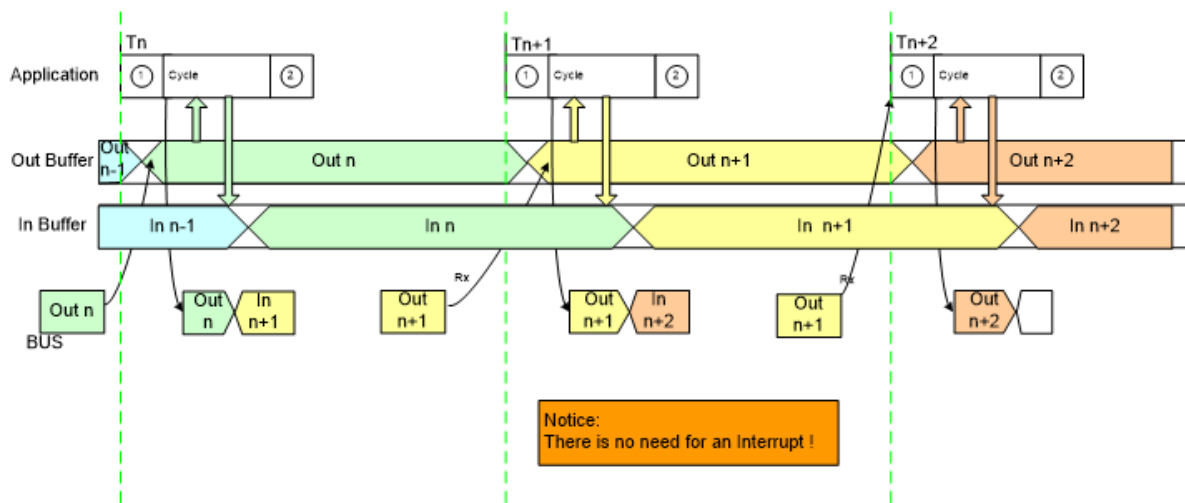
- **eUsrJob_ProcessAllRxFrames (polling mode only)**
Process all received frames from Real-time Ethernet Driver
- **eUsrJob_SimulatorTimer**
Trigger state machines

Important: In polling mode, the master cycle time must be at least two times higher than the simulator cycle time. E.g. if the simulator runs with 1 ms, the master 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 slaves 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 `esIoControl - EC_IOCTL_GET_PDMEMORYSIZE` and `esRegisterClient()` at `EC_T_REGISTERRESULTS::dwPDOutSize` and `EC_T_REGISTERRESULTS::dwPDInSize`.

See also:

- `esGetProcessImageOutputPtr()`
- `esGetProcessImageInputPtr()`
- `esIoControl - 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 array of bits are bit aligned and not byte aligned in process data.

See also:

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

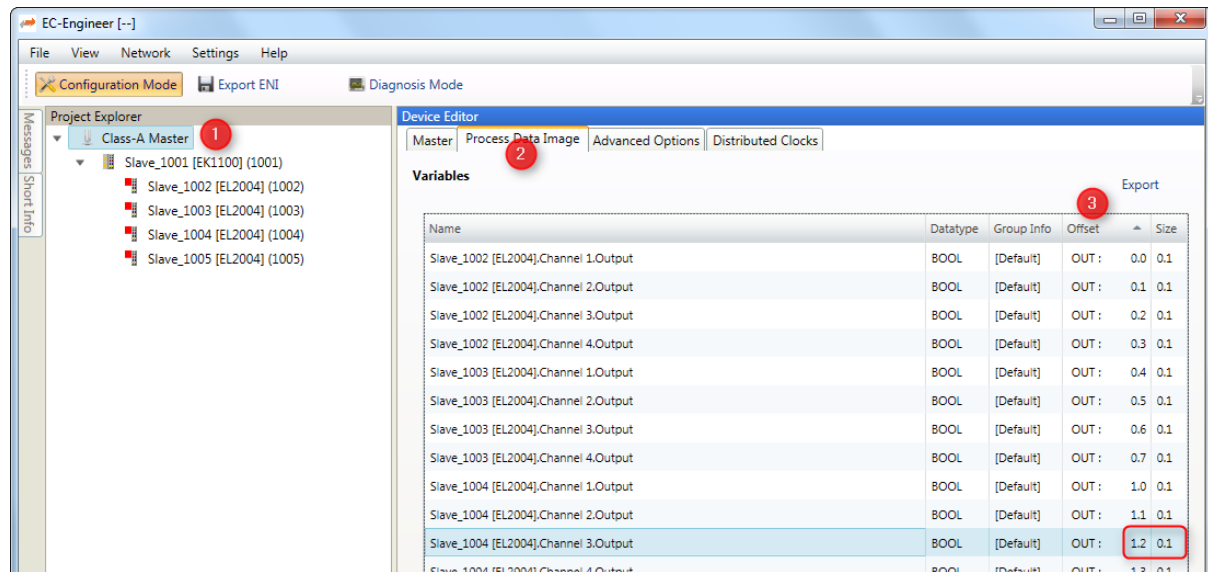
Getting and setting bits that are bit aligned and not byte aligned should be done using `EC_SETBITS` and `EC_GETBITS`. Accessing complete `EC_T_BYTE`, `EC_T_WORD`, `EC_T_DWORD`, `EC_T_QWORD` can be accessed more efficiently using the appropriate macros according to the following table.

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

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

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 slave 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 changes in case of adding/removing slaves to/from the configuration.

As listed in the screenshot above "Slave_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 $8 \cdot 1 + 2 = 10$ and size is $0 \cdot 8 + 1 = 1$.

Sample code:

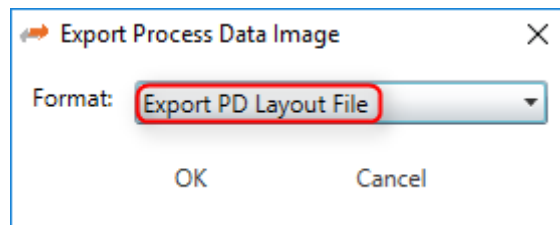
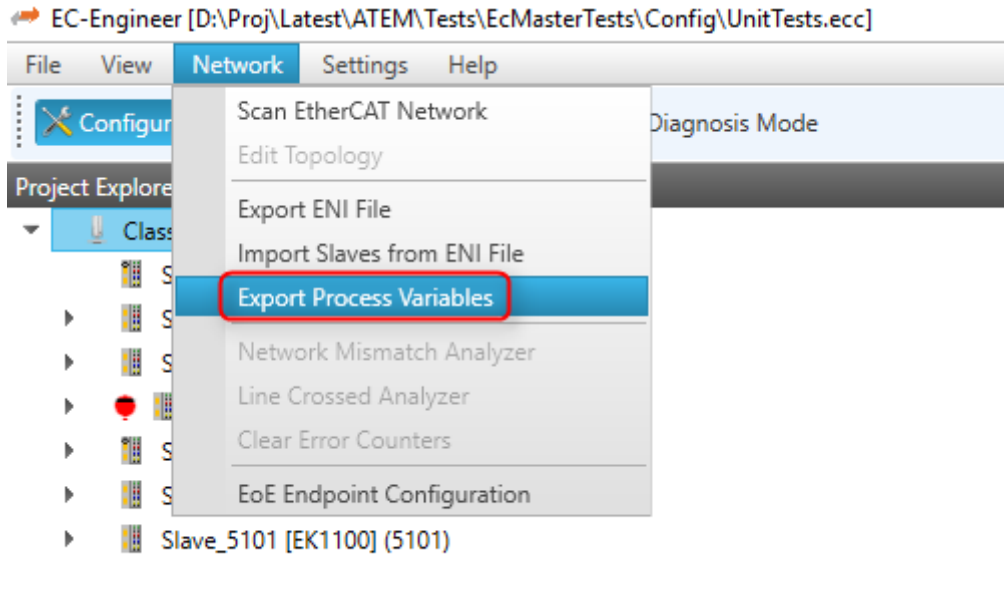
```
EC_T_BYTE byNewValue = 0x01;

/* 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 slaves' 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);
```

Slave / variable offset from configuration

esGetCfgSlaveInfo

Figure out the slave 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 "Slave_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 slave 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 variables names are stored in the ENI file under “EtherCATConfig/Config/ProcessImage/Inputs/Variable”. E.g. setting “Slave_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, "Slave_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 slaves 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 Master 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 slave needs to be power cycled.

The following examples 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 */
```

(continues on next page)

(continued from previous page)

```
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 Error detection and diagnosis

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

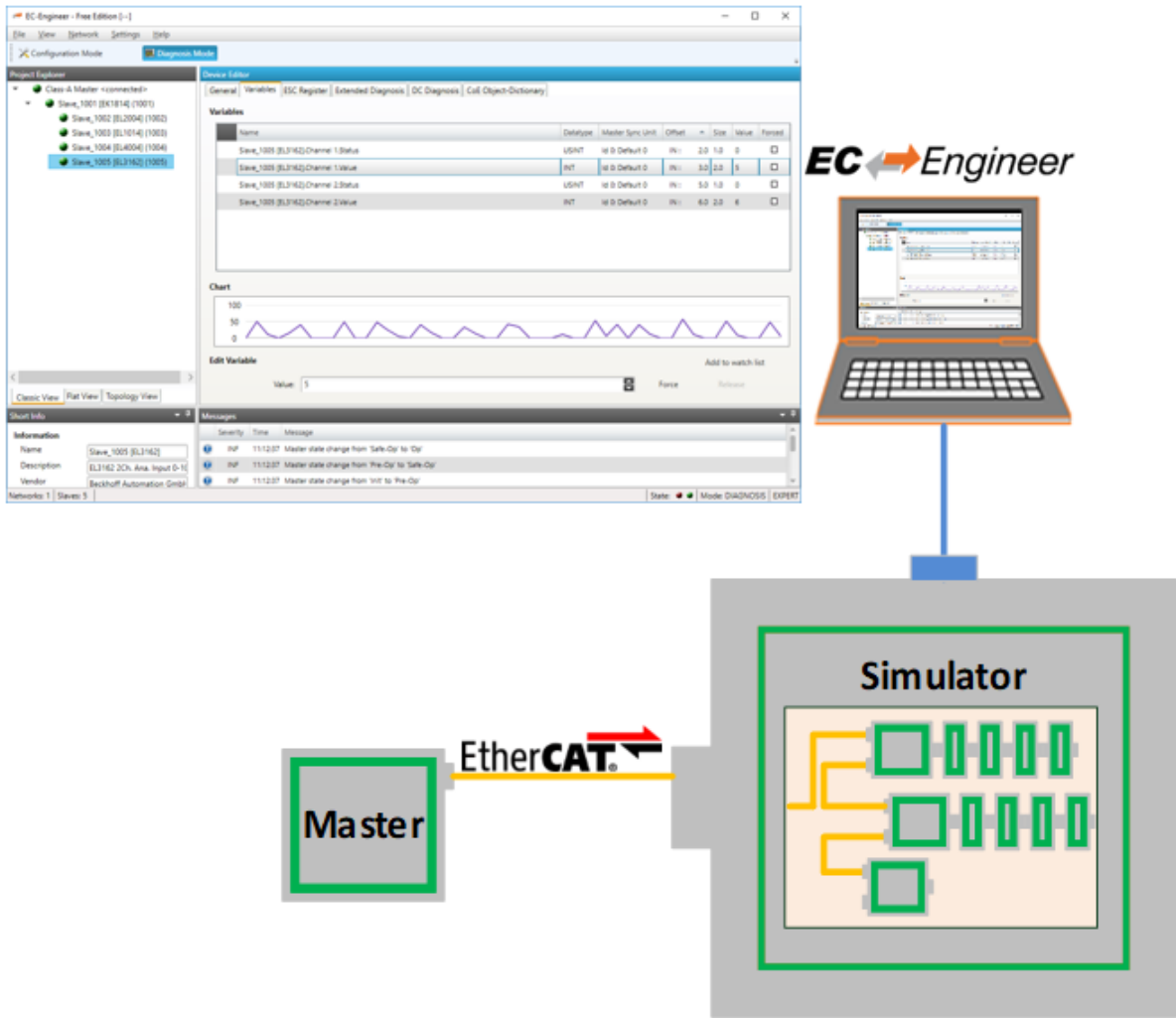
Messages are logged as described in *esInitSimulator()*.

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

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

3.6.1 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 in case is preconfigured to listen for EC-Lyser on TCP Port 6000 when commandline parameter “-sp” is given.

To clarify the steps, which are 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.

3.6.2 Pseudo Code

```
#include "EcRasServer.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* pParams       /**< [in]   Notification data portion */
)
{
    /* custom notification handler */
}

/* initialization */
ECMASTERRAS_T_SRVPARMS oRemoteApiConfig = {0};
```

(continues on next page)

(continued from previous page)

```

EC_T_PVOID          pvRemApiHandle;
oRemoteApiConfig.dwSignature          = ECMASSTERRASSERVER_SIGNATURE;
oRemoteApiConfig.dwSize               = sizeof(ECMASTERRAS_T_SRVPARMS);
/* INADDR_ANY */
oRemoteApiConfig.oAddr.dwAddr         = 0;
/*< default is port 6000 > */
oRemoteApiConfig.wPort                = wServerPort;
/*< default is 2 msec */
oRemoteApiConfig.dwCycleTime          = ECMASSTERRAS_CYCLE_TIME;
oRemoteApiConfig.dwCommunicationTimeout = ECMASSTERRAS_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(...);
}

```

3.6.3 Required API Calls

esRasSrvStart

```

EC_T_DWORD EC_NAMESPACE::esRasSrvStart (
    ECMASSTERRAS_T_SRVPARMS *pParms,
    EC_T_PVOID *ppHandle
)

```

Initializes and start 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 ECMASSTERRAS_T_SRVPARMS
```

Public Members

EC_T_DWORD dwSignature

[in] Set to ECMASTERRASSERVER_SIGNATURE

EC_T_DWORD dwSize

[in] Set to sizeof(ECMASTERRAS_T_SRVPARMS)

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 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

union **EC_T_IPADDR**

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 - ECMASERRAS_NOTIFY_CONNECTION

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

emRasNotify - ECMASERRAS_NOTIFY_CONNECTION

Parameter

- pbyInBuf: [in] Pointer to data of type ECMASERRAS_T_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 **ECMASERRAS_T_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 - ECMASERRAS_NOTIFY_REGISTER

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

emRasNotify - ECMASERRAS_NOTIFY_REGISTER

Parameter

- pbyInBuf: [in] Pointer to data of type ECMASERRAS_T_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 **ECMASERRAS_T_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 - ECMASERRAS_NOTIFY_UNREGISTER

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

emRasNotify - ECMASERRAS_NOTIFY_UNREGISTER

Parameter

- pbyInBuf: [in] Pointer to data of type ECMASERRAS_T_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 - ECMASERRAS_NOTIFY_REGISTER

RasNotifyCallback - ECMASERRAS_NOTIFY_MARSHALERROR

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

emRasNotify - ECMASERRAS_NOTIFY_MARSHALERROR

Parameter

- pbyInBuf: [in] Pointer to data of type ECMASERRAS_T_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 **ECMASERRAS_T_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 faulty command

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

RasNotifyCallback - ECMASTERRAS_NOTIFY_ACKERROR

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

emRasNotify - ECMASTERRAS_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 - ECMASTERRAS_NOTIFY_NONOTIFYMEMORY

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

emRasNotify - ECMASTERRAS_NOTIFY_NONOTIFYMEMORY

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

RasNotifyCallback - ECMASERRAS_NOTIFY_STDNOTIFYMEMORYSMALL

Notification given, when buffersize for standard notifications available in pre-allocated notification store are too small to carry a specific notification. This points to a configuration error.

RasNotifyCallback - ECMASERRAS_NOTIFY_MBXNOTIFYMEMORYSMALL

Notification given, when buffersize for Mailbox notifications available in pre-allocated notification store are too small to carry a specific notification. This points to a configuration error.

emRasNotify - ECMASERRAS_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 is given, Mailbox Transfer objects may have been become out of sync and therefore no more valid usable.

See also:

esRasSrvStart () for Mailbox notifications should be dimensioned correctly.

3.7 EC-Simulator stack Source Code

In a source code delivery the stack sources are divided into 4 parts:

- SDK Header files
- Real-time Ethernet Driver files (multiple Real-time Ethernet Driver may be shipped)
- Link OS layer files (only valid for the Real-time Ethernet Driver)
- stack files (configuration, core and interface layer)
- OS layer files (only valid for the stack)

The stack can be ported to several different operating systems and CPU architectures with different compilers and development environments. Typically no supported build environment files like IDE projects are shipped with the source code.

To build the stack the appropriate build environment for the target operating system has to be used. If an integrated development environment (IDE) exists (Visual Studio, Eclipse, etc.) several projects containing all necessary files are needed to build the artefacts. If no integrated development environment is available makefiles and dependency rules may have to be created which contain the necessary stack source and header files.

For most platforms three separate independent binaries will have to be generated: #. Real-time Ethernet Driver Binary (e.g. a DLL in Windows). The Real-time Ethernet Driver binary will be dynamically bound to the application at runtime. (currently not for On Time RTOS-32 which uses static libraries) #. Stack Library #. Remote API Server Library

Real-time Ethernet Driver Libraries: the following files have to be included into an IDE project or makefile:

- Real-time Ethernet Driver files. Only one single Real-time Ethernet Driver must be selected even if multiple Real-time Ethernet Driver are shipped. For each Real-time Ethernet Driver a separate binary has to be created.
- Link OS layer files
- Windows: a dynamic link library (.dll) has to be created. The name of the DLL has to be emllXxxx.dll where Xxxx shall be replaced by the Real-time Ethernet Drivertype (e.g. emllI8255x.dll for the I8255x Real-time Ethernet Driver).
- Linux/QNX: a shared object library (.so) has to be created.
- RTX a RTX dynamic link library (.rtdll) has to be created. The name of the DLL has to be emllXxxx.dll where Xxxx shall be replaced by the Real-time Ethernet Driver type (e.g. emllI8255x.dll for the I8255x Real-time Ethernet Driver).
- INtime: a shared library (.rsl) has to be created. The name of the RSL has to be emllXxxx.rsl where Xxxx shall be replaced by the Real-time Ethernet Driver type (e.g. emllI8255x.rsl for the I8255x Real-time Ethernet Driver).

Stack Libraries: the following files have to be included into an IDE project or makefile:

- stack files
- OS layer files
- For all platforms a static library has to be created. This library will have to be linked together with the application.

Remote API Server Binaries: the following files have to be included into an IDE project or makefile:

- Remote API server files.
- For all platforms a static library has to be created. This library will have to be linked together with the application.

See also:

OS Compiler settings for required toolchain settings

3.8 EcSimulatorHilDemoMotion

The EcSimulatorHilDemoMotion is an extension of the EcSimulatorHilDemo that handles the following additional tasks:

- Simulating CiA402 capabilities of slaves

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

See also:

EcSimulatorHilDemo

3.8.1 Command line parameters

EcSimulatorHilDemoMotion supports all parameters supported by *EcSimulatorHilDemo* . The following additional parameters are supported:

- “-ds402 <FixedAddress1,...,FixedAddressN>”: Simulate CiA402 capability of slaves. Slaves are passed as comma separated list of fixed Slave addresses to simulate.

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

- Examples\Common\EcSimulatorDs402.h: Simulation of CiA402 capabilities using slave application’s callback functions, see *esSetSlaveSscApplication()* .

4 Platform and Operating Systems (OS)

4.1 tenAsys INtime

Real-time Ethernet Driver are available for INtime. If using INtime with Windows running in parallel on the same host the network adapter card 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 Driver can be adjusted using the PATH environment variable

4.1.1 Setting up and running EcSimulatorHilDemo

The file EcSimulatorHilDemo.rta has to be executed. The full path and file name of the configuration file has to be given as a command line parameter as well as the appropriate Real-time Ethernet Driver. To start the application from the command prompt, enter following commands:

```
> nodemgr start NodeA
> sleep 5
> piperta.exe -node NodeA -stderr EcSimulatorHilDemo.rta -intelgbe 1 0 -f exi.xml
```

See also:

Running EcSimulatorHilDemo

4.1.2 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 a 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
```

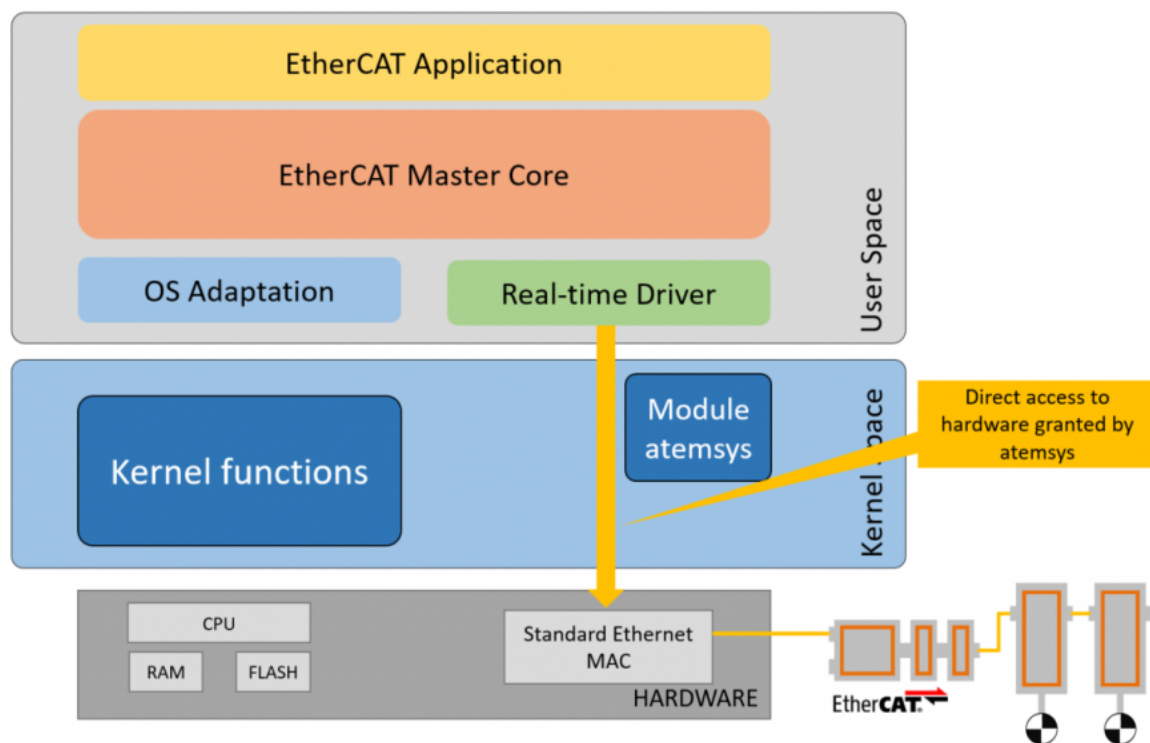
Running EcSimulatorHilDemo on the isolated CPU core by setting the CPU affinity `-a`:

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

4.2.2 atemsys kernel module

To use Real-time Ethernet Driver 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 required 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 drivers 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 adapters. 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	
DesignWare 3504	emllDW3504	stmmac	
	emllEG20T		
Freescall TSEC/eTSEC v1/2	emllETSEC	gianfar_driver	
Freescall FEC and ENET controller	emllFslFec	fec, fec_ptp	
Xilinx Zynq-7000/Ultrascale	emllGEM		
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		

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 Setting up and running EcSimulatorHilDemo

1. Unbind Ethernet Driver instance, e.g.

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

2. Load atemsys kernel module

```
$ insmod atemsys.ko
```

3. Copy files from EC-Simulator package /bin and a `exi.xml` to directory e.g. /tmp.

4. Adjust `LD_LIBRARY_PATH` search locations for Real-time Ethernet Driver if necessary, e.g.

```
$ export LD_LIBRARY_PATH=/tmp:$LD_LIBRARY_PATH
```

5. Run EcSimulatorHilDemo

```
$ cd /tmp
$ ./EcSimulatorHilDemo -f exi.xml -intelgbe 1 0 -perf
```

See also:

[Running EcSimulatorHilDemo](#)

Run in Docker container

1. Unbind Ethernet Driver instance and load atemsys on the host.

2. Create a directory on the host (e.g. `~/docker`) and copy files from EC-Simulator package /bin and `exi.xml` into this directory.

3. Start bash console in container

```
$ sudo docker run -it --name atem_container
↳ --device=/dev/atemsys:/dev/atemsys --ulimit rtprio=99
↳ --cap-add=sys_nice --cap-add=ipc_lock -v ~/docker:/home/docker
↳ -p 6000:6000 ubuntu bash
```

Command line arguments:

- `-it` Allocate a pseudo-TTY and run container
- `--name atem_container` Container name
- `--device=/dev/atemsys:/dev/atemsys` Add *atemsys* device to container
- `--ulimit rtprio=99` Allow max realtime priority
- `--cap-add=sys_nice` Add Linux capability to set priority
- `--cap-add=ipc_lock` Add Linux capability to lock memory
- `-v ~/docker:/home/docker` Mount previously create directory to container
- `-p 6000:6000` Publish RAS server port *6000*
- `ubuntu bash` Use Docker image *ubuntu* and start *bash*

4. Run EcSimulatorHilDemo in container

```
# cd /home/docker
# export LD_LIBRARY_PATH=.
# ./EcSimulatorHilDemo -f exi.xml -intelgbe 1 0 -perf
```

4.2.6 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):

- armv4t-eabi
- armv6-vfp-eabihf
- aarch64
- x64 (aka amd64)
- x86 (aka i686)
- PPC

***armv4t-eabi* and *armv6-vfp-eabihf* are incompatible with each other. A potentially *armv6-vfp-eabihf* compatible system returns success on**

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

Extra include paths

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

4.3 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's about changing the priority range refer to the QNX documentation.

Attention: Don't changing the priority range leads to bad 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 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 em1I8254x with instance number 1

```
smm_em1I8254x-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 Setting up and running EcSimulatorHiIDemo

1. **QNX Neutrino OS configuration**

In order to get real-time priority (e.g. 250), see [Thread priority](#) and also set JOBS_PRIORITY. The applications needs root privileges to increase the priority above 63.

2. **Unbind Ethernet Driver instance, e.g.**

```
$ ifconfig en1 destroy
```

3. Copy files from EC-Simulator package /bin and `exi.xml` to directory, e.g. /tmp.

4. Adjust `LD_LIBRARY_PATH` search locations for Real-time Ethernet Driver if necessary, e.g.

```
$ export LD_LIBRARY_PATH=/tmp:$LD_LIBRARY_PATH
```

5. Run `EcSimulatorHilDemo`

```
$ cd /tmp  
$ ./EcSimulatorHilDemo -f exi.xml -intelgbe 1 0 -perf
```

See also:

Running `EcSimulatorHilDemo`

4.3.5 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
```

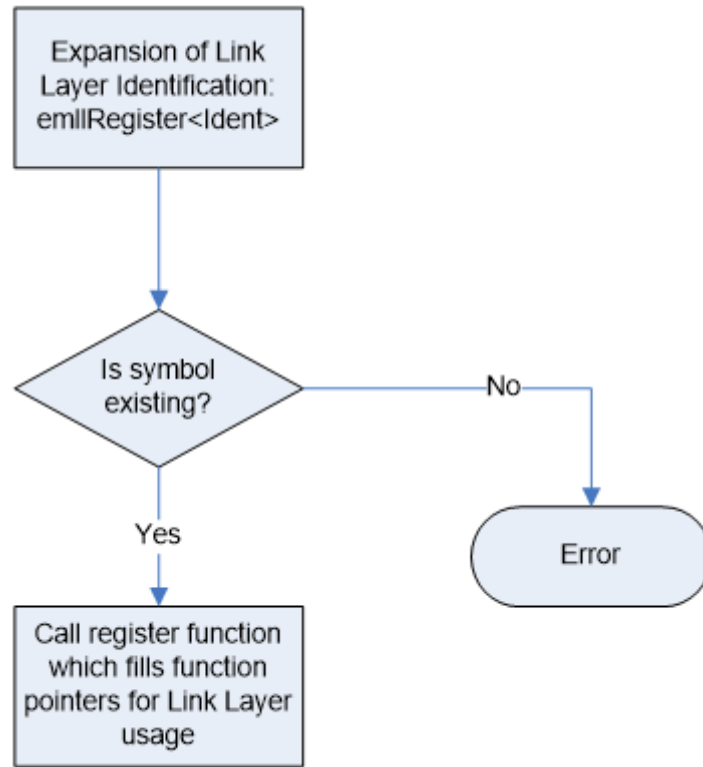
Extra libraries (in this order)

```
EcSimulatorRasServer EcSimulator socket
```

4.4 Windriver VxWorks

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

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



4.4.1 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 device 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 network devices

4.4.2 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.

4.4.3 Setting up and running EcSimulatorHiIDemo

1. VxWorks OS configuration

See sections above.

2. Determine the network interface

Using the command line option the network interface card and Real-time Ethernet Driver to be used in the example application can be determined.

3. Connection of the EtherCAT slaves

The slaves have to be connected with the VxWorks system using an Ethernet switch or a patch cable. Local IT infrastructure should not be mixed with EtherCAT modules at the same switch as the EC-Simulator will send many broadcast packets! EtherCAT requires a 100Mbit/s connection. If the VxWorks network adapter card does not support this speed an 100Mbit/s (!) Ethernet switch has to be used.

4. Download an Real-time Ethernet Driver module

The Real-time Ethernet Driver library (e.g. `emllIntelGbe.out`) which contains hardware support for the corresponding NIC must be downloaded. By default the Ethernet Driver `emllSnarfGpp` are contained with the binary delivery.

5. Download the example application

The target has to be started and a target-server connection will have to be established. After this the example application can be downloaded into the target.

6. Set up a FTP server connection on host

The demo application needs to load a XML file (`exi.xml`) for the configuration of the master. This file can be accessed using a FTP server. The screen shot below show, how to configure the FTP server. The directory contents can be checked via FTP using the `ls` command. The file `exi.xml` will have to be accessed using the default directory.

7. Check for exclusive hardware access

Be sure that the network adapter instance dedicated to EtherCAT is not controlled by a VxWorks driver, this can be verified using:

```
-> muxShow
```

If it is needed, first unload the driver using: (e.g. first instance of the Intel Pro/100):

```
-> muxDevUnload "fei", 1
```

(e.g. second instance of the Intel Pro/1000):

```
-> muxDevUnload "gei", 2
```

(e.g. first instance of the Realtek 8139):

```
-> muxDevUnload "rtl", 1
```

(e.g. first instance of the Realtek 8169):

```
-> muxDevUnload "rtg", 1
```

(e.g. first instance of the FEC on Freescale iMX platform):

```
-> muxDevUnload "motfec", 1
```

(e.g. first instance of the ETSEC on Freescale PPC platform):

```
-> muxDevUnload "motetsec", 1
```

8. Run the example application

The downloadable module `EcSimulatorHilDemo.out` has to be executed. The configuration file `exi.xml` will be used and thus has to be accessible in the current working directory. The appropriate Real-time Ethernet Driver and network adapter card have to be selected. If the log files shall be written the global variable `bLogFileEnb` has to be set to 1 prior to starting the demo.

Loading and running the demo:

```
-> ld<EcSimulatorHilDemo.out
-> sp EcMasterAppMain, "-f exi.xml -intelgbe 1 0 -v 2"
```

See also:

Running EcSimulatorHilDemo

4.4.4 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>
```

VxWorks V6.1 .. V6.4: In case the SNARF Ethernet Driver shall be used and `ifconfig()` is available `-D IFCONFIG_SUPPORT` has to be set.

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

4.5 Microsoft Windows

4.5.1 EcSimulatorHilDemo - Microsoft Windows

1. Install EC-Simulator

Run `setup.exe` from EC-Simulator package, which will guide you through the installation process.

2. Determine the network interface

For example the option `-ndis 192.168.1.1 0` will be using the network adapter card with the IP address 192.168.1.1.

3. Connect EtherCAT Master

The EtherCAT Master can be directly connected to the target system. In rare cases an interposing Ethernet switch helps reducing frame loss on Master startup due to triggered auto-negotiation.

Warning: The local IT infrastructure should not be mixed with EtherCAT modules on the same Ethernet adapter. The EtherCAT Master sends many broadcast packets!

4. Run the example application

Execute `<InstallPath>\Bin\Windows\<Arch>\EcSimulatorHilDemo.exe`, with x86 or x64 for `<Arch>` accordingly. At least the (Real-time) Ethernet Driver option has to be given.

```
C:
> EcSimulatorHilDemo -ndis 192.168.1.1 0 -f exi.xml
```

See also:

[Running EcSimulatorHilDemo](#) for a detailed description of the demo application.

4.5.2 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
```

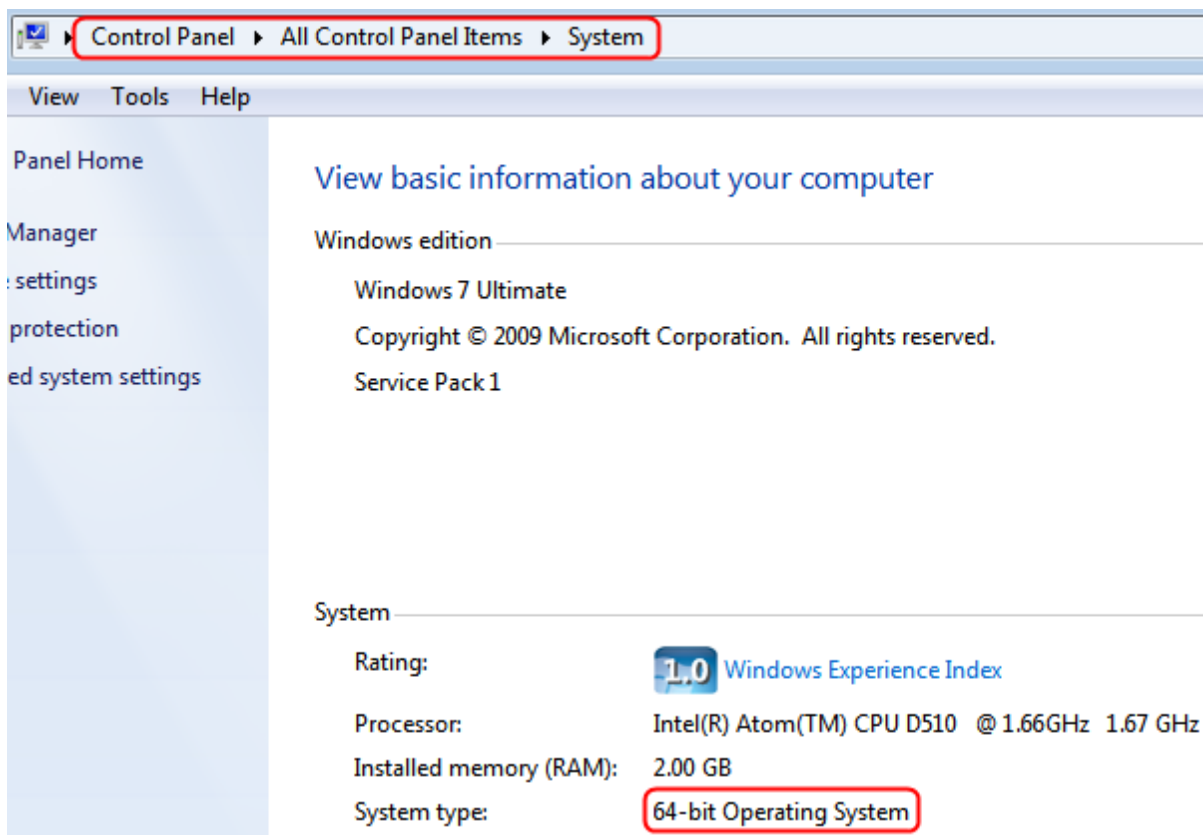
4.5.3 EcatDrv for Kernel Timer

Due to the non-real-time behavior of Windows, it is not possible to get accurate timing when using the standard timer API. The EcatDrv implements an Kernel Timer that is accurate enough to get DCM in sync. The EcatDrv does not guarantee response to the cycle's deadline.

Prerequisites of the EcatDrv

Microsoft Windows: 32 bit or 64 bit?

On Windows the System Properties dialog contains the information about the PC's architecture. The System type states if it is a 64-bit Operating System or 32-Bit.



Disable CPU Power Management Driver

To prevent influences from the Intel-PPM-driver, the driver can be disabled. The registry file “intelppmOFF.reg” to disable the Intel-PPM-driver can be merged into the registry and is located in the <InstallPath>\Files\Windows-folder of the EC-Simulator installation directory. Merging “intelppmON.reg” would enable it again.

Disable Local APIC usage of Windows

To make sure that there is no collision between Windows and the EcatDrv, disable the use of the Local APIC timer for Windows. This can be done by changing the boot configuration with the following command in the Windows console as an administrator:

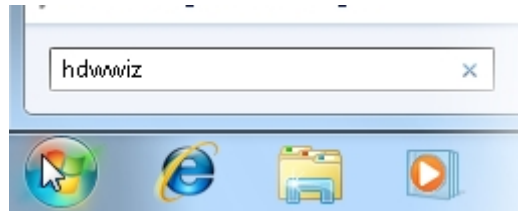
```
$ bcdedit.exe /set current useplatformtick yes
```

No Local APIC support since Windows 10

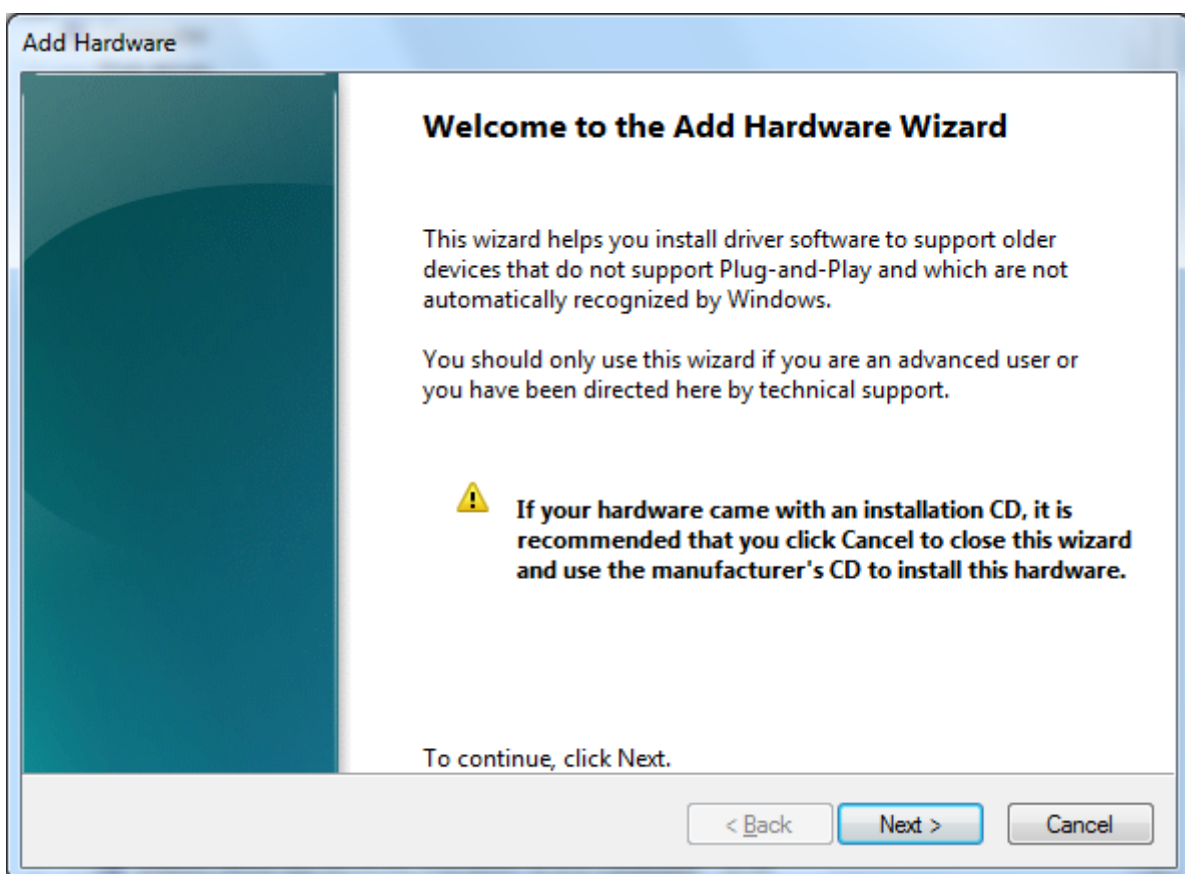
Since Windows 10, direct access to the local APIC timer is no longer allowed and can cause an blue screen of death (BSOD). In this case the EcatDrv can use the Windows Kernel Timer as clock source. The EC-Master stack automatically uses the EcatDrv if this driver is installed and activated on the Windows system. If not the standard timer API is used.

Installing and configuring the EcatDrv on Windows

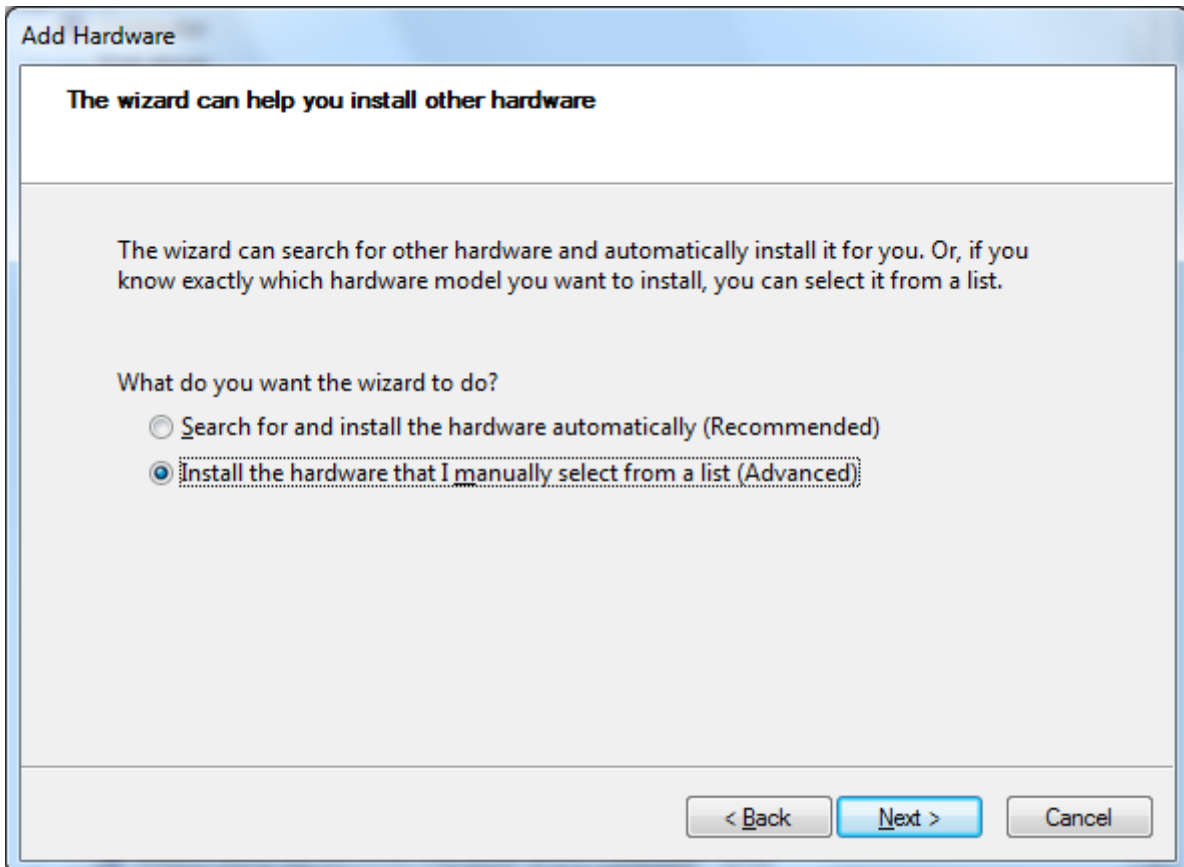
1. Start the “Add Hardware Wizard” and click “Next”



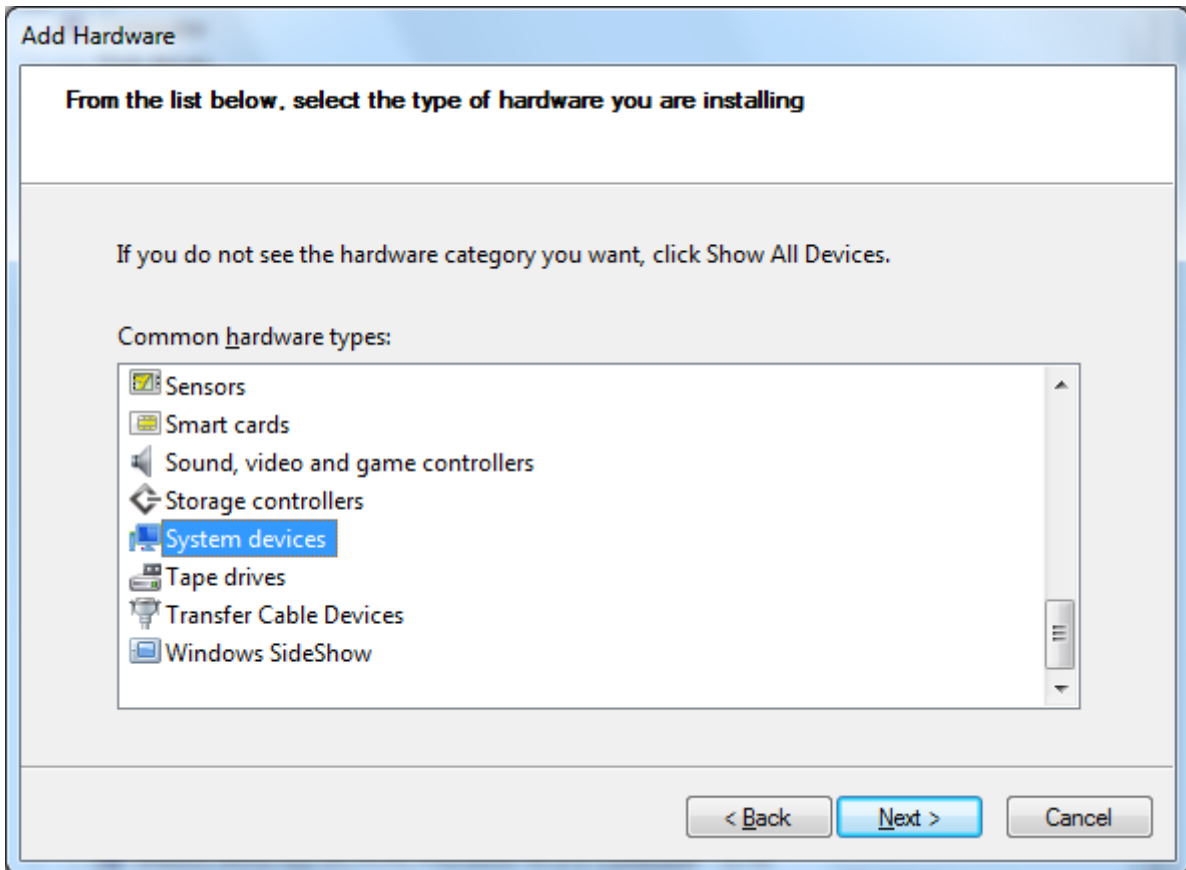
The Add Hardware Wizard shows up:



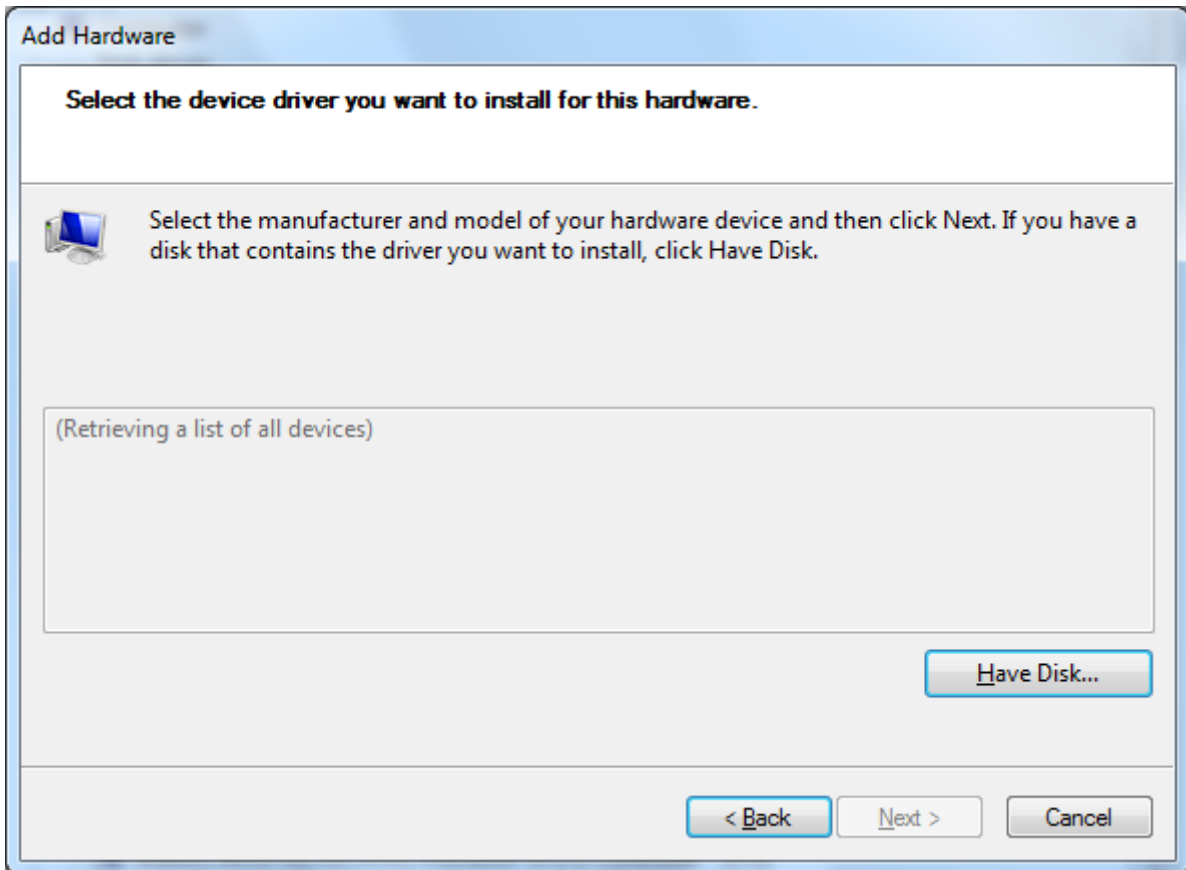
2. Select “Install the hardware...” and proceed



3. Select "System devices" and proceed



4. Click to “Have Disk...” and proceed

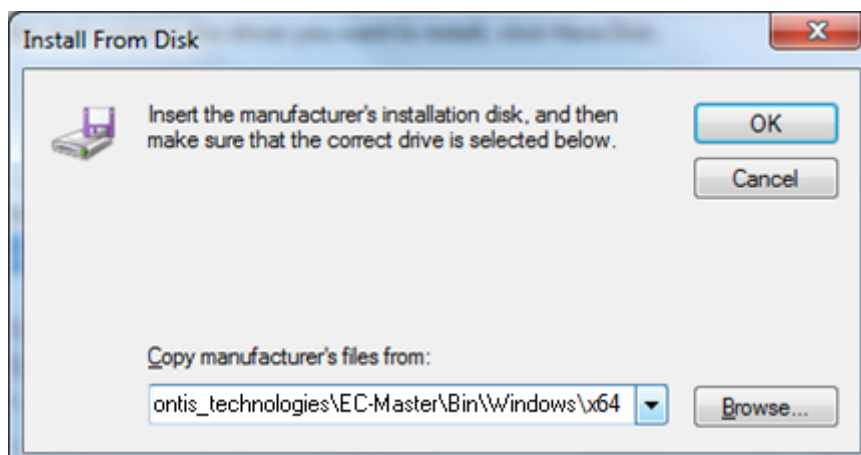


5. **Enter the directory to the correct driver version (32 bit or 64 bit)**

The folder is <InstallPath>\Bin\Windows\<Arch>, with x86 or x64 for <Arch> accordingly.

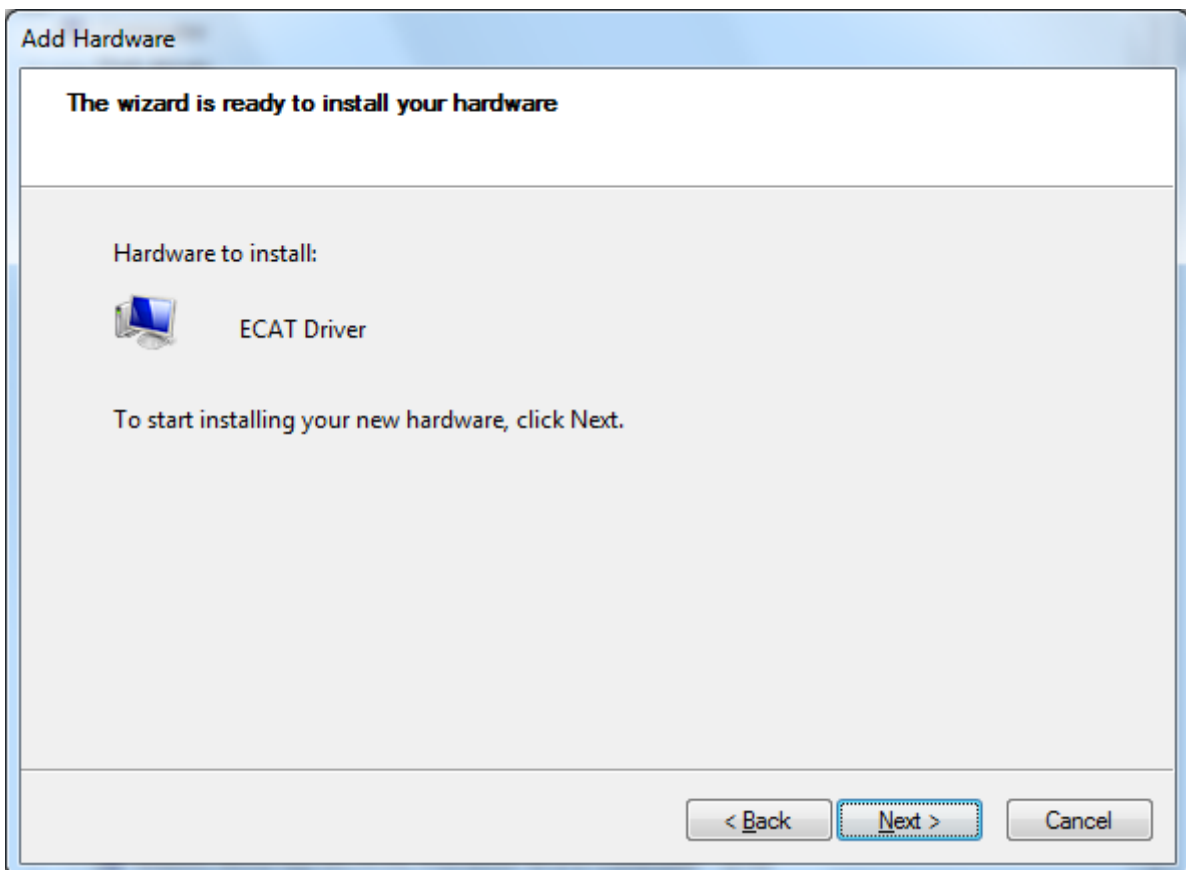
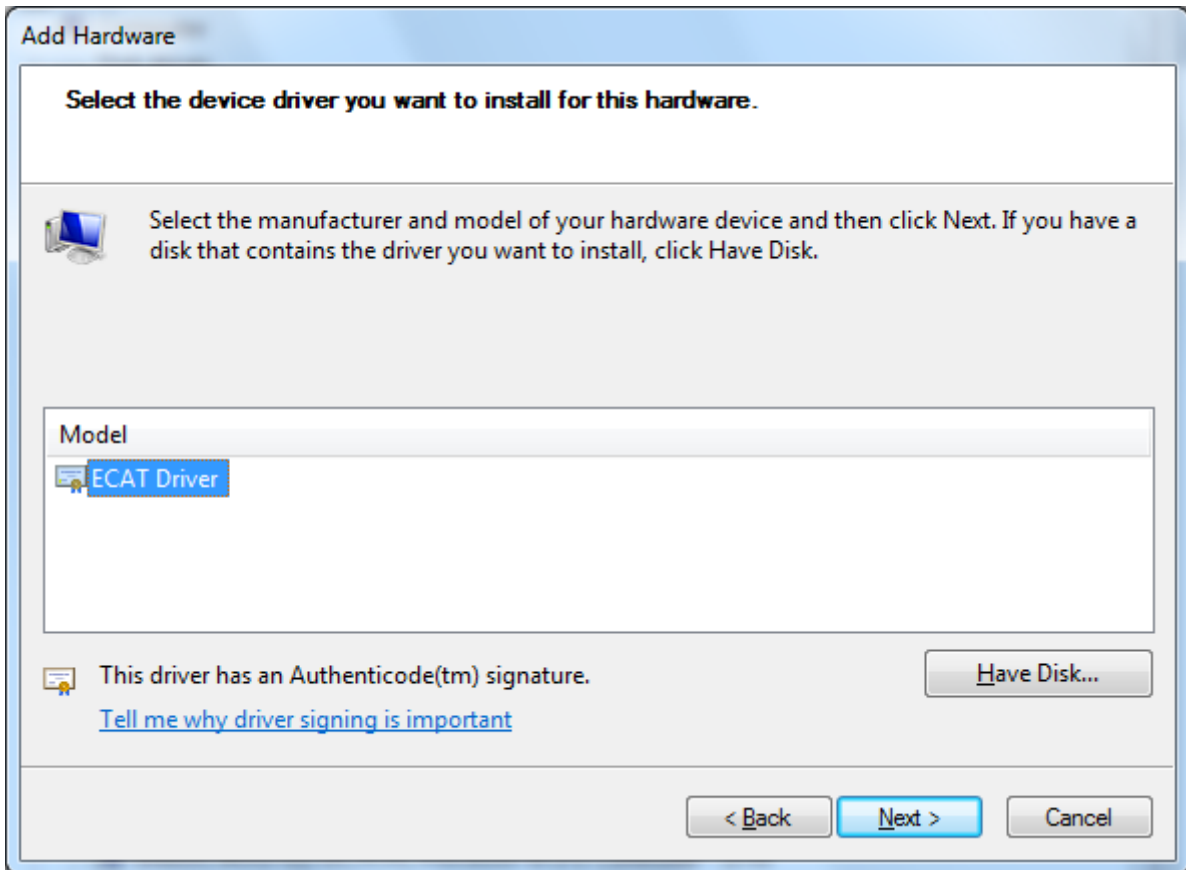
Attention: There are two different drivers available: 32 bit and 64 bit. The subfolder x86 contains the 32 bit driver files. The subfolder x64 contains the 64 bit driver files.

Enter the correct directory at the input box:

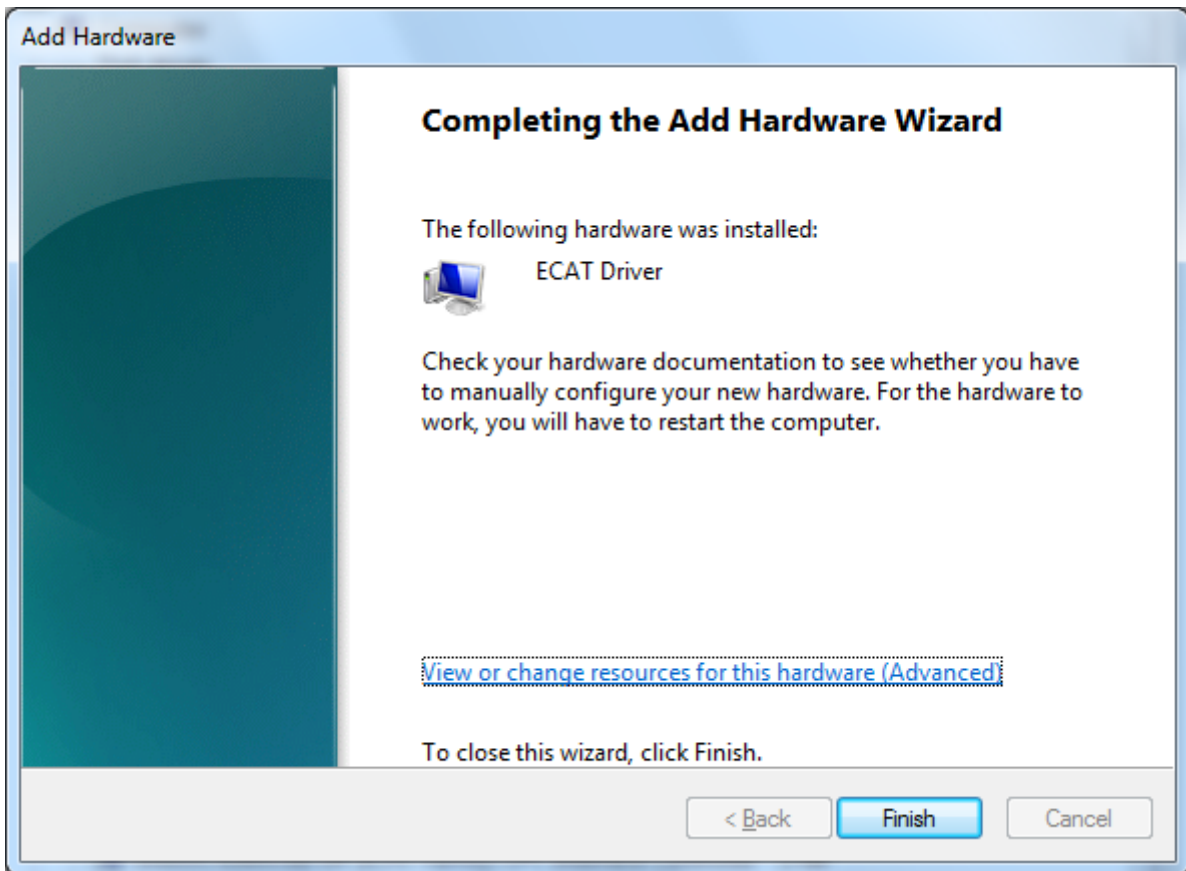


Press OK to proceed.

6. **Choose the EcatDrv and click “Next” and confirm the installation**



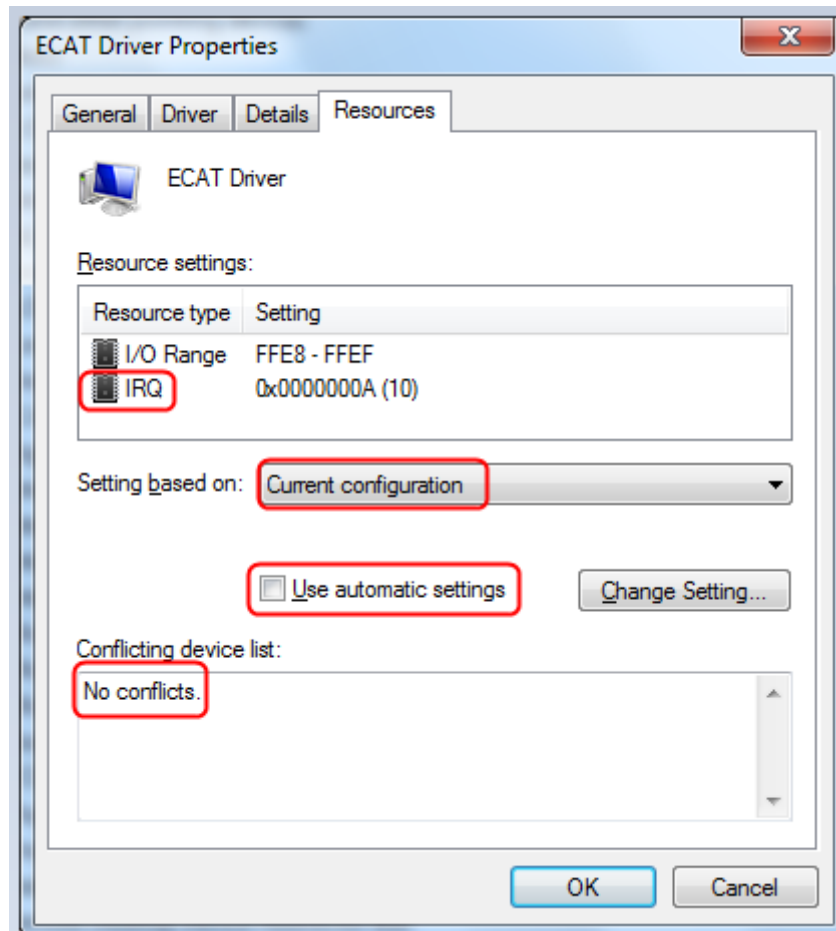
7. Continue on any warning about unsigned driver



8. Configure the driver for local APIC usage

Attention: This is not required for Windows 10 !

- Open the Device Manager and open the properties of the EcatDrv
- Uncheck “Use automatic settings”
- Change “Current Configuration” to “Basic configuration 0001”
- Double-click the “IRQ”-label and adjust the Interrupt Request Value until “No conflicts” is shown.



9. **Press “OK” to close and apply the settings**

Confirm the manual settings

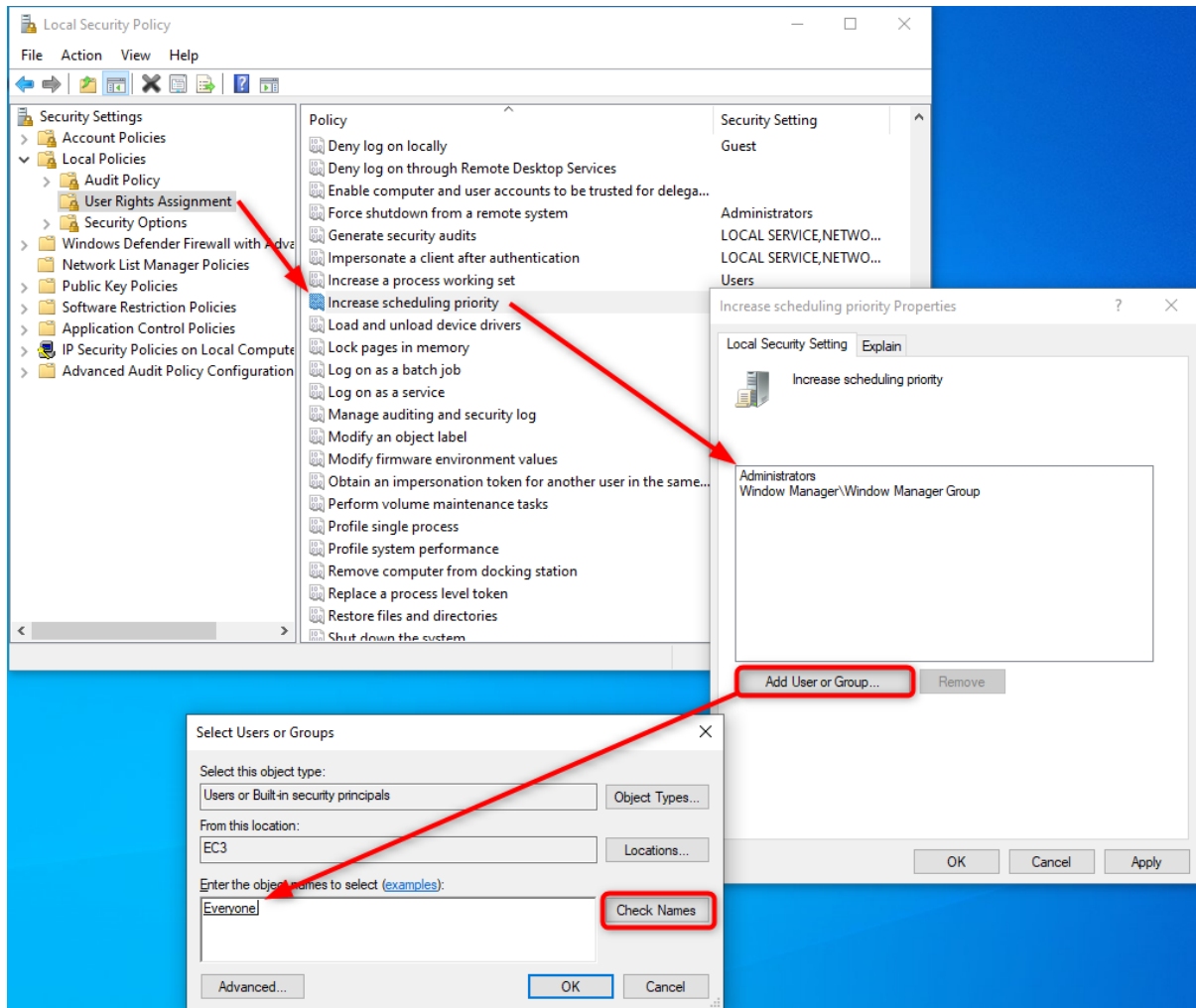
10. Restart the PC

Enable Realtime Priority Class

In order to comply with the timing as best as possible, it is necessary that the application / EcSimulatorHilDemo runs with realtime priority. By default, an application on Windows can not be run with real-time priority. To allow this the `Increase scheduling priority` user rights policy setting must be set. Also, the application must be run as an administrator.

Increase scheduling priority setting

Local Security Policy ▶ User Rights Assignment ▶ Increase scheduling priority ▶ Add User or Group ... ▶ Everyone

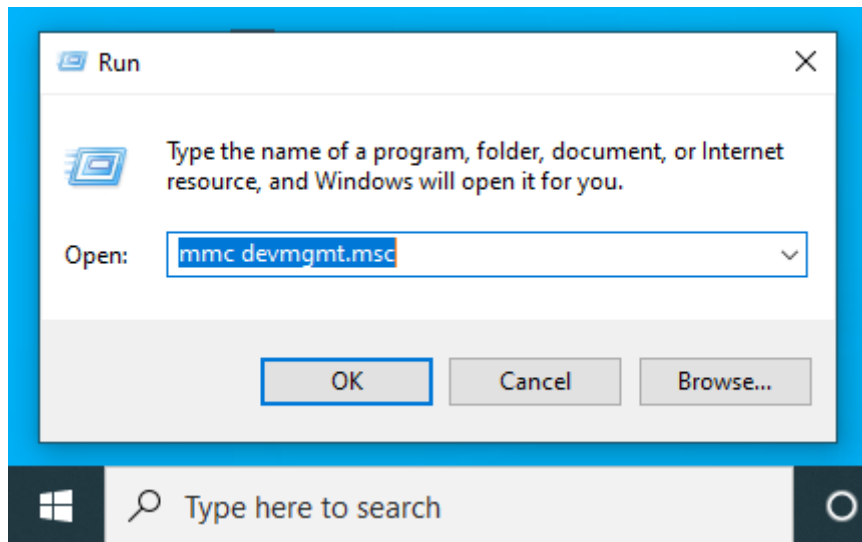


4.5.4 EcatDrv for Real-time Ethernet Driver

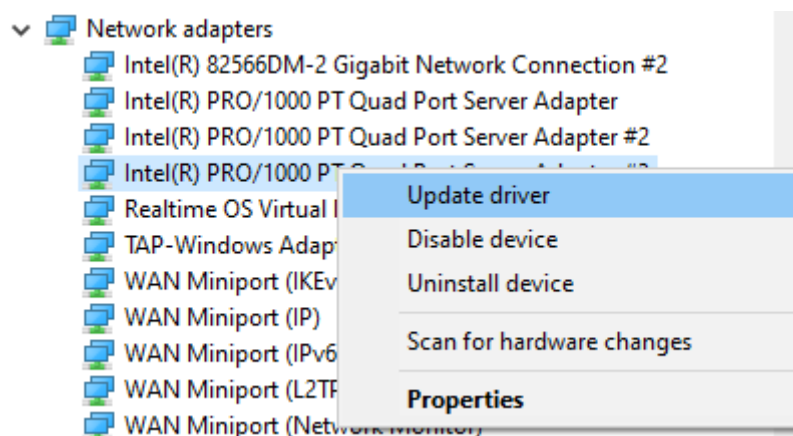
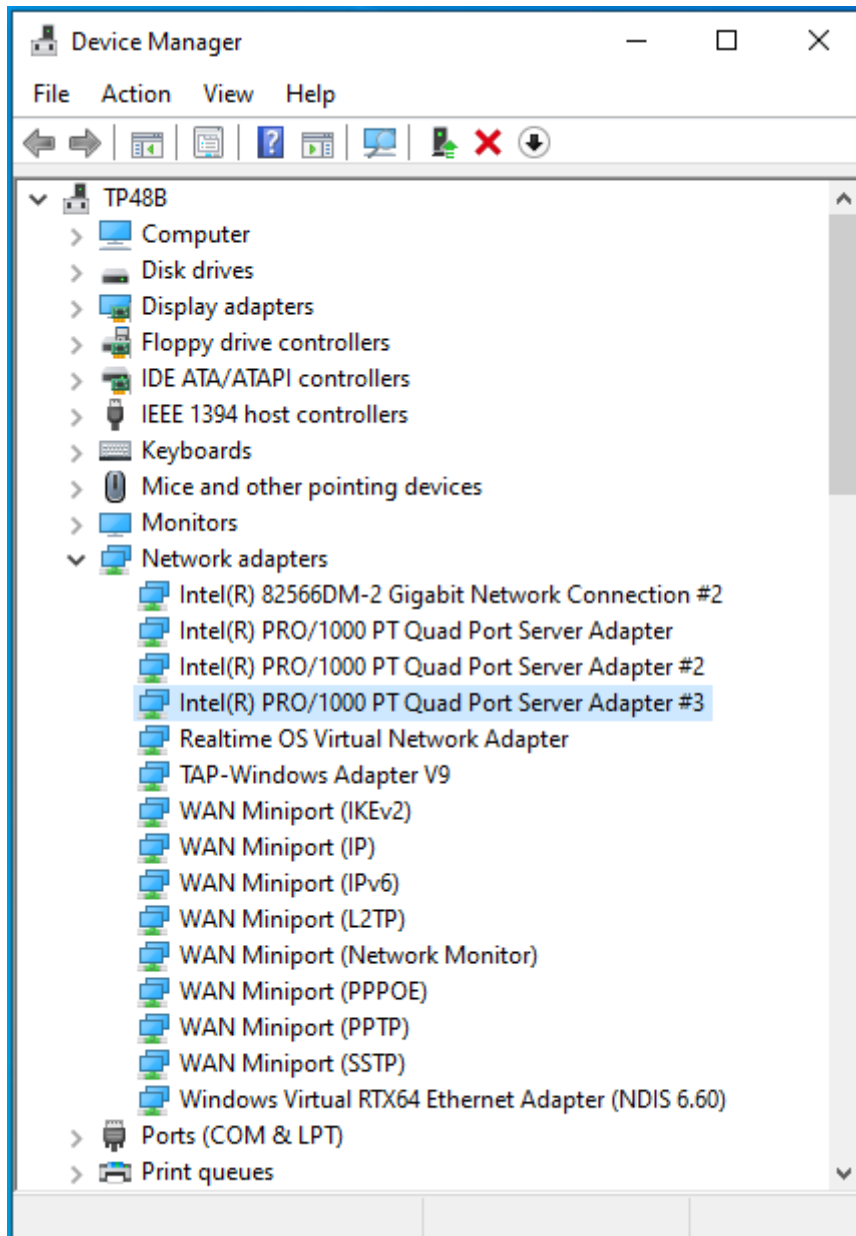
As alternative to the NDIS based or Pcap based Ethernet Driver, an optional Real-time Ethernet Driver on Windows can be installed. The Real-time Ethernet Driver replaces the original Windows driver and also requires an extra license.

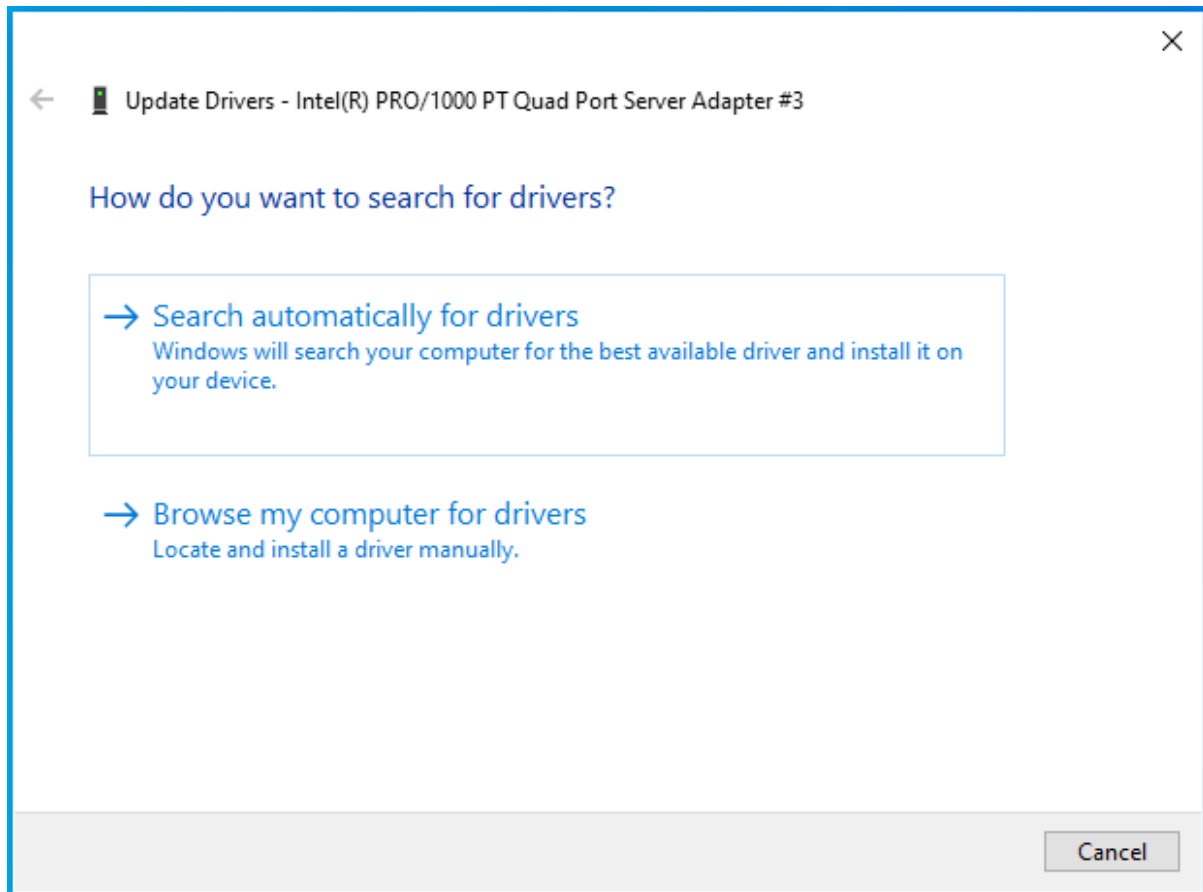
To use the Real-time Ethernet Driver under Windows, it is necessary to install the EcatDrv driver included in the Real-time Ethernet Driver delivery:

1. **Start the “Device Manager”**

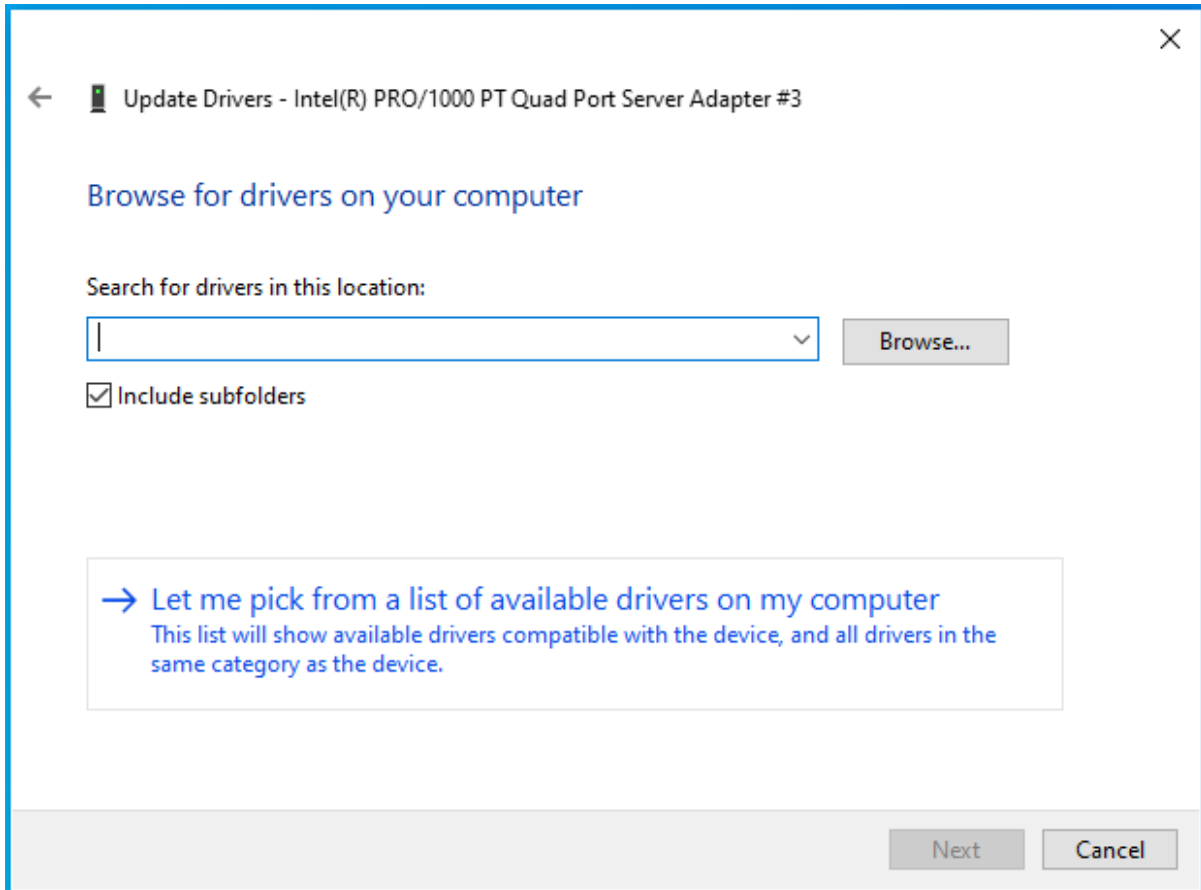


2. **Assign EcatDrv to the network adapter**

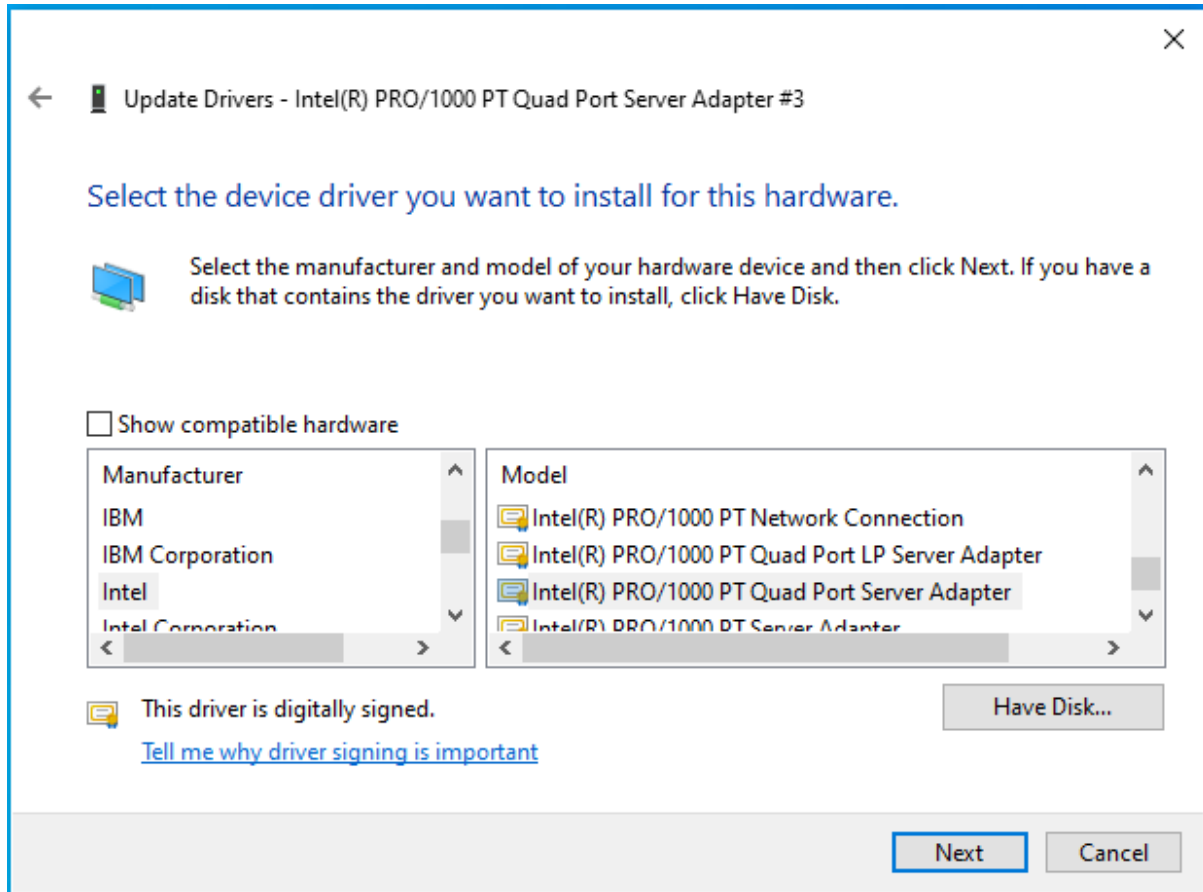




Click on “Browse my computer for driver”



Click on “Let me pick...”

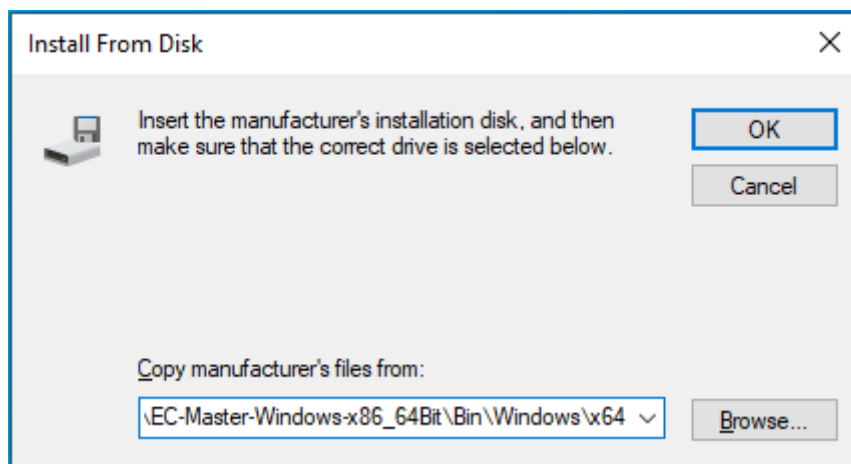


Click on “Have Disk...”

3. Enter the directory of EcatDrv

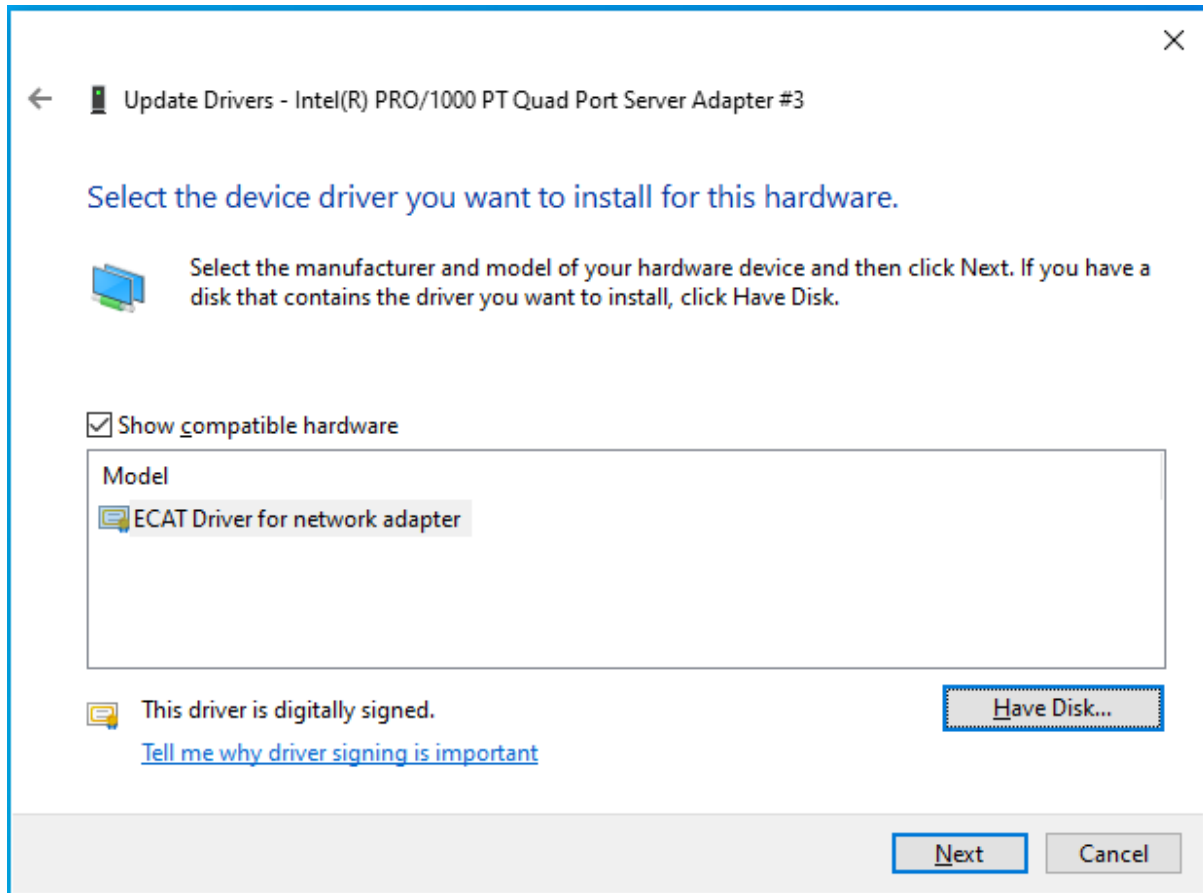
The folder is <InstallPath>\Bin\Windows\<Arch>, with x86 or x64 for <Arch> accordingly.

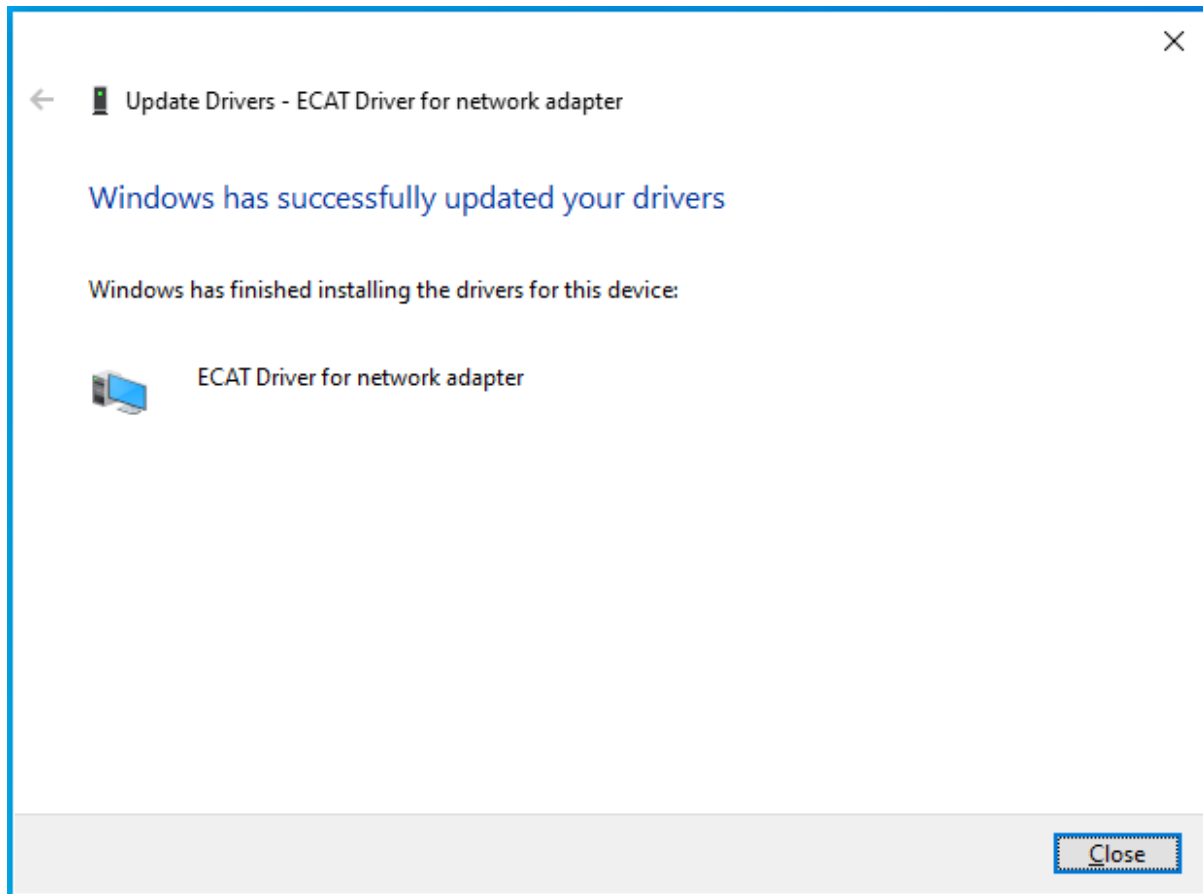
Attention: Two different versions are available: 32 bit (subfolder **x86**) and 64 bit (subfolder **x64**).



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

4. Choose the EcatDrv and click “Next” and confirm the installation





Optionally modify search location Real-time Ethernet Driver

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

4.6 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:

- **armv6-vfp-eabihf:**
 - Xenomai 2.6.3, tested on Linux Kernel 3.8.13-xenomai-2.6.4
- **x64:**
 - Xenomai 3.0.2, tested on Linux Kernel 3.18.20 (Cobalt)
- **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)

4.6.1 Setting up and running EcSimulatorHilDemo

1. Prepare system

Prepare the system to run EcSimulatorHilDemo on Linux as described in chapter [Linux](#)

2. Compile EcSimulatorHilDemo

As a starting point there is the Eclipse project for EcSimulatorHilDemo for Xenomai located at Workspace/Xenomai/EcSimulatorHilDemo. Ensure OPERATING_SYSTEM, ARCH, CFLAGS, LD_FLAGS, LD_LIBRARY_PATH are set accordingly (export ARCH=x86, ...) when compiling using Eclipse!

3. Run using GDB

Provide search path for Xenomai libraries and prevent GDB to stop execution on SIGXCPU:

```
$ export LD_LIBRARY_PATH=../../Bin/Xenomai/x86:/usr/xenomai/lib:..
$ gdb --args ./EcSimulatorHilDemo -intelgbe 1 0 -f exi.xml -v 3
$ [...]
$ (gdb) handle SIGXCPU nostop noprint nopass
$ (gdb) run
```

See also:

[Running EcSimulatorHilDemo](#)

4.6.2 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

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

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/>.

5 Real-time Ethernet Driver

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

Real-time means operating directly on the network device's register 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 master cycle time must be at least two times higher than the simulator cycle time. E.g. if the simulator runs with 1 ms, the master 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. Each Real-time Ethernet Driver specific structure start with a common `EC_T_LINK_PARMS` structure, followed by some Real-time Ethernet Driver specific members. The common link parameter structure is passed to `EC_T_INIT_SIMULATOR_PARMS::pLinkParms` with the call of `esInitSimulator()` like in the following example:

```

/* identify Link Layer in the common struture */
oLinkParmsSockRaw.linkParms.dwSignature = EC_LINK_PARMS_SIGNATURE SOCKRAW;
oLinkParmsSockRaw.linkParms.dwSize = sizeof(EC_T_LINK_PARMS SOCKRAW);
OsStrncpy(&oLinkParmsSockRaw.linkParms.szDriverIdent, EC_LINK_PARMS_IDENT SOCKRAW,
↳EC_DRIVER_IDENT_MAXLEN);

/* specific Link Layer parameters should be set here */

/* pass Link Layer parameters */
oInitMasterParms.dwSignature = ATECAT_SIGNATURE;
oInitMasterParms.dwSize = sizeof(EC_T_INIT_MASTER_PARMS);
oInitMasterParms.pLinkParms = &oLinkParmsSockRaw.linkParms;

/* more parameters should be set here */

```

(continues on next page)

(continued from previous page)

```
/* initialize master */
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_NAMESIZE]

[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 timing task parameters

EC_T_CHAR szLoadPath[EC_DRIVER_PATH_MAXLEN]

[in] path from which the libraries should be loaded

```
struct EC_T_LINKLAYER_TIMINGTASK
```

Public Members

EC_T_LINKLAYER_TIMING **eLinkLayerTiming**

[in] LinkLayer timing task mode

EC_T_DWORD **dwCycleTimeUsec**

[in] Cycle time between 2 pfnStartCycle calls in us. Will be set by the master stack for the linklayer.

EC_T_LINK_STARTCYCLE_CALLBACK **pfnStartCycle**

[in] Callback function called cyclically according dwCycleTimeUsec

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_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

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
```

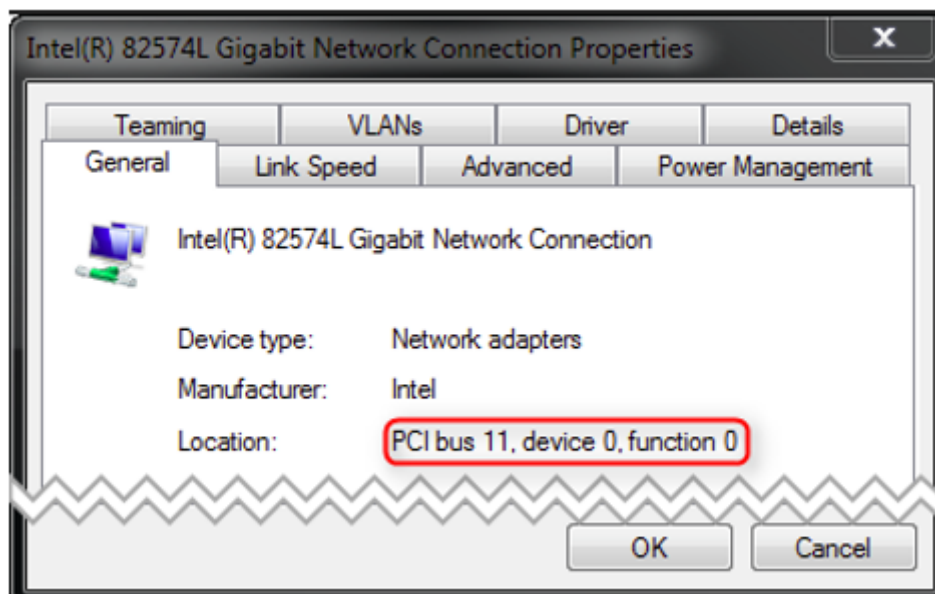
The format of `EC_T_LINK_PARMS::dwInstance` using PCI bus address is:

0x01bbdff

- *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 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 - emIII8254x

This Real-time Ethernet Driver is deprecated since V3.2.0.0. The parameters to the Intel Pro/1000 Real-time Ethernet Driver are setup-specific. The function `CreateLinkParmsFromCmdLineI8254x()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

```
struct EC_T_LINK_PARMS_I8254X
```


Public Members

EC_T_LINK_PARMS **linkParms**

Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_I8254X

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**

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

5.3 Intel Pro/1000 - emllIntelGbe

The parameters to the Real-time Ethernet Driver Intel Pro/1000 are setup-specific. The function “CreateLinkParms-FromCmdLineIntelGbe” 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_TRUE: copy buffer before processing (default), EC_FALSE: Use buffers from DMA

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.:

```
{
    EC_T_IOCTLPARMS oIoctlParms = {0};
    oIoctlParms.pbyInBuf      = (EC_T_BYTE*)EC_NULL + 0x20103;
    oIoctlParms.dwInBufSize   = sizeof(EC_T_DWORD);
    esIoControl(dwSimulatorInstanceId, EC_IOCTL_LINKLAYER_MAIN | EC_LINK_IOCTL_
↪FORCELINKMODE, &oIoctlParms);
    OsSleep(1000);
}
```

5.4 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 linkParmsCommon 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;
```

5.5 Broadcom BcmGenet - emllBcmGenet

The parameters to the BcmGenet Real-time Ethernet Driver 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_TRUE`: copy buffer before processing (default), `EC_FALSE`: Use buffers from DMA

5.6 Broadcom BcmNetXtreme - emllBcmNetXtreme

The parameters to the BcmNetXtreme Link Layer 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

5.7 Beckhoff CCAT - emllCCAT

The parameters to the Real-time Ethernet Driver CCAT are setup-specific. The function `CreateLinkParmsFromCmdLineCCAT()` 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

5.8 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

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

5.8.1 CPSW usage under Linux

Due to 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.

5.9 DW3504 - emIIDW3504

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

Use buffers from DMA (`EC_FALSE`) or from heap for receive. `AllocSend` is not supported, when `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

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

5.10 Freescale TSEC / eTSEC - emIIETSEC

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 protect 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_TRUE: copy buffer before processing, EC_FALSE: Use buffers from DMA (default)

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 be not called from the `JobTask`'s context. I.e. use:

```
dwRes = esIoControl(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 at all to avoid concurrent access if link speed and mode as 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 following PHY's are not affected by this parameter and still active. There is no forced link and no PHY access at all.

Parameters for fixed link:

```
pETSECParm->dwPhyAddr = ETSEC_FIXED_LINK;
pETSECParm->dwFixedLinkVal = ETSEC_LINKFLAG_1000baseT_Full | ETSEC_LINKFLAG_
↔LINKOK;
```

5.11 Freescale FslFec - emlIFslFec

The parameters to the Real-time Ethernet Driver `FslFec` are setup-specific. The function `CreateLinkParamsFromCmdLineFslFec()` 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 **linkParams**

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)

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)

5.12 Xilinx Zynq-7000/Ultrascale (GEM) - emIGEM

The parameters to the Real-time Ethernet Driver GEM are setup-specific. The function `CreateLinkParamsFromCmdLineGEM()` 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 GMIITORGMI Converter (`EC_TRUE`)

EC_T_DWORD **dwConvPhyAddr**

PHY address used to communicate with converter. In Linux doc it 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

5.13 Texas Instruments ICSS - emIIICSS

The parameters to the Real-time Ethernet Driver ICSS are setup-specific. The function `CreateLinkParamsFromCmdLineICSS()` 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 **linkParams**

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 link layer is used on single ICSS port. This flag is also required, when 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 `pd_k_am57xx_1_0_6`, instead of `pd_k_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

5.13.1 TTS Feature

Real-time Ethernet Driver PRU ICSS can optionally use Time-Triggered Send feature http://processors.wiki.ti.com/index.php/ICSS_EMAC_LLD_developers_guide#Time_Triggered_Send

To test it, you need to build a demo application with `INCLUDE_TTS` macro. Additionally, you need to set `bTts` flag and configure other `tts` parameters in `EC_T_LINK_PARAMS_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 to how long your own real project prepares cyclic and acyclic frames to be sent in the current cycle.

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=>

5.13.2 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

5.13.3 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).

5.13.4 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 maximum 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 future, this limitation can be removed.

5.14 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

5.15 Windows WinPcap - emIIPcap

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 main network adapter is used for both EtherCAT devices and the local area network there may be a main impact on the local area network operation. The network adapter card used by EtherCAT has to be set to a fixed private IP address, e.g. `192.168.x.y`.

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

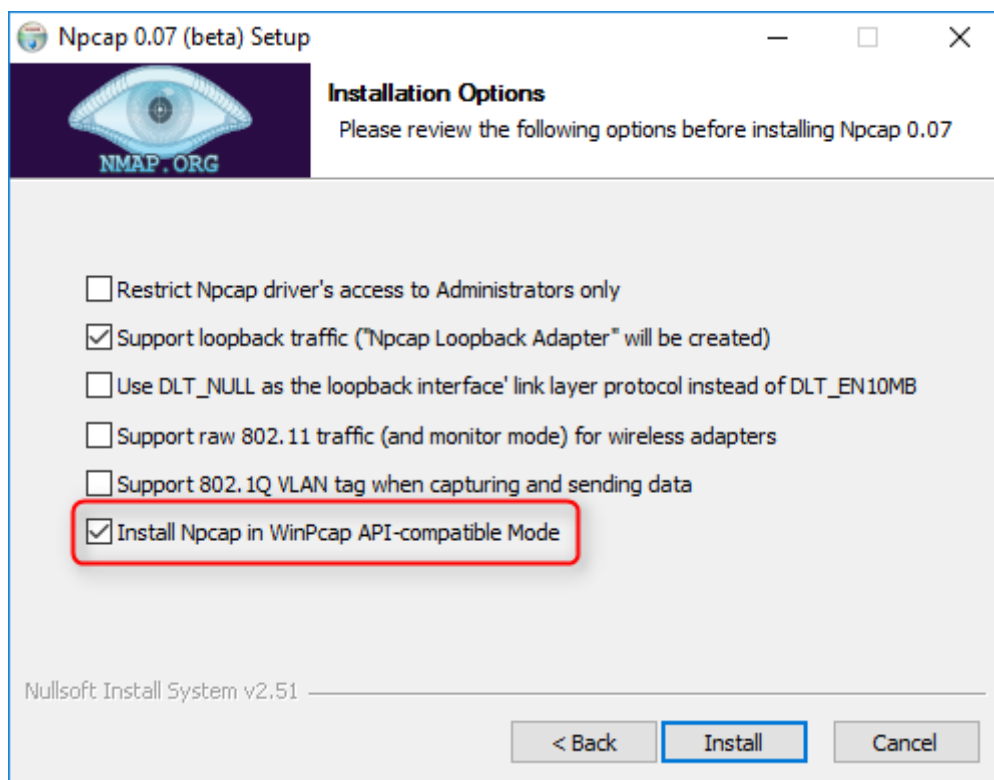
Filter input if EC_TRUE. This is needed on some system if the winpcap library notify the sent frames to the network adapter

5.15.1 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:



5.16 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 emllRemote

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

The parameters to the Remote Real-time Ethernet Driver are setup-specific. The function `CreateLinkParmsFromCmdLineRemote()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet 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 (`emrsockettype_tcp`)

`EC_T_BYTE` **abySrcIpAddress[4]**

Source adapter IP address (listen) (EC-Simulator)

`EC_T_WORD` **wSrcPort**

Source port number (listen) (EC-Simulator)

`EC_T_BYTE` **abyDstIpAddress[4]**

Destination adapter IP address (connect) (EC-Master)

`EC_T_WORD` **wDstPort**

Destination port number (connect) (EC-Master)

`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
```

5.18 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_TRUE: copy buffer before processing, EC_FALSE: Use buffers from DMA (default)

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

5.18.1 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 Renesas SHEth - emII SHEth

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

```
struct EC_T_LINK_PARMS_SHETH
```

Public Members

EC_T_LINK_PARMS **linkParms**

Common link parameters. Signature must be set to `EC_LINK_PARMS_SIGNATURE_SHETH`

EC_T_SHETH_TYPE **eType**

System on Chip type

EC_T_BYTE **abyStationAddress[6]**

MAC address

EC_T_DWORD **dwBaseAddr**

Physical address of register block

EC_T_BYTE **byPhyAddr**

PHY address

EC_T_BOOL **bNotUseDmaBuffers**

`EC_TRUE`: copy buffer before processing, `EC_FALSE`: Use buffers from DMA (default)

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

```
enum EC_T_SHETH_TYPE
```

Values:

enumerator **eSHEth_R8A777X**

Renesas R8A777X

enumerator **eSHEth_R8A779X**

Renesas R8A779X

enumerator **eSHEth_SH7724**

Renesas SH7724

enumerator **eSHEth_SH7757**

Renesas SH7757

enumerator **eSHEth_SH7757_GIGA**

Renesas SH7757_GIGA

enumerator **eSHEth_SH7734**
Renesas SH7734

enumerator **eSHEth_SH7763**
Renesas SH7763

enumerator **eSHEth_R8A7740**
Renesas R8A7740

enumerator **eSHEth_R7S72100**
Renesas R7S72100

enumerator **eSHEth_SH7619**
Renesas SH7619

enumerator **eSHEth_SH771X**
Renesas SH771X

enumerator **eSHEth_R8A77400**
Renesas R8A77400

enumerator **eSHEth_R8A77435**
Renesas R8A77435

enumerator **eSHEth_R8A77430**
Renesas R8A77430

enumerator **eSHEth_R8A77450**
Renesas R8A77450

5.19.1 SHEth link status update

On some targets like *Armadillo A800 eva* the link status can't be obtained directly by reading MAC PHY status register without access to the MII bus. 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 be not called from the JobTask's context.

```
dwRes = esIoControl(dwSimulatorInstanceId, EC_IOCTL_LINKLAYER | EC_LINKIOCTL_
->UPDATE_LINKSTATUS, EC_NULL);
```

`ecLinkGetStatus()` always returns the last known link status.

5.19.2 SHEth usage under Linux

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

5.20 VxWorks SNARF - emII SNARF

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 initialize the EC-Simulator stack.

Using the SNARF Real-time Ethernet Driver has some disadvantages. As the VxWorks network layering is involved in this architecture, the drivers are usually not optimized for realtime behavior 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 SNARF Real-time Ethernet Driver are setup-specific. The function `CreateLinkParamsFromCmdLineSNARF()` 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 linkParams`

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 oLinkParamsAdapter;

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

5.21 Linux SockRaw - emII SockRaw

The Real-time Ethernet Driver SockRaw is always part of the EC-Simulator for Linux package. It 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 devices. If the main network adapter is used for both EtherCAT devices and the local area network there may be a main impact on the local area network operation.

Note: The SockRaw cannot be used for real time applications and may need cycle time of 4 ms or higher.

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

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

5.22 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 Master can use it with e.g. `emllNdis`.

6 Application programming interface, reference

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.

6.1 Generic API return status values

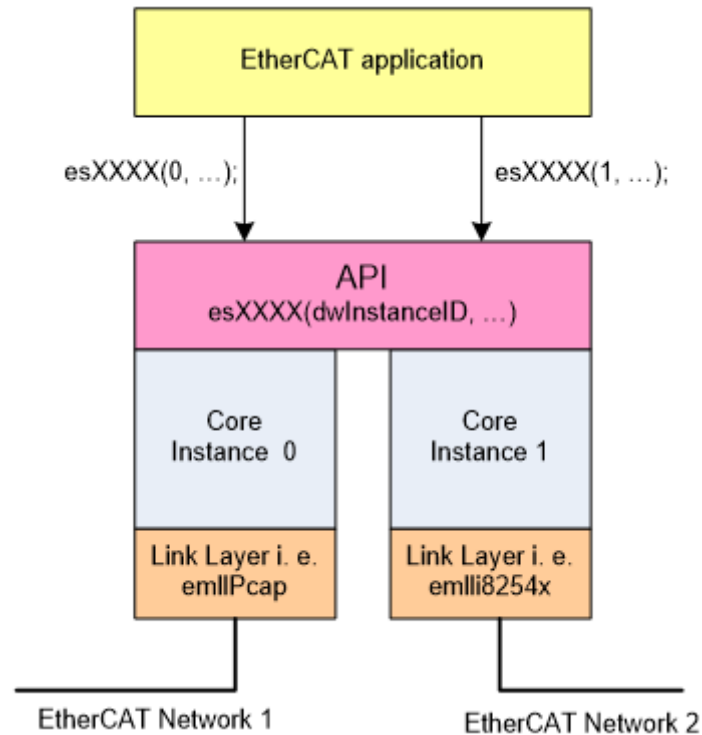
Most of the functions and also some notifications will return an error status value to indicate whether a function was successfully executed or not. Some of the return status values have a generic meaning unspecific to the called API function.

- `EC_E_NOERROR`: The function was successfully executed.
- `EC_E_NOTSUPPORTED`: Unsupported feature or functionality.
- `EC_E_BUSY`: The stack currently is 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_INVALIDPARAM`: Invalid or inconsistent parameters.
- `EC_E_TIMEOUT`: Timeout error.
- `EC_E_SLAVE_ERROR`: A slave error was detected.
- `EC_E_INVALID_SLAVE_STATE`: The slave is not in the requested state to execute the operation (e.g. when initiating a mailbox transfer the slave must be at least in PREOP state).
- `EC_E_SLAVE_NOT_ADDRESSABLE`: The slave does not respond to its station address (e.g. when requesting its `AL_STATUS` value). The slave may be removed from the bus or powered-off.
- `EC_E_LINK_DISCONNECTED`: Link cable not connected.
- `EC_E_MASTERCORE_INACCESSIBLE`: Master core inaccessible. This result code usually means a remote connected server / EC-Simulator stack does not respond anymore

6.2 Multiple EtherCAT Network Support

6.2.1 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, ...).

6.2.2 Licensing

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

6.3 General functions

6.3.1 esInitSimulator

```
EC_T_DWORD esInitSimulator (
    EC_T_DWORD dwInstanceId,
    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

struct **_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 dwBusCycleTimeUsec

[in] [usec] bus cycle time in microseconds

EC_T_BOOL bDisableProcessDataImage

[in] (legacy support, CiA402 simulation)

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)

struct **_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 msec 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

struct `_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)(struct _EC_T_LOG_CONTEXT *pContext, EC_T_DWORD dwLogMsgSeverity, const EC_T_CHAR *szFormat, ...)`

Parameters

- **pContext** – [in] Context pointer. This pointer is used as parameter when the callback function is called
- **dwLogMsgSeverity** – [in] Log message severity, `EC_LOG_LEVEL_...`
- **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.

Returns

`EC_E_NOERROR` or error code

Log messages are passed from the EC-Simulator to the callback given at `EC_T_LOG_PARMS::pfLogMsg`. `EcLogging.cpp` demonstrates how messages can be handled by the application. 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)`

Parameters

`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

6.3.2 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

6.3.3 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()

6.3.4 esSetSimulatorParms

EC_T_DWORD esSetSimulatorParms (
 EC_T_DWORD dwInstanceId,
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()

6.3.5 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()`.

6.3.6 esSetLicenseKey (HiL)

```
EC_T_DWORD esSetLicenseKey (
    EC_T_DWORD dwInstanceID,
    const EC_T_CHAR *pszLicenseKey
)
```

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)
- **pszLicenseKey** – [in] License key as zero terminated string with 26 characters.

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master isn't initialized
- *EC_E_INVALIDPARAM* if `dwInstanceID` is out of range
- *EC_E_INVALIDSIZE* the format of the license key is wrong. The correct length is 26 characters
- *EC_E_LICENSE_MISSING* the license key doesn't match to 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");
```

6.3.7 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);
```

6.3.8 esConfigureNetwork

EC_T_DWORD **esConfigureNetwork** (
 EC_T_DWORD dwInstanceID,
 EC_T_CNF_TYPE eCnfType,
 EC_T_PBYTE pbyCnfData,
 EC_T_DWORD dwCnfDataLen
)

Configure the Master.

This function must be called after the master has been initialized. 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 master isn't initialized or eCnfType is eCnfType_GenPreopENI or eCnfType_GenOpENI and link is disconnected
- *EC_E_INVALIDPARAM* 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 some format error 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 offset in the ENI are invalid
- *EC_E_XML_INVALID_OUT_OFF* if some output offset 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

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

6.3.9 esSetSlaveSscApplication

```
EC_T_DWORD esSetSlaveSscApplication (
    EC_T_DWORD dwInstanceId,
    EC_T_WORD wCfgFixedAddress,
    struct _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

See also:

- *ADS over EtherCAT (AoE)*
- *Ethernet over EtherCAT (EoE)*
- *File access over EtherCAT (FoE)*
- *esInitSimulator()*
- *esConfigureNetwork()*

6.3.10 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 master isn't initialized

- *EC_E_INVALIDPARAM* 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)
```

Parameters

- **dwCode** – [in] Notification code, see EC_NOTIFY_...
- **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)

6.3.11 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

6.3.12 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 master 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 master isn't initialized
- *EC_E_INVALIDPARG* 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 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

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 slave state machines, copy process data from simulated slave firmware as well as the mailbox handling. The slave state machines handle the EtherCAT state change transitions.

Return: EC_E_NOERROR if successful, error code in case of failures.

6.3.13 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

6.3.14 esIoControl

With `ref`esIoControl`` a generic control interface exists between the application and the EC-Simulator and its Real-time Ethernet Driver.

struct **EC_T_IOCTLPARMS**

Public Members

EC_T_BYTE *pbyInBuf

[in] Pointer to control input parameter.

EC_T_DWORD dwInBufSize

[in] Size of the input buffer provided at pbyInBuf in bytes

EC_T_BYTE *pbyOutBuf

[out] Pointer to control output buffer where the results will be copied into

EC_T_DWORD dwOutBufSize

[in] Size of the output buffer provided at pbyOutBuf in bytes

EC_T_DWORD *pdwNumOutData

[out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer

6.3.15 esIoControl - EC_IOCTL_GET_PDMEMORYSIZE

Queries the necessary size the process data image has got. This IOCTL is to be called after esConfigureNetwork.

esIoControl - EC_IOCTL_GET_PDMEMORYSIZE

Parameter

- 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 (EC_T_MEMREQ_DESC)
- dwOutBufSize: [in] Size of the output buffer provided at pbyOutBuf in bytes.
- pdwNumOutData: [out] Should be set to EC_NULL

struct EC_T_MEMREQ_DESC

6.3.16 esIoControl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB

This function call registers a callback function which is called after the cyclic frame is received. Typically this is used when the Real-time Ethernet Driver operates interrupt mode to get an event when the new input data (cyclic frame) is available. The callback function has to be registered after calling *esConfigureNetwork()* before starting the job task.

esIoControl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB

Parameter

- 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

```
struct EC_T_CYCFRAME_RX_CBDESC
```

Public Members

```
EC_T_VOID *pCallbackContext
```

[in] Context pointer. This pointer is used as parameter every time when 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)
```

Parameters

- **dwTaskId** – [in] Task ID of the received cyclic frame
- **pvContext** – [in] Arbitrarily application-defined parameter passed to callback

6.3.17 esIoControl - EC_IOCTL_ISLINK_CONNECTED

Determine whether link between the EC-Simulator stack and the first slave is connected.

```
esIoControl - EC_IOCTL_ISLINK_CONNECTED
```

Parameter

- **pbyInBuf**: [in] Should be set to EC_NULL
- **dwInBufSize**: [in] Should be set to 0
- **pbyOutBuf**: [out] Pointer to EC_T_DWORD. If value is EC_TRUE link is connected, if EC_FALSE it is not.
- **dwOutBufSize**: [in] Size of the output buffer provided at pbyOutBuf in bytes.
- **pdwNumOutData**: [out] Should be set to EC_NULL

Important: With cable redundancy support enabled, EC_FALSE is only set if all links are down.

6.3.18 esIoControl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO

Get cyclic configuration details from ENI configuration file.

```
esIoControl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO
```

Parameter

- **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.

```
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

`EC_T_DWORD dwPriority`
[out] Priority of selected cyclic entry

`EC_T_DWORD dwCycleTime`
[out] Cycle time of selected cyclic entry

6.3.19 esIoControl - EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES

Enable or disable additional variables for specific data types. Default: Enabled.

esIoControl - EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES

Parameter

- `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`

6.3.20 esIoControl - EC_IOCTL_SIMULATOR_SET_MBX_PROCESS_CTL

Enable or disable mailbox processing at slave. Default: Enabled.

esIoControl - EC_IOCTL_SIMULATOR_SET_MBX_PROCESS_CTL

Parameter

- `pbyInBuf`: [in] Pointer to value of `:cpp:struct: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`

```
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)

6.3.21 esIoControl - EC_IOCTL_SIMULATOR_GET_MBX_PROCESS_CTL

Enable or disable mailbox processing at slave. Default: Enabled.

esIoControl - EC_IOCTL_SIMULATOR_GET_MBX_PROCESS_CTL

Parameter

- `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 `:cpp:struct: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.

See also:

esIoControl - EC_IOCTL_SIMULATOR_SET_MBX_PROCESS_CTL

6.3.22 esIoControl - EC_LINKIOCTL_XXXX

The generic control interface `esIoControl()` 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`.

6.3.23 esIoControl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS

Get MAC address of network adapter

esIoControl - 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

6.3.24 esIoControl - EC_LINKIOCTL_GET_SPEED

Get current network adapter's speed in MBits.

esIoControl - 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

6.3.25 esGetVersion

```
EC_T_DWORD esGetVersion (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD *pdwVersion,
    EC_T_DWORD *pdwVersionType
)
```

Gets the version number as a 32-bit value.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pdwVersion** – [out] Pointer to `EC_T_DWORD` to carry out version number
- **pdwVersionType** – [out] Pointer to `EC_T_DWORD` to carry out version type

Returns

- `EC_E_NOERROR` if successful
- `EC_E_INVALIDSTATE` if master isn't initialized
- `EC_E_INVALIDPARAM` if `dwInstanceID` is out of range or the output pointer is `EC_NULL`

6.3.26 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

6.3.27 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

6.3.28 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 to 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 master state */
    eMasterState = (EC_T_STATE) (dwLogFlags & EC_LOG_FRAME_FLAG_MASTERSTATE_MASK);

```

(continues on next page)

(continued from previous page)

```

/* 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 ... */
}

```

6.3.29 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

6.4 Process Data Access Functions

6.4.1 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 *esIoControl - EC_IOCTL_GET_PDMEMORYSIZE* and *esRegisterClient ()* at *EC_T_REGISTERRESULTS::dwPDInSize*.

See also:

- *esConfigureNetwork ()*
- *esIoControl - EC_IOCTL_GET_PDMEMORYSIZE*
- *esIoControl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB*
- *esExecJob ()* (eUsrJob_ProcessAllRxFrames) in case of polling mode

6.4.2 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 *esIoControl - EC_IOCTL_GET_PDMEMORYSIZE* and *esRegisterClient ()* at *EC_T_REGISTERRESULTS::dwPDOOutSize*.

See also:

- *esConfigureNetwork ()*
- *esIoControl - EC_IOCTL_GET_PDMEMORYSIZE*
- *esIoControl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB*
- *esExecJob ()* (eUsrJob_ProcessAllRxFrames) in case of polling mode

6.4.3 EC_COPYBITS

EC_COPYBITS (pbyDst, nDstBitOffs, pbySrc, nSrcBitOffs, nBitSize)

Copies a block of bits from a source buffer to a destination buffer.

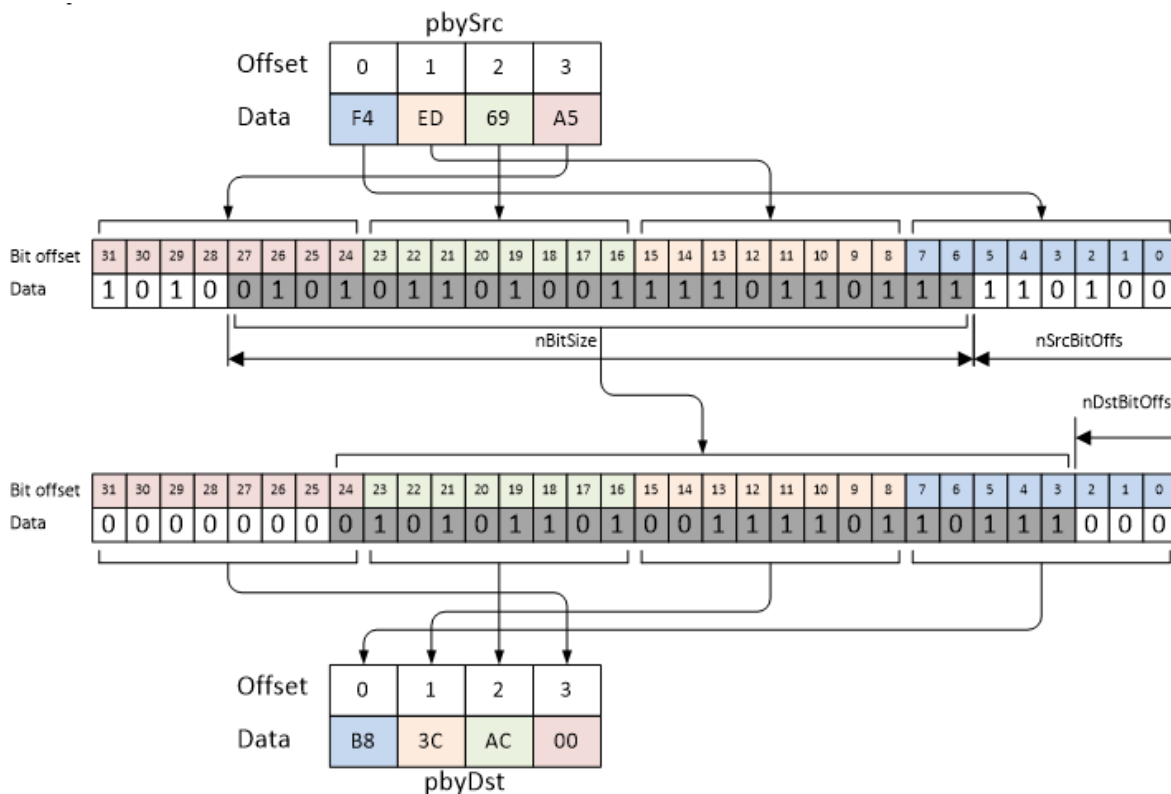
Note: The memory buffers must be allocated before. The buffers must be big enough to hold the block starting at the given offsets! The buffers are not checked for overrun.

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 */
```

6.4.4 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 */
```

6.4.5 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 */
```

6.4.6 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 */
```

6.4.7 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 ... */
```

6.4.8 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 ... */
```


6.4.9 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 ... */
```

6.4.10 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 be only 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*

6.4.11 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 be only 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*

6.4.12 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
)
```

Retrieve Process data synchronized.

If process data are required outside the cyclic master job task (which is calling `ecatExecJob`), 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 master stack and then check every millisecond whether new data are provided. The master stack will provide new data after calling `ecatExecJob(eUsrJob_MasterTimer)` 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

6.4.13 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 master job task (which is calling `ecatExecJob`), 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 master stack and then check every millisecond whether new data is written. The master stack will copy the data after calling `ecatExecJob(eUsrJob_MasterTimer)` 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

6.4.14 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

6.4.15 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

6.5 Notifications

When a client registers with the EC-Simulator stack the client has to determine a generic notification callback function. The stack calls this function every time an event (for example an error event or operational state change event) occurs about which the client has to be informed.

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
```

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 input buffer in byte

EC_T_BYTE *pbyOutBuf
[out] Notification output (result)

EC_T_DWORD dwOutBufSize
[in] Size of output buffer in byte

EC_T_DWORD *pdwNumOutData
[out] Amount of bytes written to the output buffer

6.6 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_INVALIDPARAM 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_INVALIDPARAM 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] 0: switch off, 1: 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 physical layer error (“single shot”) destroying frames at slave port.

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
)
```

Generate physical layer errors destroying frames at slave port randomly or at fixed intervals.

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
- **dwLikelihoodPpm** – [in] Random simulation: error likelihood (ppm)
- **dwFixedGoodFramesCnt** – [in] after error was simulated: fixed amount of frames without errors (at least)
- **dwFixedErroneousFramesCnt** – [in] on error simulation: fixed amount of frames with errors (at least)

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 in ms

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 in ms (at least)
- **dwFixedLinkUpTimeMs** – [in] after link down was simulated: fixed link up duration in 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_PFLFRAME_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 Master.

```
/* 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;
}
```

6.9 Slave control and status functions

6.9.1 esGetNumConfiguredSlaves

```
EC_T_DWORD esGetNumConfiguredSlaves (EC_T_DWORD dwInstanceId)
```

Returns 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 esGetNumConnectedSlavesEC_T_DWORD **esGetNumConnectedSlaves** (EC_T_DWORD dwInstanceID)

Get amount of currently connected slaves.

Parameters**dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)**Returns**

Number of connected slaves

6.9.3 esGetSlaveIdEC_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 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 esGetSlaveIdAtPositionEC_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 master isn't initialized
- *EC_E_INVALIDPARAM* if dwInstanceID is out of range or the output pointers are EC_NULL
- *EC_E_SLAVE_NOT_PRESENT* if slave 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 wNewReqDevState,
    EC_T_DWORD dwTimeout
)

```

Set a specified slave into the requested 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.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wNewReqDevState** – [in] Requested state
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master 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 master 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 slave is currently connected to the bus, EC_FALSE if not.

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master 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 emGetCfgSlaveInfo 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: wwwwww xxxx yyyy zzzz (each nibble : port 3210)

wwwwww : 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 master isn't initialized
- *EC_E_INVALIDPARG* 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 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 master isn't initialized
- *EC_E_INVALIDPARAM* 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

6.9.11 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

6.9.12 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 an 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 EcCommon.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 bit 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

6.9.13 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 master isn't initialized
- *EC_E_INVALIDPARAM* 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 EcCommon.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 bit 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 (emGetDiagnosisImagePtr)

EC_T_WORD wMasterSyncUnit

[out] Master Sync Unit (ENI: RxPdo[1..4]@Su, TxPdo[1..4]@Su)

EC_T_CYC_COPY_INFO CopyInfo

[out] Copy Info if applied to the variable

MAX_PROCESS_VAR_NAME_LEN_EX

Maximum length of a extended process variable name: 127 characters

6.9.14 esGetSlaveOutputVarInfo

EC_T_DWORD esGetSlaveOutputVarInfo (

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

6.9.15 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

6.9.16 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***6.9.17 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***6.9.18 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*

6.9.19 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

6.9.20 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

6.9.21 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

6.9.22 EC_NOTIFY_SLAVE_PRESENCE

This notification is given, if slave appears or disappears from the network.

This notification is enabled by default.

emNotify - EC_NOTIFY_SLAVE_PRESENCE

Parameter

- **pbyInBuf**: [in] Pointer to *EC_T_SLAVE_PRESENCE_NOTIFY_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 slave from the network, powering it off or a bad connection can produce this notification.

struct **EC_T_SLAVE_PRESENCE_NOTIFY_DESC**

Public Members

EC_T_WORD **wStationAddress**
Slave station address

EC_T_BYTE **bPresent**
EC_TRUE: present , *EC_FALSE*: absent

6.9.23 EC_NOTIFY_SLAVE_STATECHANGED

This notification is given, when a slave changed its EtherCAT state. This notification is disabled by default.

emNotify - EC_NOTIFY_SLAVES_STATECHANGED

Parameter

- **pbyInBuf**: [in] Pointer to *EC_T_SLAVE_STATECHANGED_NOTIFY_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_NOTIFY_DESC
```

Public Members

EC_T_SLAVE_PROP **SlaveProp**
Slave properties

EC_T_STATE **newState**
New slave state

6.9.24 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()*

6.9.25 esWriteSlaveRegister

Warning: Changing contents of ESC registers may lead to unpredictable behavior of the slaves and/or the master.

```
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 transferred data
- **wLen** – [in] Number of bytes to send
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master isn't initialized
- *EC_E_INVALIDPARAM* 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* 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

6.9.26 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.
- **pbpData** – [out] Buffer receiving transferred data
- **wLen** – [in] Number of bytes to receive
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master isn't initialized
- *EC_E_INVALIDPARAM* 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* 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

6.9.27 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 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

6.9.28 esWriteSlaveEEPROM

```
EC_T_DWORD esWriteSlaveEEPROM (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wEEPROMStartOffset,
    EC_T_WORD *pwWriteData,
    EC_T_DWORD dwWriteLen,
    EC_T_DWORD dwTimeout
)
```

Write EEPROM data to slave.

The EEPROM's CRC is updated automatically. `emResetSlaveController()` is needed to reload the alias address in register 0x12. 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 write data
- **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.

6.9.29 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] The slave's ID to bind bus slave and config slave information

EC_T_CHAR **abyDeviceName**[ECAT_DEVICE_NAMESIZE]

[out] The slave's configured name (80 Byte) (from ENI file)

EC_T_DWORD **dwHCGroupIdx**

[out] Index of the hot connect group, 0 for mandatory

EC_T_BOOL **bIsPresent**

[out] Slave is currently present on bus

EC_T_BOOL **bIsHCGroupPresent**

[out] Slave's hot connect group is currently 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] The slave's station address (from ENI file)

EC_T_WORD **wAutoIncAddress**

[out] The slave's auto increment address (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 (emSetSlaveDisabled / emSetSlavesDisabled).
- EC_T_BOOL bDisconnected**
[out] Slave disconnected by API (emSetSlaveDisconnected / emSetSlavesDisconnected).
- EC_T_BOOL bExtended**
[out] Slave generated by emConfigExtend
- 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 in 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 in ns (from ENI file)

Supported mailbox protocols flags

group **EC_MBX_PROTOCOLS**

Defines

EC_MBX_PROTOCOL_AOE

EC_MBX_PROTOCOL_EOE

EC_MBX_PROTOCOL_COE

EC_MBX_PROTOCOL_FOE

EC_MBX_PROTOCOL_SOE

EC_MBX_PROTOCOL_VOE

6.9.30 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 in topology segment connected to master. 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

6.9.31 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 master isn't initialized
- *EC_E_INVALIDPARG* 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)

www : 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]. This value is only valid if a DC configuration is used

EC_T_DWORD dwPropagDelay

[out] Propagation delay [ns]. ESC register 0x0928, This value is only valid if a DC configuration is used

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 slave 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 slave 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;
}

```

6.9.32 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 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 master isn't initialized
- *EC_E_INVALIDPARAM* 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* 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

6.9.33 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

6.9.34 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 Master 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 master isn't initialized
- *EC_E_INVALIDPARAM* 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 a 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*
- *EC_E_AOE_VENDOR_SPECIFIC*: will be returned in case the AoE device has responded with an 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 a 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*
- *EC_E_AOE_VENDOR_SPECIFIC*: will be returned in case the AoE device has responded with an 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 a 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 read from 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*
- *EC_E_AOE_VENDOR_SPECIFIC*: will be returned in case the AoE device has responded with an 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 Master Example

The following example demonstrates how to handle AoE object transfers within the simulator application:

AoE Simulator and Master 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,
```

(continues on next page)

(continued from previous page)

```

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;
    }

    /* 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:
    EC_LogMsg(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)

```

(continues on next page)

(continued from previous page)

```

{
    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) */)
{
    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 */)
    {

```

(continues on next page)

(continued from previous page)

```

        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);
    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 objects transfers between Master 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);

```

(continues on next page)

(continued from previous page)

```

    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));
        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,

```

(continues on next page)

(continued from previous page)

```

        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%x)\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,
        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%x)\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%x)\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%x)\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 Master Example*

6.11.1 esExtendSlaveCoeObjectDictionary

```
EC_T_DWORD esExtendSlaveCoeObjectDictionary (
    EC_T_DWORD dwInstanceId,
    EC_T_WORD wCfgFixedAddress,
    struct _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. 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_INVALIDPARAM*: dwInstanceId is out of range
- *EC_E_INVALIDSTATE*: Simulator instance (dwInstanceId) not initialized, see esInitSimulator()
- *EC_E_NOTFOUND*: No slave matching (wObjectIndex)
- *EC_E_NO_MBX_SUPPORT*: Slave without Mailbox support
- *EC_E_SDO_ABORTCODE_INDEX*: Nonexistent object

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.

Parameters

- **pvContext** – [in] Arbitrarily application-defined parameter passed to callback
- **dwSimulatorId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave fixed address from master request
- **wIndex** – [in] Object Index from master request, e.g. 0x1018
- **bySubindex** – [in] Object SubIndex from master request
- **dwSize** – [in] Read buffer size according to MbxIn Sync Manager size
- **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
- **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
- **bCompleteAccess** – [in] EC_TRUE: CoE SDO Upload with Complete Access. See also #EC_MAILBOX_FLAG_SDO_COMPLETE
- **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
- **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*: Object exists in the simulated slave's object dictionary and is readable in the current EtherCAT state
- *EC_E_SDO_ABORTCODE_INDEX*: 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*: Object SubIndex does not exist in the accessed object in the simulated slave's object dictionary
- *EC_E_SDO_ABORTCODE_DATA_LENGTH_NOT_MATCH*: The read data length does not match the accessed object length
- *EC_E_SDO_ABORTCODE_ACCESS*: 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*: Accessed Entry does not support read operation
- *EC_E_SDO_ABORTCODE_TRANSFER_DEVICE_STATE*: 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.

Parameters

- **pvContext** – [in] Arbitrarily application-defined parameter passed to callback
- **dwSimulatorId** – [in] Simulator Instance ID
- **wCfgFixedAddress** – [in] Slave fixed address from master request
- **wIndex** – [in] Object Index from master request, e.g. 0x40A2
- **bySubindex** – [in] Object SubIndex from master request
- **dwSize** – [in] Write data length
- **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
- **bCompleteAccess** – [in] EC_TRUE: CoE SDO Download with Complete Access. See also #EC_MAILBOX_FLAG_SDO_COMPLETE
- **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
- **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*: Object exists in the simulated slave's object dictionary and is write-able in the current EtherCAT state
 - *EC_E_SDO_ABORTCODE_INDEX*: 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*: Object SubIndex does not exist in the accessed object in the simulated slave's object dictionary
 - *EC_E_SDO_ABORTCODE_DATA_LENGTH_NOT_MATCH*: The write data length does not match the accessed object length

- *EC_E_SDO_ABORTCODE_ACCESS*: 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*: Accessed Entry does not support write operation
- *EC_E_SDO_ABORTCODE_TRANSFER_DEVICE_STATE*: 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,
    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 sub index. 0 or 1 if Complete Access.
- **pbyData** – [in] Buffer containing transfered data
- **dwDataLen** – [in] Buffer length [bytes]
- **dwTimeout** – [in] Timeout [ms]
- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (*EC_MAILBOX_FLAG_SDO_COMPLETE*).

Returns

EC_E_NOERROR or 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. 0 or 1 if Complete Access.
- **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 or error code

6.11.8 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
)
```

Gets a list of object IDs that are available in a slave.

A unique transfer ID must be written into EC_T_MBXTFER.dwTferId. EC_NOTIFY_MBOXRCV is given on completion. This function may not be called from within the JobTask's context.

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] The function will block at most for this time. If the timeout value is set to EC_NOWAIT the function will return immediately.

Returns

EC_E_NOERROR or error code

6.11.9 esCoeGetObjectDescReq

```
EC_T_DWORD esCoeGetObjectDescReq (
    EC_T_DWORD dwInstanceId,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObjIndex,
    EC_T_DWORD dwTimeout
)
```

Determines the description of a specific object.

A unique transfer ID must be written into EC_T_MBXTFER.dwTferId. EC_NOTIFY_MBOXRCV is given on completion. This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **wObjIndex** – [in] Object index
- **dwTimeout** – [in] Timeout [ms] The function will block at most for this time. If the timeout value is set to EC_NOWAIT the function will return immediately.

Returns

EC_E_NOERROR or error code

```
struct EC_T_COE_OBDESC
```

Public Members

EC_T_WORD **wObjIndex**
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!)

6.11.10 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
)
```

Determines the description of a specific object entry.

A unique transfer ID must be written into `EC_T_MBXTFER.dwTferId`. `EC_NOTIFY_MBOXRCV` is given on completion. This function may not be called from within the `JobTask`'s context.

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] The function will block at most for this time. If the timeout value is set to `EC_NOWAIT` the function will return immediately

Returns

EC_E_NOERROR or 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.

6.11.11 CoE transfer Simulator and Master Example

The following example demonstrates how to handle CoE object transfers within the simulator application:

CoE transfer Simulator and Master 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;
    }
}

```

(continues on next page)

(continued from previous page)

```

    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);
        }
        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));
}

```

(continues on next page)

(continued from previous page)

```

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 Master 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));

/* 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;
}

```

(continues on next page)

(continued from previous page)

```

/* 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 */
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 `pfEoeReceive` 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 master isn't initialized
- *EC_E_INVALIDPARAM* 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

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%x)\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.abbyMacAddr[0], oInfo.abbyMacAddr[1], oInfo.abbyMacAddr[2],
        oInfo.abbyMacAddr[3], oInfo.abbyMacAddr[4], oInfo.abbyMacAddr[5],

        oInfo.oIpAddress.sAddr.by[0], oInfo.oIpAddress.sAddr.by[1], oInfo.oIpAddress.sAddr.
↪ by[2], oInfo.oIpAddress.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 Master 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
*/
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_FRAME_TYPE(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;
                    pArpReply->Address.IpV4.SourceMac = MacAddress;
                }
            }
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

        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;
            pIcmpReply->IpHdr.wChecksum = EC_HTONS (EcCalculateCrcRfc1071((EC_T_
↔WORD*)&pIcmpReply->IpHdr, EC_IP_HEADER_MINIMUM_LEN));

            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;
}
}

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)
{
    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 */

```

(continues on next page)

(continued from previous page)

```

    myAppProcessEoeFrameArpAndPing(&oReceiveFrame, &poContext->oSendFrame,
    ↪poContext->oMacAddress, poContext->oIPv4Address);
    if (poContext->oSendFrame.dwSize > 0)
    {
        esEoeSendFrame(dwSimulatorId, wCfgFixedAddress, poContext->oSendFrame.
    ↪pbyFrame, poContext->oSendFrame.dwSize);
    }
}

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);
}

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);
        if (EC_E_NOERROR != dwRes)
        {
            dwRetVal = dwRes;
            goto Exit;
        }
    }

    dwRetVal = EC_E_NOERROR;
}

```

(continues on next page)

(continued from previous page)

```
Exit:
    if (EC_E_NOERROR != dwRetVal)
    {
        SafeOsFree(poContext->oSendFrame.pbyFrame);
    }

    return dwRetVal;
}
```

The callbacks need a context. Each slave needs its individual context. If there is only one slave 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 Master 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;
```

`myAppFoeWrite()` handles download requests from the Master, `myAppFoeWriteData()` copies the FoE payload from the mailbox to `T_MY_CONTEXT::abyFileBuf`:

```
extern "C" EC_T_WORD 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 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;
}
```

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));

oSlaveApp.dwSignature = SIMULATOR_SIGNATURE;
oSlaveApp.dwSize = sizeof(struct _EC_T_SLAVE_SSC_APPL_DESC);
```

(continues on next page)

(continued from previous page)

```
oSlaveApp.szName = (EC_T_CHAR*)"mySlaveAppl";
oSlaveApp.pvContext = poSlaveApplContext; /* pvContext of callbacks */
oSlaveApp.pfnFoeWrite = myAppFoeWrite;
oSlaveApp.pfnFoeWriteData = myAppFoeWriteData;

/* register SlaveSscApplication callbacks (Master in INIT) */
dwRes = esSetSlaveSscApplication(dwSimulatorId, wSlaveAddress, &oSlaveApp);
if (dwRes != EC_E_NOERROR)
{
    goto Exit;
}
```

The following code at Master downloads the file to the slave:

```
/* 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 serve FoE data from a local buffer to the Master. The data served 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 Master, `myAppFoeReadData()` copies the FoE payload from `T_MY_CONTEXT::abyFileBuf` to the mailbox:

```
extern "C" EC_T_WORD 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 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 `EcSimulatorHiIDemo / EcSimulatorSiIDemo`:

```
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 Master uploads the file from the slave:

```

/* 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,
 EC_T_VOID *pvData,
 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
- **pvData** – [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 or error code

6.14.3 VoE Receive Example

The following examples demonstrate how to register a handler for VoE writes from the Master or from other slaves.

VoE Receive Example

The following code demonstrates how to send VoE from the Master to the slave and store it in a buffer at the Simulator. The data received in this example is stored in `T_MY_CONTEXT::abyVoeBuf` :

```
#define MYAPP_FOE_BUFFER_SIZE (35264)
typedef struct
{
    EC_T_BYTE abyFileBuf[MYAPP_FOE_BUFFER_SIZE];
    EC_T_WORD wFileBufLen;

    /* ... */
} T_MY_CONTEXT;
```

`myAppVoeReceiveCallback()` handles write requests from the Master 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 */,
    EC_T_VOID* pvData, EC_T_DWORD dwDataLen)
{
    T_MY_CONTEXT* poContext = (T_MY_CONTEXT*)pvContext;

    OsMemcpy(poContext->abyVoeBuf, pvData, 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 Master to the simulated slave:

```
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));
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);
```

(continues on next page)

(continued from previous page)

```

    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:
    SafeOsDeleteEvent(oContext.pvEvent);

```

6.15 Distributed Clocks (DC)

Distributed Clocks are rudimentary supported.

7 Error Codes

7.1 Groups

No.	Group	Abbr.	Description
1	Application Error	APP	Error within application, running the master E.g. API function call with invalid parameters
2	EtherCAT network information file problem	ENI	Master configuration XML file mismatches slave configuration on bus E.g. Bus Topology Scan cannot detect all slaves configured within network information file
3	Master parameter configuration	CFG	Master configuration parameters erroneous E.g. mailbox command queue not large enough
4	Bus/Slave Error	SLV	Slave 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	Master internal error E.g. Master state machine in undefined state
8	DC Master Sync	DCM	DC slave and host time synchronization
9	Pass-Through-Server	PTS	Initialisation/De-Initialisation errors
10	System Setup	SYS	Errors from Operating System or obviously due to System Setup

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. master 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. stack is busy currently 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 (master 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_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 a 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_PARAMS.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_MBXSEND_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: Master configuration not found (e.g. path to master 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 master 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 master 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 too 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 are 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 mailbox protocol header of the mailbox protocol is wrong (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_PARAMS_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 to 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_RUNTIME

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 an 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 a 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, SIO 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 too 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)