acontis technologies GmbH

SOFTWARE

EC-Monitor

User's Manual

Version 3.1

Edition: September 4, 2023

Contents

1 Introduction 8
1.1 What is EtherCAT? 8
1.2 The EC-Monitor - Features 8
1.3 Protected version 8
1.3.1 Licensing procedure for Development Licenses 8
1.3.2 Licensing procedure for Runtime Licenses 9
1.4 License 9
1.4.1 EC-Monitor license 9
1.4.2 Free Open Source Software contained in EC-Monitor 9
1.4.2.1 Expat XML parser license 9
1.4.3 Free Open Source Software supported by EC-Monitor 10
1.4.3.1 acontis atemsys Linux kernel module 10
1.4.3.2 WinPCap 10
1.4.3.3 Npcap 10

2 Architecture 11
2.1 EtherCAT Network Configuration (ENI) 12
2.2 Operating system configuration 12

3 Ethernet TAP 13
3.1 Generic 100MBit/s Ethernet Switch 13
3.2 Beckhoff ET2000 13
3.3 Dualcomm ETAP-1000 14
3.4 Hilscher netAnalyser 14
3.5 ProfiTap ProfiShark 14

4 Getting Started 15
4.1 Running EcMonitorDemo 15
4.1.1 Command line parameters 16
4.1.1.1 Link Layer 17
4.2 Compiling the EcMonitorDemo 18
4.2.1 Software Development Kit (SDK) 18
4.2.2 Include search path 19
4.2.3 Libraries 19
4.2.4 Preprocessor definitions 20

5 Software Integration 21
5.1 Example application 21
5.1.1 File reference 21
5.1.2 EC-Monitor life cycle 22
5.2 Event notification 24
5.2.1 Direct event notification handling 25
5.2.2 Postponed notification handling 25
5.3 Logging 26
5.4 EtherCAT Network Configuration ENI 27
5.4.1 Single cyclic entry configuration 27
5.4.2 Multiple cyclic entries configuration 28
5.5 Process Data Access 28
5.5.1 Process Data Access Functions 28
5.5.2 Process variables’ offset and size 29
5.5.3 Process variable access via hard coded offsets 29
5.5.4 Process variable access via generated PD Layout 30
5.5.5 Process variable access dynamically from ENI 31
5.5.5.1 emGetCfgSlaveInfo 31

5.5.5.2 emGetSlaveOutpVarInfo . . . 31
5.5.5.3 emFindOutpVarByName . . . 32
5.5.5.4 emGetSlaveOutpVarByObjectEx . . . 32
5.6 EC-Monitor Source Code . . . 33
5.6.1 Link Layer Binaries . . . 33
5.6.2 EC-Monitor Binaries . . . 33
5.6.3 Remote API Server Binaries: . . . 34

6 Platform and Operating Systems (OS) 35
6.1 Linux . . . 35
6.1.1 OS optimizations . . . 35
6.1.1.1 CPUIDLE sub-system . . . 35
6.1.1.2 CPUFREQ sub-system . . . 35
6.1.1.3 ISOLCPUS . . . 36
6.1.2 atemsys kernel module . . . 36
6.1.2.1 atemsys as Device Tree Ethernet Driver . . . 36
6.1.2.2 atemsys and PHY OS Driver . . . 37
6.1.3 Unbind Link Layer instance . . . 37
6.1.3.1 Unbind from kernel driver . . . 37
6.1.3.2 Unload kernel driver . . . 37
6.1.4 Docker . . . 38
6.1.5 Setting up and running EcMonitorDemo . . . 38
6.1.5.1 Run in Docker container . . . 38
6.1.6 OS Compiler settings . . . 39
6.2 QNX Neutrino . . . 39
6.2.1 Thread priority . . . 39
6.2.2 Unbind Link Layer instance . . . 40
6.2.3 IOMMU/SMMU support . . . 40
6.2.4 Setting up and running EcMonitorDemo . . . 40
6.2.5 OS Compiler settings . . . 41
6.3 Windriver VxWorks . . . 41
6.3.1 VxWorks native . . . 41
6.3.2 SNARF Link Layer . . . 42
6.3.3 Setting up and running EcMonitorDemo . . . 42
6.3.4 OS Compiler settings . . . 44
6.4 Microsoft Windows . . . 44
6.4.1 EcMonitorDemo . . . 44
6.4.2 OS Compiler settings . . . 45

7 Link Layer 46
7.1 Link Layer selection . . . 46
7.1.1 Optimized Link Layer drivers . . . 46
7.1.2 Optimized Link Layer drivers and PHY OS Driver . . . 46
7.1.3 Link Layer selection and initialization . . . 47
7.1.4 Link Layer instance selection via PCI location . . . 48
7.2 Windows NDIS - emllNdis . . . 49
7.3 Windows WinPcap - emllPcap . . . 50
7.3.1 WinPcap, Npcap support . . . 51

8 Application programming interface, reference 52
8.1 General functions . . . 53
8.1.1 emInitMonitor . . . 53
8.1.2 emDeinitMonitor . . . 58
8.1.3 emConfigureNetwork . . . 58
8.1.4 emGetMonitorStatus . . . 59
8.1.5 emSetLicenseKey . . . 60
8.1.6 emRegisterClient . . . 61
8.1.7 emUnregisterClient . . . 62
8.1.8 emGetSrcMacAddress . . . 62

8.1.9 emExecJob . . . 62
8.1.10 emGetMonitorParms . . . 65
8.1.11 emSetMonitorParms . . . 66
8.1.12 emGetVersion . . . 66
8.1.13 emGetText . . . 67
8.1.14 emGetMemoryUsage . . . 67
8.1.15 emGetMasterState . . . 67
8.1.16 emGetMasterStateEx . . . 68
8.1.17 emFindInpVarByName - “Inputs.DevicesState” . . . 68
8.1.18 emFindInpVarByName - “Inputs.BusTime” . . . 69
8.1.19 emIoControl . . . 69
8.1.20 emIoControl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB . . . 69
8.1.21 emIoControl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO . . . 70
8.1.22 emIoControl - EC_IOCTL_IS_SLAVETOSLAVE_COMM_CONFIGURED . . . 71
8.2 Packet Capture . . . 71
8.2.1 emOpenPacketCapture . . . 71
8.2.2 emClosePacketCapture . . . 72
8.2.3 emGetPacketCaptureInfo . . . 73
8.2.4 emStartLivePacketCapture . . . 74
8.2.5 emStopLivePacketCapture . . . 74
8.2.6 emBacktracePacketCapture . . . 75
8.3 Process Data functions . . . 75
8.3.1 emGetProcessData . . . 75
8.3.2 emGetProcessDataBits . . . 76
8.3.3 emGetProcessImageInputPtr . . . 77
8.3.4 emGetProcessImageOutputPtr . . . 77
8.3.5 emFindInpVarByName . . . 77
8.3.6 emFindInpVarByNameEx . . . 77
8.3.7 emFindOutpVarByName . . . 78
8.3.8 emFindOutpVarByNameEx . . . 78
8.3.9 emIoControl - EC_IOCTL_GET_PDMEMORYSIZE . . . 79
8.3.10 Process Data access functions . . . 79
8.3.10.1 EC_COPYBITS . . . 79
8.3.10.2 EC_GET_FRM_WORD . . . 80
8.3.10.3 EC_GET_FRM_DWORD . . . 81
8.3.10.4 EC_GET_FRM_QWORD . . . 81
8.3.10.5 EC_GETBITS . . . 81
8.3.11 emIoControl - EC_IOCTL_SET_IGNORE_INPUTS_ON_WKC_ERROR . . . 82
8.3.12 emIoControl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ERROR . . . 82
8.3.13 emIoControl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ZERO . . . 83
8.4 Slave status functions . . . 83
8.4.1 emGetNumConfiguredSlaves . . . 83
8.4.2 emGetNumConnectedSlaves . . . 83
8.4.3 emGetSlaveId . . . 84
8.4.4 emGetSlaveIdAtPosition . . . 84
8.4.5 emGetSlaveState . . . 84
8.4.6 emIsSlavePresent . . . 85
8.4.7 emGetSlaveProp . . . 86
8.4.8 emGetSlaveInpVarInfoNumOf . . . 86
8.4.9 emGetSlaveInpVarInfo . . . 87
8.4.10 emGetSlaveInpVarInfoEx . . . 88
8.4.11 emGetSlaveOutpVarInfoNumOf . . . 90
8.4.12 emGetSlaveOutpVarInfo . . . 90
8.4.13 emGetSlaveOutpVarInfoEx . . . 91
8.4.14 emReadSlaveRegister . . . 91
8.4.15 emGetCfgSlaveInfo . . . 92
8.4.16 emGetBusSlaveInfo . . . 96
8.5 Diagnosis . . . 98

8.5.1 emIoControl - EC_IOCTL_SB_STATUS_GET . . . 99
8.5.2 emIoControl - EC_IOCTL_GET_SLVSTATISTICS . . . 100
8.5.3 emGetSlaveStatistics . . . 101
8.5.4 emIoControl - EC_IOCTL_CLR_SLVSTATISTICS . . . 101
8.5.5 emClearSlaveStatistics . . . 101
8.5.6 emGetDiagnosisImagePtr . . . 102
8.5.7 emGetMasterSyncUnitInfoNumOf . . . 102
8.5.8 emGetMasterSyncUnitInfo . . . 102
8.6 Link Layer Control Interface . . . 103
8.6.1 emIoControl - EC_IOCTL_ISLINK_CONNECTED . . . 103
8.6.2 emIoControl - EC_IOCTL_GET_LINKLAYER_MODE . . . 104
8.6.3 emIoControl - EC_LINKIOCTL... . . . 105
8.6.4 emIoControl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS . . . 105
8.6.5 emIoControl - EC_LINKIOCTL_GET_SPEED . . . 105
8.7 EtherCAT Mailbox Transfer . . . 105
8.7.1 Mailbox transfer object states . . . 107
8.7.2 emMbxTferCreate . . . 107
8.7.3 emMbxTferAbort . . . 111
8.7.4 emMbxTferDelete . . . 111
8.7.5 emNotify - EC_NOTIFY_MBOXRCV . . . 111
8.8 CAN application protocol over EtherCAT (CoE) . . . 112
8.8.1 emNotify - eMbxTferType_COE_SDO_DOWNLOAD . . . 112
8.8.2 emNotify - eMbxTferType_COE_SDO_UPLOAD . . . 113
8.8.3 CoE Emergency (emNotify - eMbxTferType_COE_EMERGENCY) . . . 113
8.8.4 emCoeSdoUpload . . . 114
8.8.5 emCoeSdoUploadReq . . . 115
8.8.6 emCoeGetODList . . . 116
8.8.7 emNotify - eMbxTferType_COE_GETODLIST . . . 117
8.8.8 emNotify - eMbxTferType_COE_GETENTRYDESC . . . 118
8.9 File access over EtherCAT (FoE) . . . 120
8.9.1 Notification sequence . . . 121
8.9.2 emNotify - eMbxTferType_FOE_DOWNLOAD_REQ . . . 122
8.9.3 emNotify - eMbxTferType_FOE_SEG_DOWNLOAD . . . 122
8.9.4 emNotify - eMbxTferType_FOE_UPLOAD_REQ . . . 123
8.9.5 emNotify - eMbxTferType_FOE_SEG_UPLOAD . . . 124
8.9.6 emNotify - EC_NOTIFY_FOE_MBSLAVE_ERROR . . . 125
8.9.7 emConvertEcErrorToFoeError . . . 125
8.10 Hot Connect . . . 125
8.10.1 emHCGetNumGroupMembers . . . 125
8.10.2 emHCGetSlaveIdsOfGroup . . . 126
8.10.3 emNotify - EC_NOTIFY_HC_DETECTADDGROUPS . . . 126
8.10.4 emNotify - EC_NOTIFY_HC_PROBEALLGROUPS . . . 127
8.10.5 emNotify - EC_NOTIFY_HC_TOPOCHGDONE . . . 127

9 Generic notification interface 128
9.1 Notification callback . . . 128
9.2 emNotifyApp . . . 129
9.3 Enable/Disable notifications . . . 129
9.3.1 emIoControl - EC_IOCTL_SET_NOTIFICATION_ENABLED . . . 130
9.3.2 emIoControl - EC_IOCTL_GET_NOTIFICATION_ENABLED . . . 131
9.4 Status notifications . . . 131
9.4.1 emNotify - EC_NOTIFY_STATECHANGED . . . 131
9.4.2 emNotify - EC_NOTIFY_SB_STATUS . . . 132
9.4.3 emNotify - EC_NOTIFY_SB_MISMATCH . . . 132
9.4.4 emNotify - EC_NOTIFY_HC_TOPOCHGDONE . . . 134
9.4.5 emNotify - EC_NOTIFY_SLAVE_PRESENCE . . . 134
9.4.6 emNotify - EC_NOTIFY_SLAVE_STATECHANGED . . . 135
9.4.7 emNotify - EC_NOTIFY_SLAVE_REGISTER_TRANSFER . . . 135

9.5 Error notifications . . . 137
9.5.1 emNotify - EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL . . . 138
9.5.2 emNotify - EC_NOTIFY_ALL_DEVICES_OPERATIONAL . . . 138
9.5.3 emNotify - EC_NOTIFY_CLIENTREGISTRATION_DROPPED . . . 139
9.5.4 emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR . . . 139
9.5.5 emNotify - EC_NOTIFY_FRAME_RESPONSE_ERROR . . . 140
9.5.6 emNotify - EC_NOTIFY_STATUS_SLAVE_ERROR . . . 141
9.5.7 emNotify - EC_NOTIFY_SLAVE_ERROR_STATUS_INFO . . . 141
9.5.8 emNotify - EC_NOTIFY_PDIWATCHDOG . . . 141
9.5.9 emNotify - EC_NOTIFY_COMMUNICATION_TIMEOUT . . . 142
9.5.10 emNotify - EC_NOTIFY_TAP_LINK_STATUS . . . 142

10 RAS-Server for EC-Inspector and EC-Engineer 143
10.1 Integration Requirements . . . 143
10.2 Application programming interface . . . 143
10.2.1 emRasSrvStart . . . 143
10.2.2 emRasSrvStop . . . 145
10.2.3 emRasNotify . . . 145
10.2.4 emRasNotify - ATEMRAS_NOTIFY_CONNECTION . . . 145
10.2.5 emRasNotify - ATEMRAS_NOTIFY_REGISTER . . . 146
10.2.6 emRasNotify - ATEMRAS_NOTIFY_UNREGISTER . . . 146
10.2.7 emRasNotify - ATEMRAS_NOTIFY_MARSHALERROR . . . 147
10.2.8 emRasNotify - ATEMRAS_NOTIFY_ACKERROR . . . 147
10.2.9 emRasNotify - ATEMRAS_NOTIFY_NONOTIFYMEMORY . . . 148
10.2.10 emRasNotify - ATEMRAS_NOTIFY_STDNOTIFYMEMORYSMALL . . . 148
10.2.11 emRasNotify - ATEMRAS_NOTIFY_MBXNOTIFYMEMORYSMALL . . . 148

11 Error Codes 150
11.1 Groups . . . 150
11.2 Generic Error Codes . . . 150
11.3 DCM Error Codes . . . 161
11.4 ADS over EtherCAT (AoE) Error Codes . . . 163
11.5 CAN application protocol over EtherCAT (CoE) SDO Error Codes . . . 165
11.6 File Transfer over EtherCAT (FoE) Error Codes . . . 168
11.7 Servo Drive Profil over EtherCAT (SoE) Error Codes . . . 170
11.8 Remote API Error Codes . . . 174

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 The EC-Monitor - Features

1.3 Protected version

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

Protected
Binary with MAC protection

Unrestricted
Binary without MAC protection

Source
Source code

The protected version will automatically stop after about 30 minutes of continuous operation. In order to remove this restriction a valid runtime license key is required. The runtime license protection is based on the MAC address of the Ethernet controller used for the EtherCAT protocol. With a valid License Key the protected version will automatically become an unrestricted version.

1.3.1 Licensing procedure for Development Licenses

1. Installation of EC-Monitor protected version
2. Determine the MAC Address by calling `emGetSrcMacAddress()` or from a sticker applied on the hardware near the Ethernet controller
3. Send an Email with the subject **Development License Key Request** 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
2;64-31-50-80-20-4E;1B7C1F86-D08E40A8-4F96F2BA
```

5. Activate the License Key by calling `emSetLicenseKey()` with the license key that corresponds to the MAC address on the hardware and check the return code. The license key is 26 characters long.

```
dwRes = emSetLicenseKey(0, "DA1099F2-15C249E9-54327FBC");
```

1.3.2 Licensing procedure for Runtime Licenses

1. Installation of EC-Monitor protected version
2. Determine the MAC Address by calling *emGetSrcMacAddress()* 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.

```
S/N; 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” 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;1B7C1F86-D08E40A8-4F96F2BA
```

6. Activate the License Key by calling *emSetLicenseKey()* with the license key that corresponds to the MAC address on the hardware and check the return code.

```
dwRes = emSetLicenseKey(0, "DA1099F2-15C249E9-54327FBC");
```

1.4 License

1.4.1 EC-Monitor license

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

1.4.2 Free Open Source Software contained in EC-Monitor

1.4.2.1 Expat XML parser license

(continues on next page)

(continued from previous page)

1.4.3 Free Open Source Software supported by EC-Monitor

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

1.4.3.1 acontis atemsys Linux kernel module

The acontis atemsys is licensed under the GPL:

1.4.3.2 WinPCap

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

1.4.3.3 Npcap

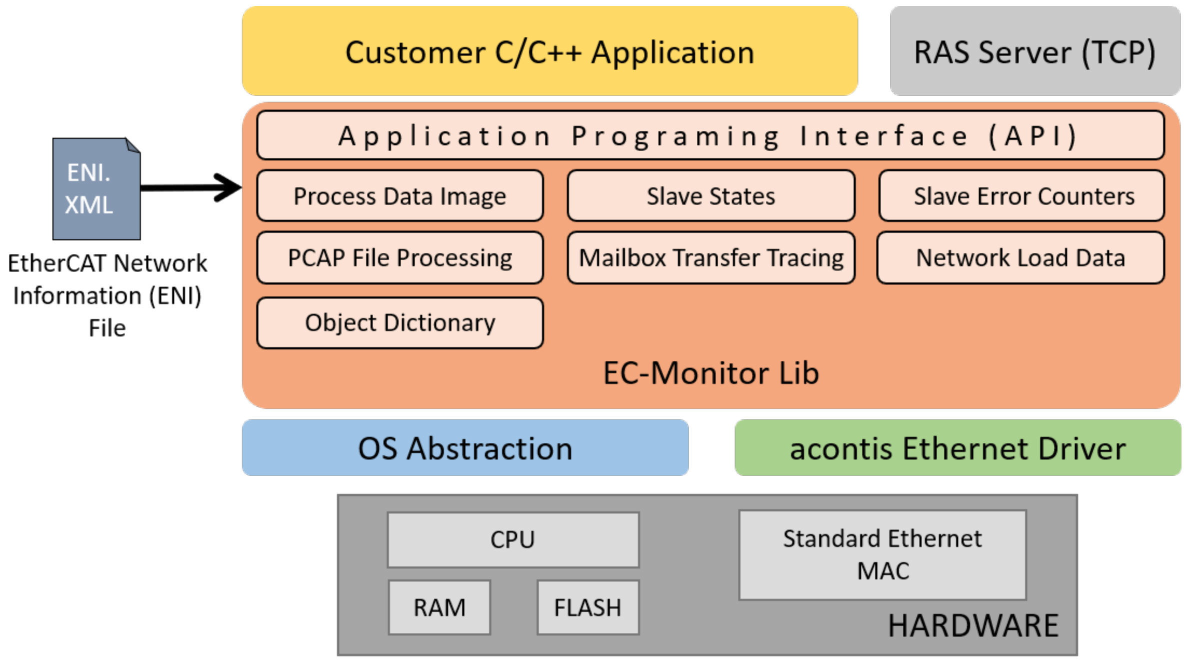
The Npcap library is supported, but not shipped with EC-Monitor.

2 Architecture

The EC-Monitor Software Development Kit (SDK) offers the possibility for Data Tracing / Listening / Sniffing / Logging Diagnosis and Monitoring of EtherCAT Networks. It's suitable for new (Greenfield) and existing (Brownfield) installations. Also it's independent from EtherCAT Master Controller Software and Hardware.

EC-Monitor 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 EC-Monitor can be used in ANSI-C as well as in C++ environments.

The EC-Monitor is divided into modules, see diagram and descriptions below:



EC-Monitor Library:
In the core module cyclic (process data update) and acyclic (mailbox) EtherCAT commands are received and processed.

Configuration Layer:
The EC-Monitor is configured using a XML file whose format is fixed in the EtherCAT specification ETG.2100. EC-Monitor contains an OS independent XML parser.

OS Abstraction 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.

Ethernet Driver Layer:
This layer receives Ethernet frames from the TAP devices.

2.1 EtherCAT Network Configuration (ENI)

The EC-Monitor has to know about the EtherCAT bus topology and the cyclic/acyclic frames which are exchanged by the third party EtherCAT master 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 EtherCAT master vendors and from EtherCAT configuration tools. Thus interoperability between those vendors is guaranteed.

2.2 Operating system configuration

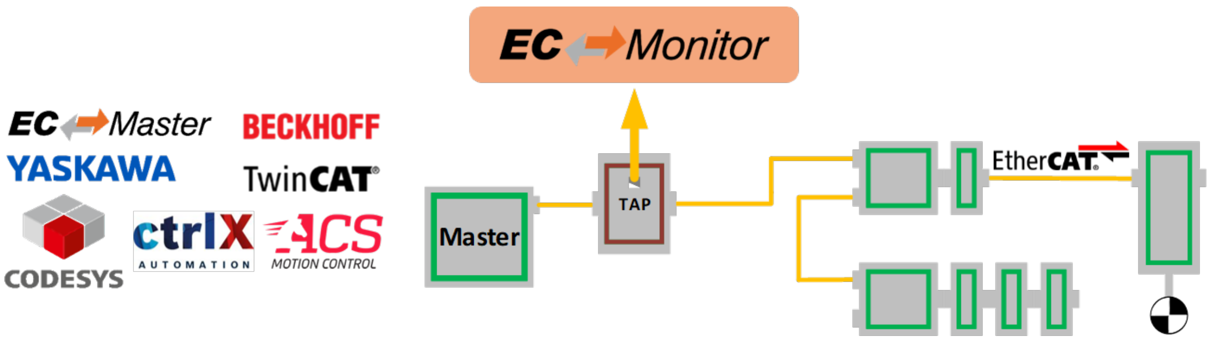
The main task is to setup the operating system to support the appropriate network adapter for EtherCAT usage and for some systems real-time configuration may be needed.

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

3 Ethernet TAP

To capture the EtherCAT traffic, EC-Monitor supports a variety of different Ethernet Test Access Points (TAP). These can be special real-time optimized TAPs with minimal propagation delay and extended diagnostic options, or simple 100MBit/s Ethernet switches. The only requirement is that the input (RX) and output (TX) traffic is forwarded to the EC-Monitor via a common up-link port.

The Ethernet TAP device can be inserted in the network between the EtherCAT master and slaves or, if this is not possible, between two slaves. The position of the Ethernet TAP is detected automatically and the EtherCAT traffic is processed accordingly.



The various Ethernet TAP devices can be automatically detected by the EC-Monitor via `EC_T_MONITOR_INIT_PARMS::eEthTapType` set to `EC_T_ETHERNET_TAP_TYPE::eEthTap_AutoDetect`.

3.1 Generic 100MBit/s Ethernet Switch

A generic 100MBit/s Ethernet switch with at least 3 ports can be used with EC-Monitor. The propagation delay can be up to 150 µs per port and should therefore only be used for slower cycle times.

To manually select this device set `EC_T_MONITOR_INIT_PARMS::eEthTapType` to `EC_T_ETHERNET_TAP_TYPE::eEthTap_Generic`.

3.2 Beckhoff ET2000

The Beckhoff ET2000 comes with propagation delay below 1 µs, a high-precision timestamp and extended frame error detection capabilities.

To manually select this device set `EC_T_MONITOR_INIT_PARMS::eEthTapType` to `EC_T_ETHERNET_TAP_TYPE::eEthTap_Beckhoff_ET2000`.

3.3 Dualcomm ETAP-1000

The Dualcomm ETAP-1000 has a propagation delay below 1 µs. Since it has no other extended capabilities it can be used as generic Ethernet TAP *EC_T_ETHERNET_TAP_TYPE::eEthTap_Generic*.

3.4 Hilscher netAnalyser

The Hilscher netAnalyser comes with propagation delay below 1 µs and it will be detected as generic Ethernet TAP *EC_T_ETHERNET_TAP_TYPE::eEthTap_Generic*.

3.5 ProfiTap ProfiShark

The ProfiTap ProfiShark devices are USB3.0 based Ethernet TAPs with a propagation delay below 1 µs. Since the devices appear as a virtual Ethernet interface in the operating system they can be used as a generic Ethernet TAP *EC_T_ETHERNET_TAP_TYPE::eEthTap_Generic*.

4 Getting Started

To enable a quick and easy start, every EC-Monitor package comes with a pre-compiled EcMonitorDemo executable. This example application handles the following tasks:

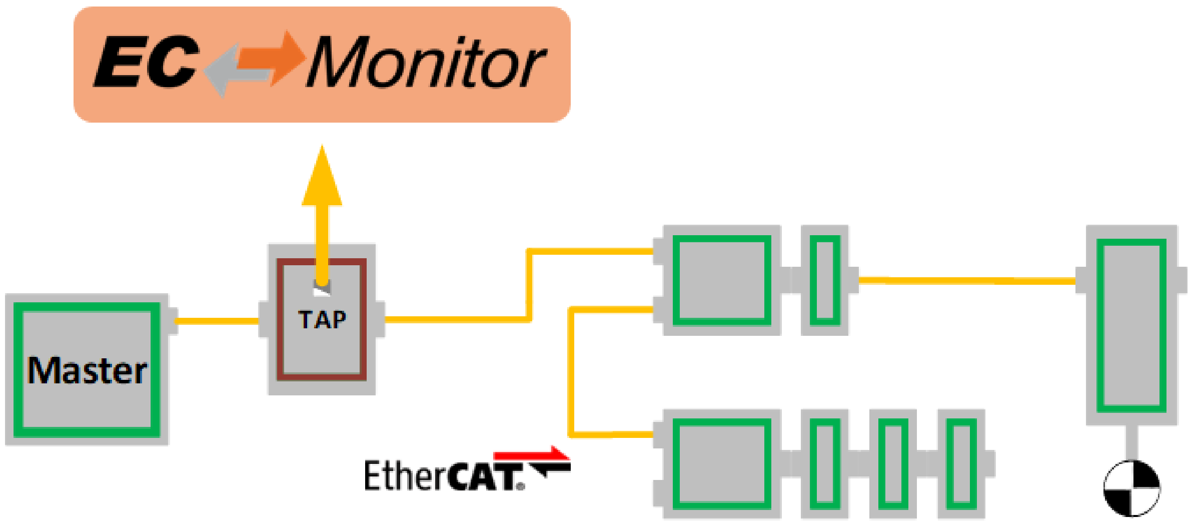
- EC-Monitor initialization
- Process Data acquisition with EC-DAQ
- Periodic Job Task in polling or interrupt mode
- Record and replay wireshark traces
- Logging

See also:

Example application for detailed explanation

4.1 Running EcMonitorDemo

To capture the EtherCAT traffic insert a TAP device after the Master Controller.



Start the EcMonitorDemo from the command line to process the captured EtherCAT frames. At least a Link Layer and a ENI file must be specified.

```
> EcMonitorDemo -winpcap 192.168.157.2 1 -f eni.xml -t 0 -v 3
```

See also:

Platform and Operating Systems (OS) for OS specific additional instructions to run the demo application

4.1.1 Command line parameters

```
EcMonitorDemo <LinkLayer> [-f ENI-FileName] [-t time] [-b cycle time] [-
a affinity] [-v level] [-perf] [-log prefix [msg cnt]] [-lic key] [-
sp [port]] [-auxclk period] [-rec [prefix]] [-play pcap-FileName] [-
dacrec file name]
```

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 µs (1 ms).

<cycle time>
Bus cycle time in µsec

-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 µs 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.

-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

-auxclk `<period>`
Use auxiliary clock

<period>
Clock period in µs (if supported by Operating System).

-rec `[<prefix>]`
Packet capture file recording

<prefix>
File name prefix

-play `<FileName>`
Packet capture file processing

<FileName>
File name (*.pcap|*.pcapng)

-dacrec `<FileName>`

<FileName>
Configuration file

4.1.1.1 Link Layer

Using one of the following demo application Link Layer options, the EC-Monitor 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.

-i8254x `<instance> <mode>`

Hardware: Intel Pro/1000 network adapter card

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

<mode>
0 = Interrupt 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>

`-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

`-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

4.2 Compiling the EcMonitorDemo

The following main rules can be used to generate the example applications for all operating systems.

- `<OS>` is a placeholder for the operating system used.
- `<ARCH>` for the architecture. If different architectures are supported.

4.2.1 Software Development Kit (SDK)

The EC-Monitor development kit is needed to write applications based on the EC-Monitor core. The EC-Monitor core 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-Monitor core
- `/Doc`: Documentation
- `/Examples`: Example applications as source code
- `/SDK`: EtherCAT Software Development Kit containing libraries and header files to build C/C++-applications
- `/SDK/INC`: Header files to be included with the application

- `/SDK/LIB`: Libraries to be linked with the application
- `/SDK/FILES`: Additional files for platform integration
- `/Sources/Common`: Shared source code

4.2.2 Include search path

The header files are located in the following directories:

```
<InstallPath>/SDK/INC/<OS>/<ARCH>
<InstallPath>/SDK/INC
<InstallPath>/Sources/Common
```

4.2.3 Libraries

The libraries are delivered as static, dynamic or both. This is depending on the operating system. They are located in the following directories:

Static libraries

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

EC-Monitor core

```
libEcMonitor.a
```

EC-Monitor RAS server (optional)

```
libEcMonitorRasSrv.a
```

Dynamic libraries

```
<InstallPath>/Bin/<OS>/<ARCH>
```

EC-Monitor core

```
libEcMonitor.so
```

EC-Monitor RAS server (optional)

```
libEcMonitorRasSrv.so
```

Whether it is a Shared Object `*.so` or a Dynamic Link Library `*.dll` depends on the operating system.

4.2.4 Preprocessor definitions

The following preprocessor directives must be set in the build environment or project:

```
EC_MONITOR
```

Exclude the EC-DAQ support in the demo:

```
EXCLUDE_DAQ_SUPPORT
```

5 Software Integration

For the integration of the EC-Monitor, the EcMonitorDemo can be seen as an application framework, serve as a template and be expanded accordingly.

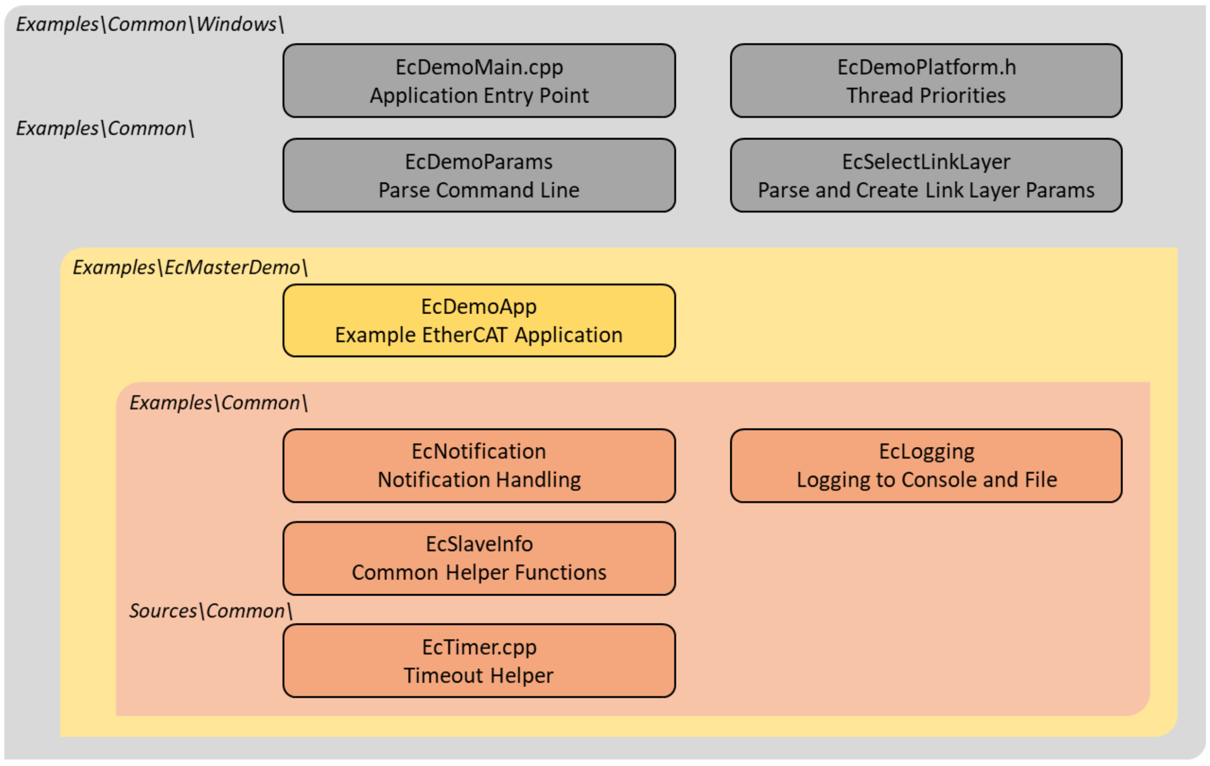
5.1 Example application

The example application will handle the following tasks:

- EC-Monitor initialization
- Process Data acquisition with EC-DAQ
- Periodic Job Task in polling or interrupt mode
- Thread with periodic tasks and application thread already implemented
- Record and replay wireshark traces
- Logging. The output messages of the demo application will be printed on the console as well as in some files.
- "Out of the box" solution for different operating systems: Windows, Linux …

5.1.1 File reference

The EcMonitorDemo application consists of the following files:



EcDemoMain.cpp	Entry point for the different operating systems
EcDemoPlatform.h	Operating system specific settings (task priorities, timer settings)
EcDemoApp.cpp	Initialize, start and terminate EC-Monitor
EcDemoApp.h	Application specific settings for EcDemoApp
EcDemoParms.cpp	Parsing of command line parameters
EcDemoParms.h	Basic configuration parameters
EcSelectLinkLayer.cpp	Common Functions which abstract the command line parsing into Link Layer parameters
EcNotification.cpp	Slave monitoring and error detection (function `ecatNotify()`)
EcSlaveInfo.cpp	Slave information services
EcLogging.cpp	Message logging functions
EcTimer.cpp	Start and monitor timeouts

5.1.2 EC-Monitor life cycle

Basically the operation of the EC-Monitor is wrapped between the functions

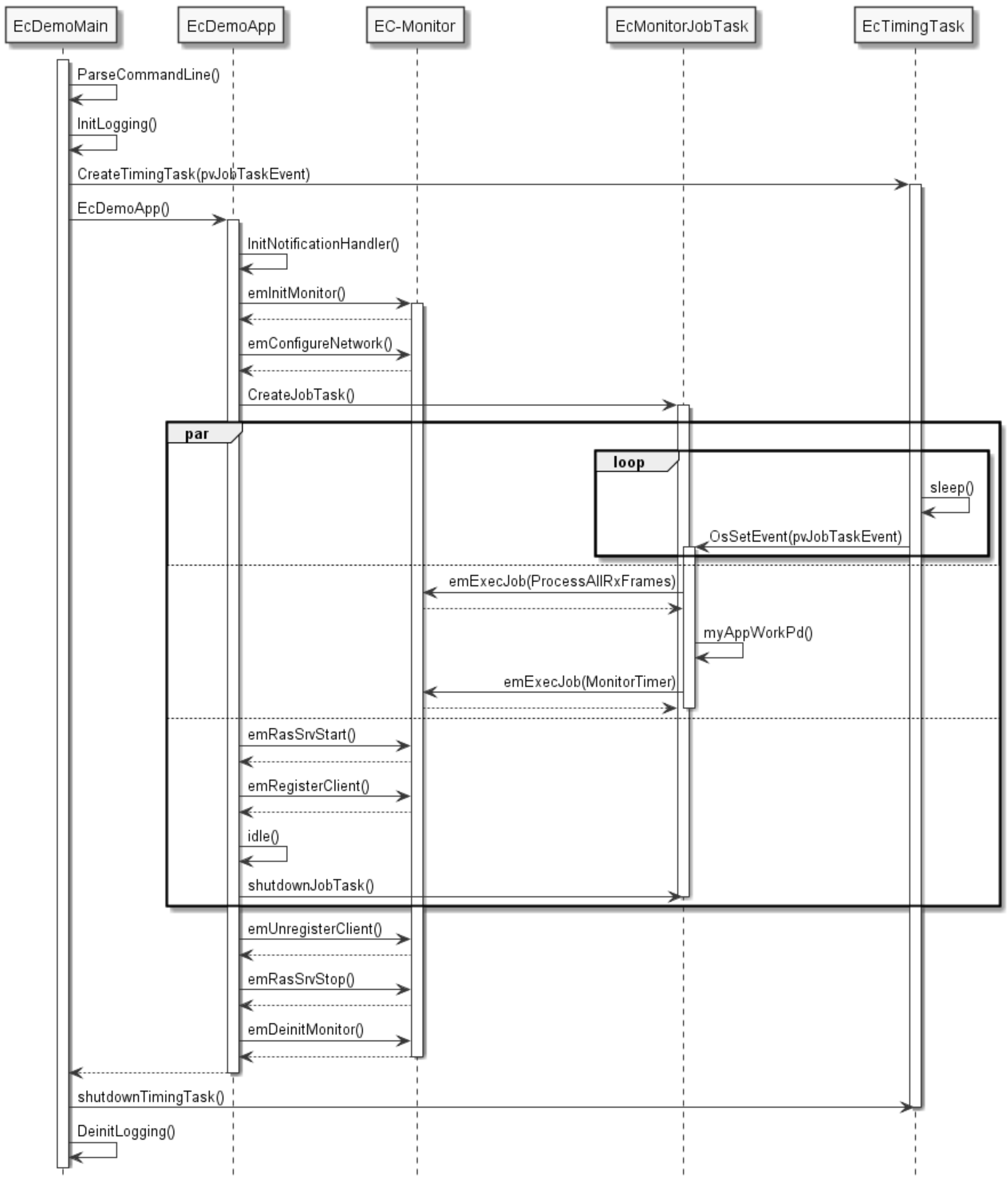
- *`emInitMonitor()`*
- *`emConfigureNetwork()`*

and

- *`emDeinitMonitor()`*

The EC-Monitor is made ready for operation and started with the first two functions mentioned. During this preparation, a thread is set up and started that handles all the cyclic tasks of the EC-Monitor. The last function stops the EC-Monitor and clears the memory.

An overview of the complete life cycle as a sequence diagram:



A more detailed description of the functions:

EcDemoMain()
: A wrapper to start the demo from the respective operating system. In addition to initializing the operating system, parsing command line parameters and initializing logging it also starts the timing task.

EcDemoApp()
: Demo application. The function takes care of starting and stopping the EC-Monitor and all related tasks. In between, the function runs idle, while all relevant work is done by the EcMonitorJobTask().

EcMonitorJobTask()
: Thread that does the necessary periodic work. Very important here is myAppWorkPd() between `eUsrJob_ProcessAllRxFrames` and `eUsrJob_MonitorTimer`. Application-specific access to the process data image can be made here, which is synchronous with the bus cycle.

EcTimingTask()
Timing Thread. This thread sets the timing event that triggers the EcMonitorJobTask for the next cycle.

`emInitMonitor()`
Prepare the EC-Monitor for operation and set operational parameters, e.g. used Link Layer, buffer sizes, maximum number of slaves, … .

`emConfigureNetwork()`
Loads the configuration from the ENI (XML file).

`emRegisterClient()`
Register the application as a client at the EC-Monitor to receive event notifications.

`emDeinitMonitor()`
Clean up.

5.2 Event notification

The EC-Monitor provides event notification for a great number of events. These events are for example:

- Bus state change
- Link state change
- Working counter errors
- …

Any thread can register for these events to be notified. This is achieved by calling the API function

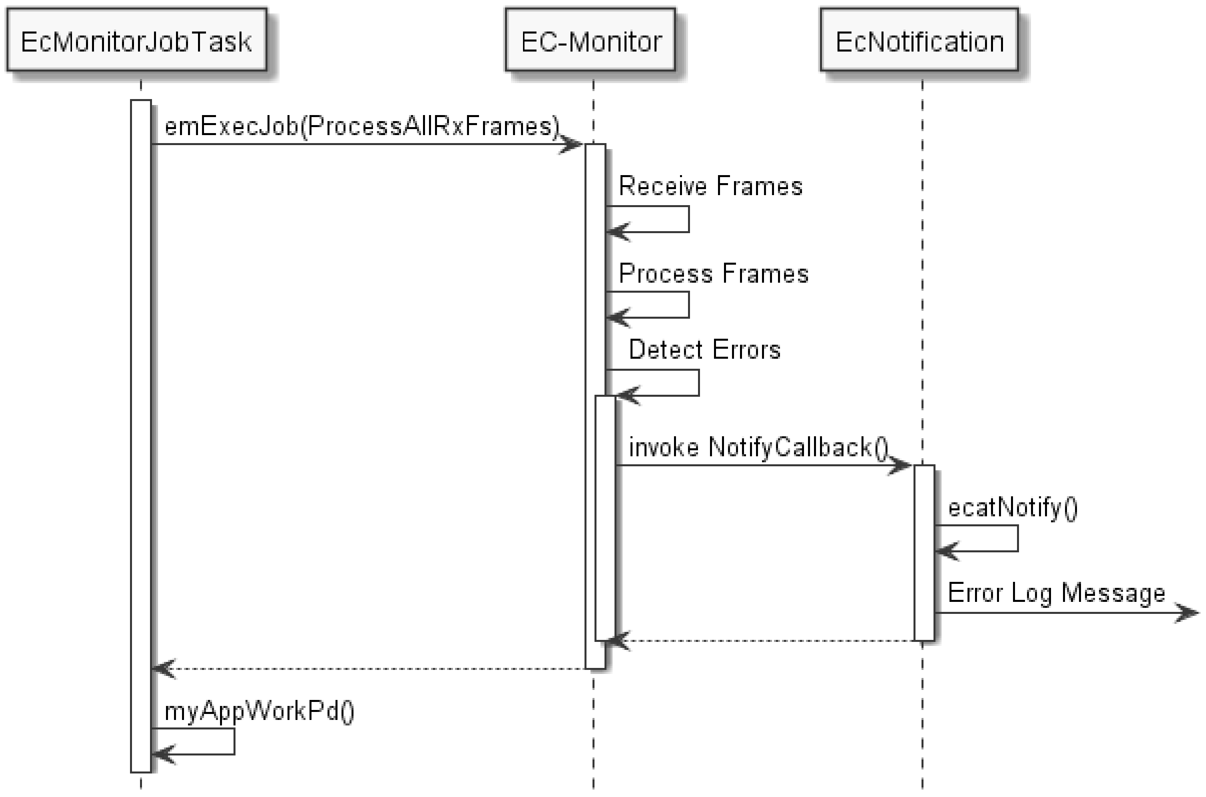
EC_T_DWORD emRegisterClient(*EC_T_DWORD* dwInstanceID, *EC_PF_NOTIFY* pfnNotify, EC_T_VOID *pCallerData, *EC_T_REGISTERRESULTS* *pRegResults)

An example implementation for processing notifications is contained in the class `CEmNotification` of the EcMonitorDemo example, see `Examples/Common/EcNotification.cpp`.. It implements the full framework to catch and process the EC-Monitor notifications. The class is instantiated once and registered at the EC-Monitor with the call `emRegisterClient()`. It contains the method `ecatNotify()` as major entry point (or callback function) for every event notification.

There are two different ways events can be handled. The method of handling an event is primarily determined by the time required to handle the event and the processing context in which the event is to be handled.

5.2.1 Direct event notification handling

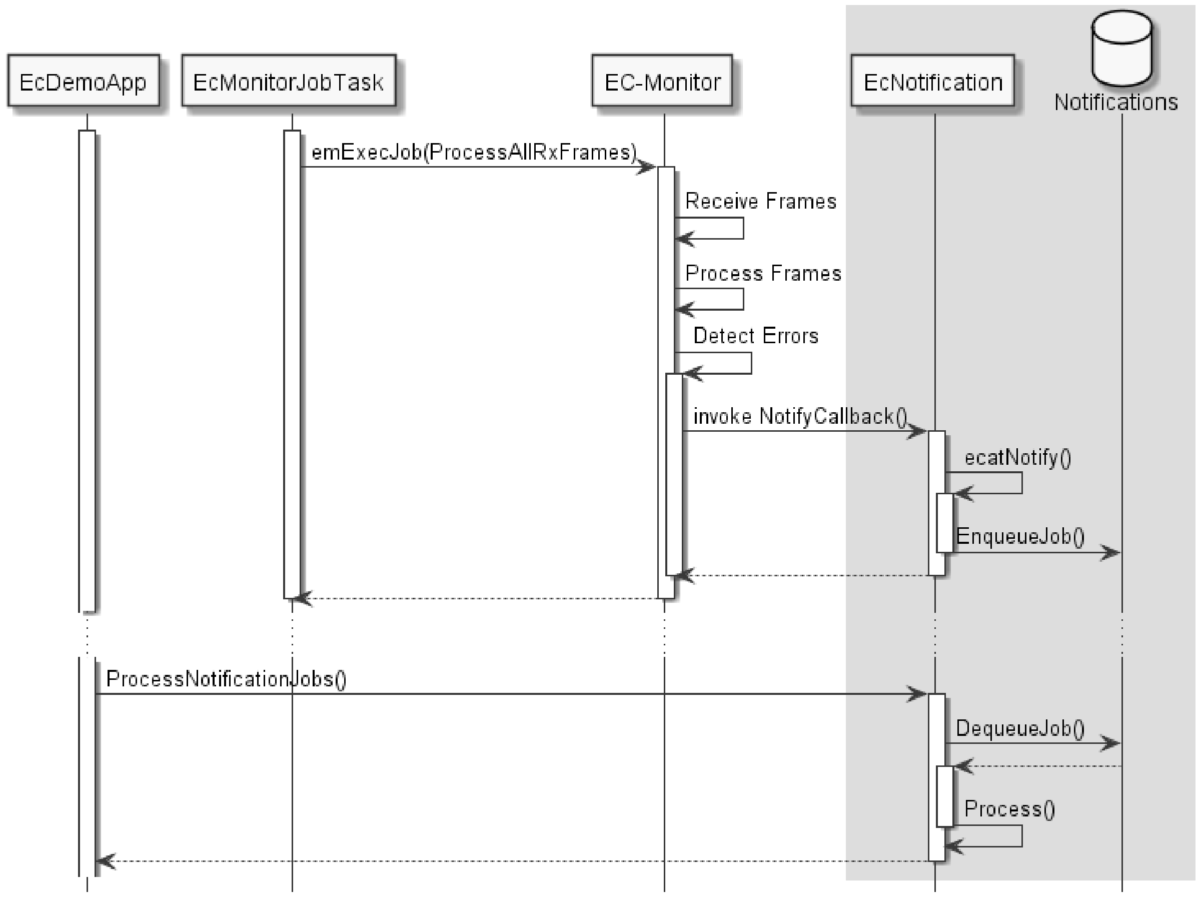
Minor events that take a very short time to process can be handled directly in the context in which they are recognized. A possible example of such an event is the detection of a false working counter (WKC).



The event handling is reduced to simply issuing a log message, which is not time critical. The event is handled directly within the context of the `emExecJob()` function.

5.2.2 Postponed notification handling

Events that require more time-consuming processing cannot be handled directly in the context in which they are detected. The handling or processing of the event must be postponed. This is accomplished through a queue, which is also readily implemented using the `CEmNotification` class.



By calling periodically `CEmNotification::ProcessNotificationJobs()`, the application checks and handles all queued notifications.

Important: The call of `CEmNotification::ProcessNotificationJobs()` shall NOT be executed in the context of `EcMonitorJobTask()`. As the CPU time consumption may be high, this would have a high impact to the real-time behavior of the cyclic operation.

5.3 Logging

The EC-Monitor offers a logging interface for a more detailed analysis of application errors, problems in the EtherCAT network and for diagnosing internal processes. The log messages are passed from the EC-Monitor to the application via the callback *`EC_T_LOG_PARMS::pfLogMsg`* given at *`EC_T_MONITOR_INIT_PARMS::LogParms`*.

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

The level of detail of the logging output can be set via *`EC_T_LOG_PARMS::dwLogLevel`*. The log levels are firmly defined:

EC_LOG_LEVELS

- **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**

For performance reasons, the log messages are automatically filtered based on the log level and then passed to the callback.

Example

The `EcMonitorDemo` examples demonstrate how log messages can be processed by the application, see `Examples/Common/EcLogging.cpp`. The messages processed by `EcLogging.cpp` are of different types, e.g. EC-Monitor log messages and application messages are logged to the console and/or files. Identical messages are skipped automatically by default.

Note: With some operating systems, logging in files is deactivated, e.g. because a file system is not available.

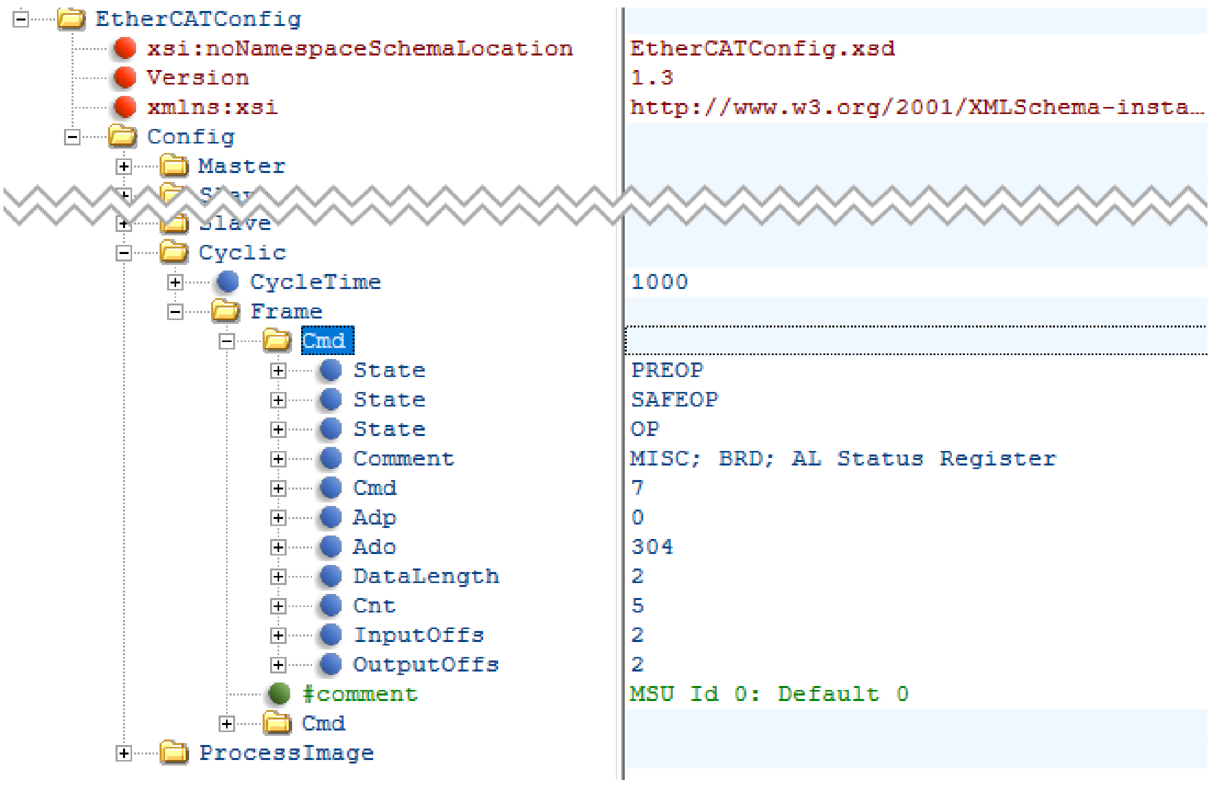
The verbosity of the `EcMonitorDemo` is specified as a `-v` command line parameter. It is used to determine the log level of the application, see `EcDemoMain.cpp`. `EcLogging.cpp` has various parameters beside the log level, like Roll Over setting, log task priority, CPU affinity, log buffer size and etc.

5.4 EtherCAT Network Configuration ENI

The EtherCAT configuration file ENI contains one or more *Cyclic* entries for reading new input data values and output data values (process data update). These entries contain one or more frames, so-called cyclic frames, which are to be sent cyclically by the EtherCAT master. Within the cyclic frames are one or more EtherCAT datagrams that contain logical read/write commands for accessing the process data values.

5.4.1 Single cyclic entry configuration

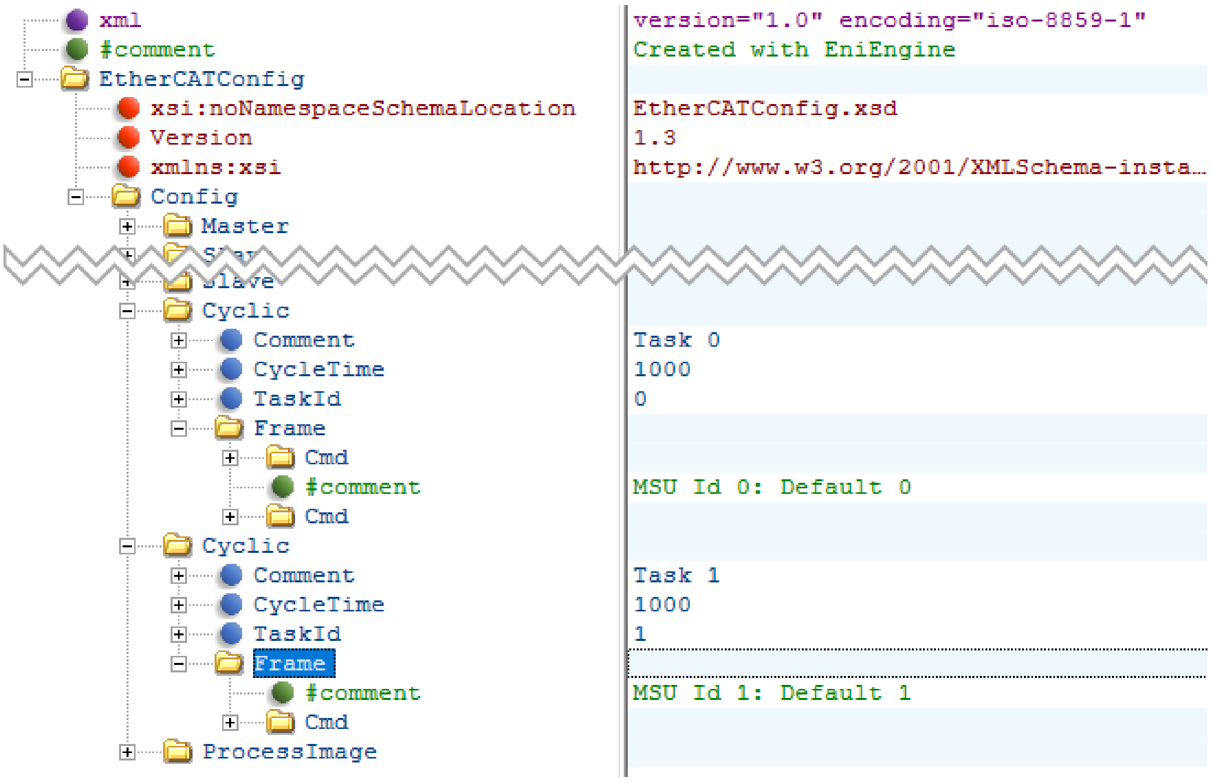
In the simplest case, there is only a single cyclic entry with one or more cyclic frames.



```
EtherCATConfig
    xsi:noNamespaceSchemaLocation        EtherCATConfig.xsd
    Version                              1.3
    xmlns:xsi                            http://www.w3.org/2001/XMLSchema-insta…
    Config
        Master
        Slave
        Cyclic
            CycleTime                    1000
            Frame
                Cmd
                    State                PREOP
                    State                SAFEOP
                    State                OP
                    Comment              MISC; BRD; AL Status Register
                    Cmd                  7
                    Adp                  0
                    Ado                  304
                    DataLength           2
                    Cnt                  5
                    InputOffs            2
                    OutputOffs           2
                #comment                 MSU Id 0: Default 0
                Cmd
        ProcessImage
```

5.4.2 Multiple cyclic entries configuration

For more complex scenarios it is possible to configure the system using multiple cyclic entries with one or more cyclic frames for each cyclic entry.



```
xml                                      version="1.0" encoding="iso-8859-1"
#comment                                 Created with EniEngine
EtherCATConfig
  xsi:noNamespaceSchemaLocation          EtherCATConfig.xsd
  Version                                1.3
  xmlns:xsi                              http://www.w3.org/2001/XMLSchema-insta…
  Config
    Master
    Slave
    Slave
    Cyclic
      Comment                            Task 0
      CycleTime                          1000
      TaskId                             0
      Frame
        Cmd
        #comment                         MSU Id 0: Default 0
        Cmd
    Cyclic
      Comment                            Task 1
      CycleTime                          1000
      TaskId                             1
      Frame
        #comment                         MSU Id 1: Default 1
        Cmd
    ProcessImage
```

5.5 Process Data Access

The process data that is exchanged between an EtherCAT master and the slaves in each cycle is stored in the process data image. There are two separate memory areas, one for the input data and one for the output data. The base addresses of these areas are provided by calling the functions *emGetProcessImageInputPtr()* and *emGetProcessImageOutputPtr()*. The size of the process data input image is defined in the ENI file under `EtherCATConfig/Config/ProcessImage/Inputs/ByteSize` and `EtherCATConfig/Config/ProcessImage/Outputs/ByteSize` and is returned by *emRegisterClient()* at *EC_T_REGISTERRESULTS::dwPDOutSize* and *EC_T_REGISTERRESULTS::dwPDInSize*.

5.5.1 Process Data Access Functions

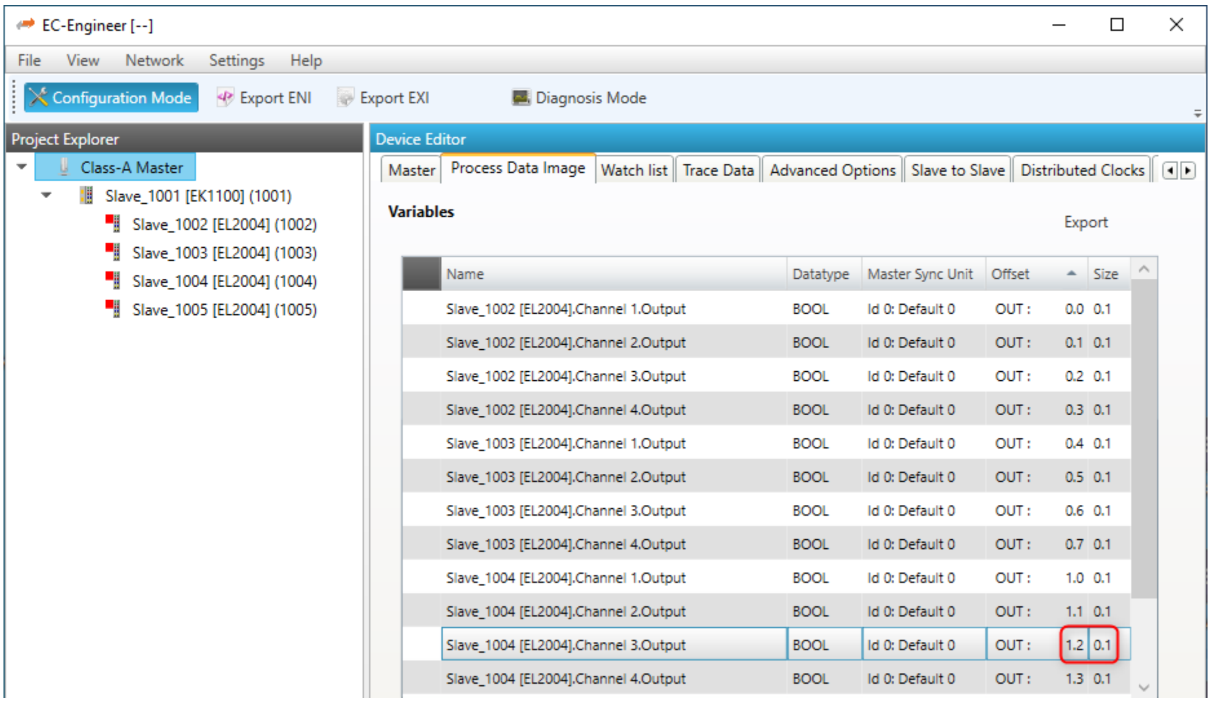
Process data variables that are packed as array of bits are bit aligned and not byte aligned in process data. Accessing bits that are bit aligned and not byte aligned should be done using *EC_GETBITS*. See *EC_COPYBITS* for how to copy data areas with bit offsets that are not byte aligned. Access to corresponding aligned variables, e.g. of the types *EC_T_BYTE*, *EC_T_WORD*, *EC_T_DWORD*, *EC_T_UINT64*, can be accessed more efficiently using the appropriate macros according to the following table.

Note: Process data is typically transmitted as little endian and must therefore be swapped on big endian systems in order to be correctly interpreted.

Variable type	Bit size	Macro	Hint
Bit	1	EC_GETBITS	Contains swap for big endian systems
EC_T_BYTE	8	N/A	Bytes can be directly addressed at pbyBuffer[BitOffset/8]
EC_T_WORD	16	EC_GET_FRM_WORD	Contains swap for big endian systems
EC_T_DWORD	32	EC_GET_FRM_DWORD	Contains swap for big endian systems
EC_T_UINT64	64	EC_GET_FRM_QWORD	Contains swap for big endian systems

5.5.2 Process variables' offset and size

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


EC-Engineer [--]
File View Network Settings Help
Configuration Mode Export ENI Export EXI Diagnosis Mode
Project Explorer
Class-A Master
Slave_1001 [EK1100] (1001)
Slave_1002 [EL2004] (1002)
Slave_1003 [EL2004] (1003)
Slave_1004 [EL2004] (1004)
Slave_1005 [EL2004] (1005)
Device Editor
Master Process Data Image Watch list Trace Data Advanced Options Slave to Slave Distributed Clocks
Variables
Export

Name	Datatype	Master Sync Unit		Offset	Size
Slave_1002 [EL2004].Channel 1.Output	BOOL	Id 0: Default 0	OUT :	0.0	0.1
Slave_1002 [EL2004].Channel 2.Output	BOOL	Id 0: Default 0	OUT :	0.1	0.1
Slave_1002 [EL2004].Channel 3.Output	BOOL	Id 0: Default 0	OUT :	0.2	0.1
Slave_1002 [EL2004].Channel 4.Output	BOOL	Id 0: Default 0	OUT :	0.3	0.1
Slave_1003 [EL2004].Channel 1.Output	BOOL	Id 0: Default 0	OUT :	0.4	0.1
Slave_1003 [EL2004].Channel 2.Output	BOOL	Id 0: Default 0	OUT :	0.5	0.1
Slave_1003 [EL2004].Channel 3.Output	BOOL	Id 0: Default 0	OUT :	0.6	0.1
Slave_1003 [EL2004].Channel 4.Output	BOOL	Id 0: Default 0	OUT :	0.7	0.1
Slave_1004 [EL2004].Channel 1.Output	BOOL	Id 0: Default 0	OUT :	1.0	0.1
Slave_1004 [EL2004].Channel 2.Output	BOOL	Id 0: Default 0	OUT :	1.1	0.1
Slave_1004 [EL2004].Channel 3.Output	BOOL	Id 0: Default 0	OUT :	1.2	0.1
Slave_1004 [EL2004].Channel 4.Output	BOOL	Id 0: Default 0	OUT :	1.3	0.1

Accessing the process data of a specific slave always works by adding an offset to the base address. All offsets are given as bit offsets!

There are different ways possible to get this offset. The offset values will not change until a new configuration is provided, therefore it is sufficient to load them once right after `emConfigureNetwork()`, it is not needed every cycle.

5.5.3 Process variable access via hard coded offsets

The offset value can be determined from an EtherCAT configuration tool. It is not recommended to use fixed values as the offsets will change as slaves are added/removed from the configuration.

As shown in the screenshot above, *Slave_1004 [EL2004].Channel 3.Output* is at offset 1.2 with size 0.1 in the example.

The numbering is *Byte.Bit* so the offset in the example is *Byte 1*, *Bit 2*, Bit offset:

$$8 * 1 + 2 = 10$$

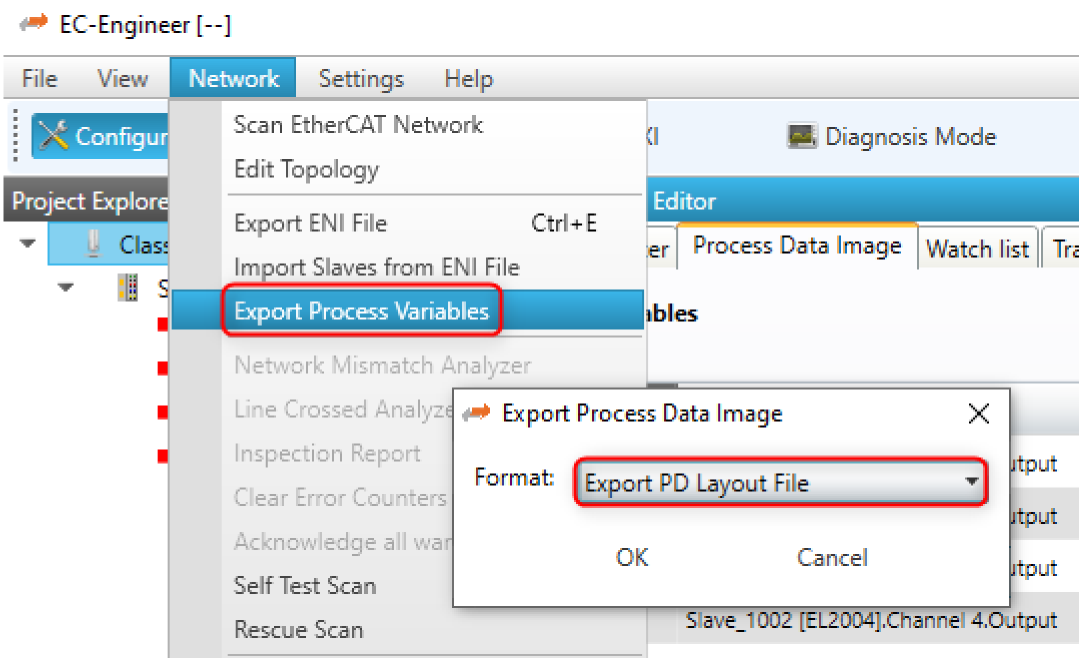
Bit size

$$0 * 8 + 1 = 1$$

```
EC_T_BYTE* pbyPdOut = emGetProcessImageInputPtr(dwInstanceId);
EC_T_BYTE  byValue = 0x00;
EC_T_DWORD dwBitOffset = 10;
EC_T_DWORD dwBitSize = 1;

/* get variable in process data */
EC_GETBITS(pbyPdOut, &byValue, dwBitOffset, dwBitSize);
```

5.5.4 Process variable access via generated PD Layout

The EC-Engineer / EC-Inspector can export the process variables to a PD-Layout C-Header via the menu item *Network ‣ Export Process Variables* as shown in the following screenshot:



Format: Export PD Layout File

This will generate a header file containing the slaves’ variables as follows:

```
#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 how a value can be accessed:

```
EC_T_BYTE* pbyPdOut = emGetProcessImageOutputPtr(dwInstanceId);
T_PDLAYOUT_OUT_SLAVE_2002* pPdOutSlave2002 = (T_PDLAYOUT_OUT_SLAVE_2002*)(pbyPdOut␣
↪+ PDLAYOUT_OUT_OFFSET_SLAVE_2002);

EC_T_WORD wChannel1Out = EC_GET_FRM_WORD(&pPdOutSlave2002->swChannel_1_Output);
```

5.5.5 Process variable access dynamically from ENI

5.5.5.1 emGetCfgSlaveInfo

The slave offsets can be determined dynamically with the function `emGetCfgSlaveInfo()`. The offsets are stored in `EC_T_CFG_SLAVE_INFO::dwPdOffsIn` and `EC_T_CFG_SLAVE_INFO::dwPdOffsOut`.

Example of how *Slave_1004 [EL2004].Channel 3.Output* can be accessed:

```
EC_T_CFG_SLAVE_INFO SlaveInfo;
dwRes = emGetCfgSlaveInfo(dwInstanceId, EC_TRUE, 1004, &SlaveInfo);

EC_T_BYTE* pbyPdOut = emGetProcessImageOutputPtr(dwInstanceId);
EC_T_BYTE  byValue = 0x00;
EC_T_DWORD dwBitOffset = SlaveInfo.dwPdOffsOut + 2;
EC_T_DWORD dwBitSize = 1;

/* get variable in process data */
EC_GETBITS(pbyPdOut, &byValue, dwBitOffset, dwBitSize);
```

5.5.5.2 emGetSlaveOutpVarInfo

All variables of a specific slave can be determined dynamically with the functions `emGetSlaveInpVarInfoEx()` or `emGetSlaveOutpVarInfoEx()`. The offset is stored in `EC_T_PROCESS_VAR_INFO_EX::nBitOffs`.

Example of how *Slave_1004 [EL2004].Channel 3.Output* can be accessed:

```
EC_T_WORD wNumSlaveVars = 0;
EC_T_WORD wNumVarsRead = 0;
EC_T_PROCESS_VAR_INFO_EX* aProcVarInfo = EC_NULL;

/* get number of output variables */
dwRes = emGetSlaveOutpVarInfoNumOf(dwInstanceId, EC_TRUE, 1004, &wNumSlaveVars);

/* allocate buffer for the variable info structs */
aProcVarInfo = (EC_T_PROCESS_VAR_INFO_EX*)OsMalloc(sizeof(EC_T_PROCESS_VAR_INFO_
↪EX) * wNumSlaveVars);
OsMemset(aProcVarInfo, 0, sizeof(EC_T_PROCESS_VAR_INFO_EX) * wNumSlaveVars);

/* read all variables of the slave at once */
dwRes = emGetSlaveOutpVarInfoEx(dwInstanceId, EC_TRUE, 1004, wNumSlaveVars,␣
↪aProcVarInfo, &wNumVarsRead);

EC_T_BYTE* pbyPdOut = emGetProcessImageOutputPtr(dwInstanceId);
EC_T_BYTE  byValue = 0x00;

/* get variable in process data */
EC_GETBITS(pbyPdOut, &byValue, aProcVarInfo[0].nBitOffs, aProcVarInfo[0].nBitSize);
```

5.5.5.3 emFindOutpVarByName

The variable offsets can be determined dynamically using the names with the functions *emFindInpVarByNameEx()* or *emFindOutpVarByNameEx()*. Each input or output has a unique variable name, all variables names are stored in the ENI file under `EtherCATConfig/Config/ProcessImage/[Inputs|Outputs]/Variable`. The offset is stored in *EC_T_PROCESS_VAR_INFO_EX::nBitOffs*.

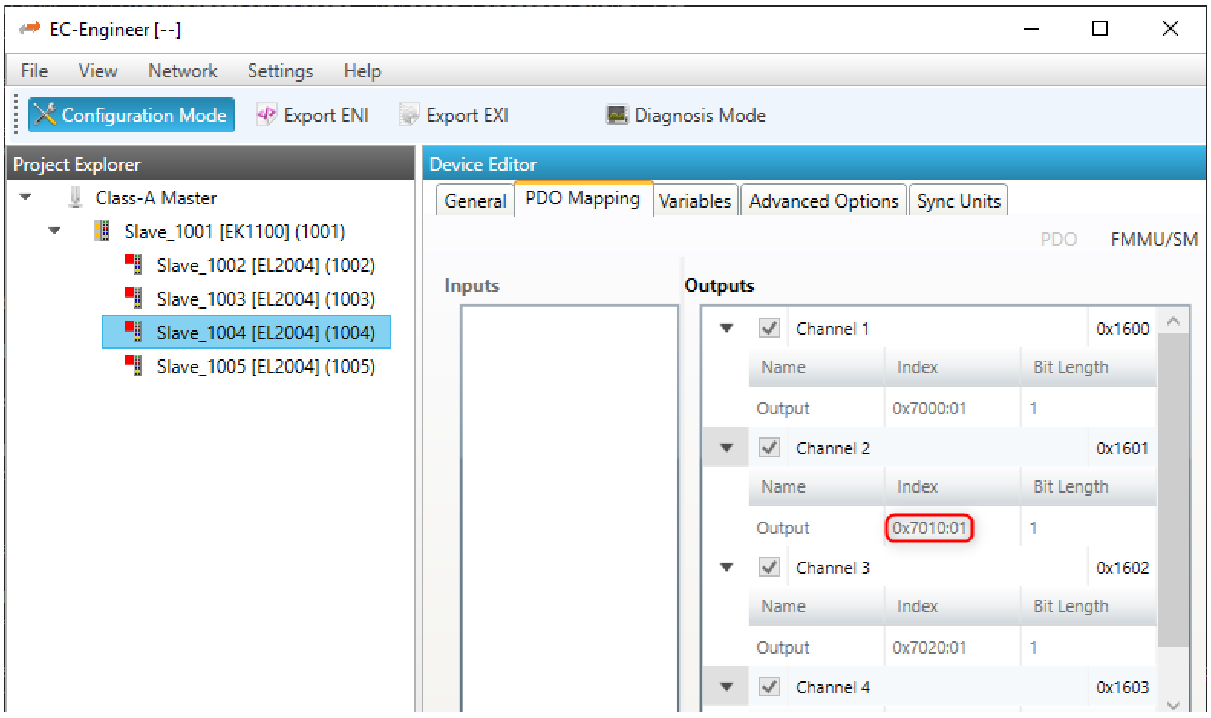
Example of how *Slave_1004 [EL2004].Channel 3.Output* can be accessed:

```
EC_T_PROCESS_VAR_INFO_EX ProcVarInfo;
dwRes = emFindOutpVarByNameEx(dwInstanceId, "Slave_1004 [EL2004].Channel 3.Output",
↪ &ProcVarInfo);

EC_T_BYTE* pbyPdOut = emGetProcessImageOutputPtr(dwInstanceId);
EC_T_BYTE  byValue = 0x00;

/* get variable in process data */
EC_GETBITS(pbyPdOut, &byValue, ProcVarInfo.nBitOffs, ProcVarInfo.nBitSize);
```

5.5.5.4 emGetSlaveOutpVarByObjectEx

The variable offsets can be determined dynamically using the object index and subindex with the functions `emGetSlaveInpVarByObjectEx()` or `emGetSlaveOutpVarByObjectEx()`.

The object index and subindex can be get with the EC-Engineer:



Channel 1 0x1600

Name	Index	Bit Length
Output	0x7000:01	1

Channel 2 0x1601

Name	Index	Bit Length
Output	0x7010:01	1

Channel 3 0x1602

Name	Index	Bit Length
Output	0x7020:01	1

Channel 4 0x1603

Example of how *Slave_1004 [EL2004].Channel 3.Output* can be accessed:

```
EC_T_PROCESS_VAR_INFO_EX ProcVarInfo;
dwRes = emGetSlaveOutpVarByObjectEx(dwInstanceId, EC_TRUE, 1004, 0x7010, 0x01, &
↪ProcVarInfo);

EC_T_BYTE* pbyPdOut = emGetProcessImageOutputPtr(dwInstanceId);
EC_T_BYTE  byValue = 0x00;
```

(continues on next page)

(continued from previous page)

```
/* get variable in process data */
EC_GETBITS(pbyPdOut, &byValue, ProcVarInfo.nBitOffs, ProcVarInfo.nBitSize);
```

5.6 EC-Monitor Source Code

In a source code delivery the EC-Monitor sources are divided into 4 parts:

- SDK Header files
- Link layer files (multiple Link Layers may be shipped)
- Link OS layer files (only valid for the Link Layers)
- EC-Monitor files (configuration, core and interface layer)
- OS layer files

The EC-Monitor 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 EC-Monitor 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 EC-Monitor source and header files.

For most platforms three separate independent binaries will have to be generated:

1. Link Layer Binary. The Link Layer binary will be dynamically bound to the application at runtime.
2. EC-Monitor Library
3. Remote API Server Library

5.6.1 Link Layer Binaries

The following files have to be included into an IDE project or makefile:

- Link layer files. Only one single Link Layer must be selected even if multiple Link Layers are shipped. For each Link Layer 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 Link Layer type (e.g. emllI8255x.dll for the I8255x Link Layer).

5.6.2 EC-Monitor Binaries

The following files have to be included into an IDE project or makefile:

- EC-Monitor 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.

5.6.3 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:

Platform and Operating Systems (OS) for required tool chain settings

6 Platform and Operating Systems (OS)

6.1 Linux

6.1.1 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-Master application on a dedicated CPU core that is isolated from the Linux scheduler (*ISOLCPUS*) can provide additional stability.

6.1.1.1 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
```

6.1.1.2 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
```

6.1.1.3 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
```

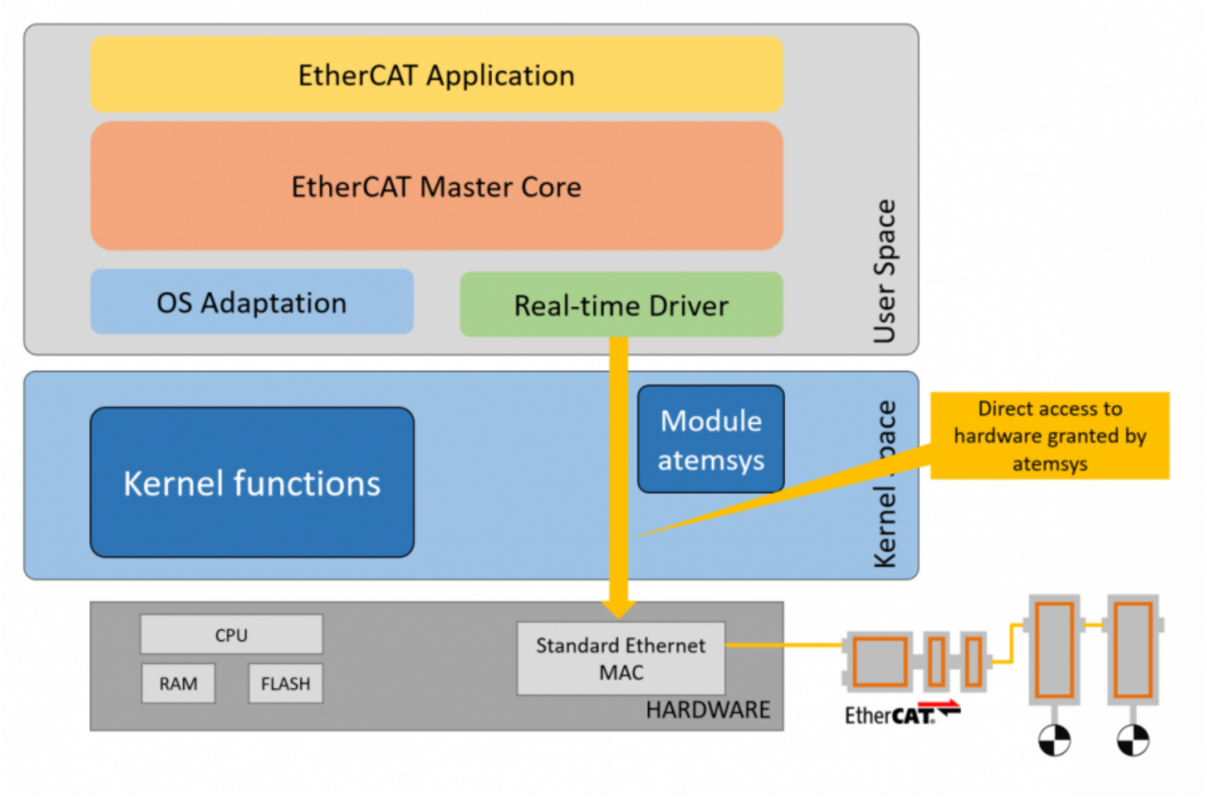
Running EcMonitorDemo on the isolated CPU core by setting the CPU affinity -a:

```
$ ./EcMonitorDemo -a 3
```

6.1.2 atemsys kernel module

To use Optimized Link Layers 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



6.1.2.1 atemsys as Device Tree Ethernet Driver

atemsys can also be used as a device tree driver to avoid certain conflicts between the link layer 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 link layers/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.

6.1.2.2 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.

6.1.3 Unbind Link Layer instance

Link Layer instances used by optimized Link Layers 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.

6.1.3.1 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
```

6.1.3.2 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 optimized Link Layers:

```
$ 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
```

6.1.4 Docker

It is possible to operate EC-Master 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
```

6.1.5 Setting up and running EcMonitorDemo

1. Unbind Link Layer 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-Master package `/bin` and a `eni.xml` to directory e.g. `/tmp`.
4. Adjust *LD_LIBRARY_PATH* search locations for Optimized Link Layers if necessary, e.g.

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

5. Run EcMonitorDemo

   ```
   $ cd /tmp
   $ ./EcMonitorDemo -f eni.xml -i8254x 1 1 -perf
   ```

See also:

Running EcMonitorDemo

6.1.5.1 Run in Docker container

1. Unbind Link Layer instance and load atemsys on the host.
2. Create a directory on the host (e.g. `~/docker`) and copy files from EC-Master package `/bin` and `eni.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 EcMonitorDemo in container

```
# cd /home/docker
# export LD_LIBRARY_PATH=.
# ./EcMonitorDemo -f eni.xml -i8254x 2 1 -perf
```

6.1.6 OS Compiler settings

Besides the general settings from *Compiling the EcMonitorDemo* the following settings are necessary to build the example application for Linux

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

```
pthread dl rt
```

6.2 QNX Neutrino

6.2.1 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!

6.2.2 Unbind Link Layer 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.

QNX6.3

```
umount /dev/io-net/en1
```

QNX >= 6.5

```
ifconfig en1 destroy
```

QNX >= 7.1

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

6.2.3 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 Link Layer. 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: Link Layer emllI8254x with instance number 1

```
smm_emllI8254x-0x00000001
```

A separate typed memory region must be defined for each Link Layer instance. The typed memory is automatically used by the Link Layer if it matches the pattern, otherwise the default memory is used.

6.2.4 Setting up and running EcMonitorDemo

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 Link Layer instance, e.g.

   ```
   $ ifconfig en1 destroy
   ```

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

4. Adjust *LD_LIBRARY_PATH* search locations for Optimized Link Layers if necessary, e.g.

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

5. Run EcMonitorDemo

   ```
   $ cd /tmp
   $ ./EcMonitorDemo -i8254x 1 1 -f eni.xml
   ```

See also:

Running EcMonitorDemo

6.2.5 OS Compiler settings

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

Extra include paths

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

Extra source paths

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

Extra library paths to the main EtherCAT components

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

Extra libraries

```
socket
```

6.3 Windriver VxWorks

Optimized Link Layers for VxWorks are available. If none of the optimized Link Layers can be used, the SNARF Link Layer must be selected.

6.3.1 VxWorks native

The BSP has to be prepared to support Optimized Link Layers:

1. To use an optimized Link Layer the adapter memory has to be mapped into VxWorks memory space (VxWorks 5.x only). I.e. for the Intel Pro/100 Link Layer 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},
```

(continues on next page)

(continued from previous page)

```
#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

6.3.2 SNARF Link Layer

The SNARF Link Layer is only needed if none of the optimized Link Layers can be used. The appropriate network adapter drivers have to be added to the VxWorks image.

6.3.3 Setting up and running EcMonitorDemo

1. **VxWorks OS configuration**
   See sections above.
2. **Determine the network interface**
   Using the command line option the network interface card and Link Layer 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-Master 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 a Link Layer module**
   The Link Layer library (e.g. `emllI8254x.out`) which contains hardware support for the corresponding NIC must be downloaded. By default the Link Layers 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 (`eni.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 `eni.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
```

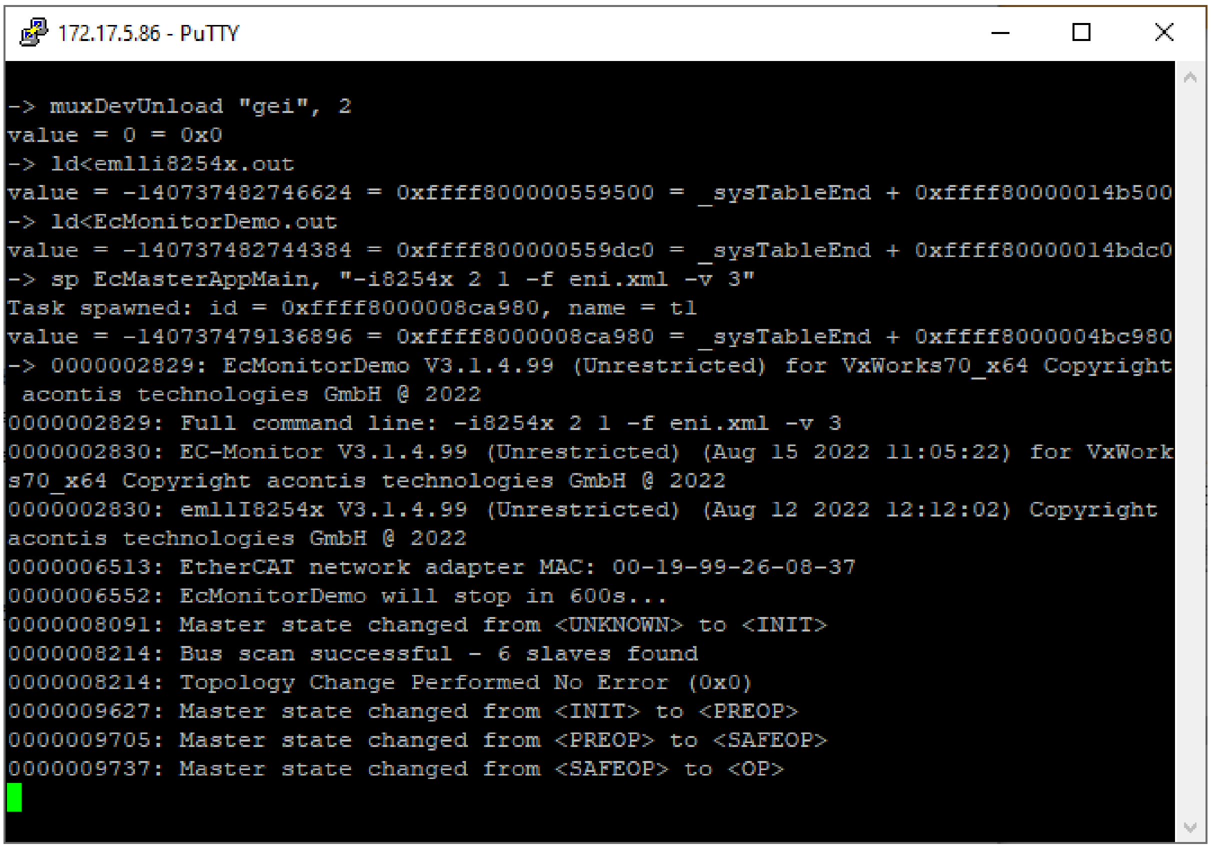
8. **Run the example application**

   The downloadable module `EcMonitorDemo.out` has to be executed. The configuration file `eni.xml` will be used and thus has to be accessible in the current working directory. The appropriate Link Layer 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<EcMonitorDemo.out
-> sp EcMasterAppMain,"-i8254x 1 1 -f eni.xml"
```

   Example:



```
-> muxDevUnload "gei", 2
value = 0 = 0x0
-> ld<emlli8254x.out
value = -140737482746624 = 0xffff800000559500 = _sysTableEnd + 0xffff80000014b500
-> ld<EcMonitorDemo.out
value = -140737482744384 = 0xffff800000559dc0 = _sysTableEnd + 0xffff80000014bdc0
-> sp EcMasterAppMain, "-i8254x 2 1 -f eni.xml -v 3"
Task spawned: id = 0xffff8000008ca980, name = t1
value = -140737479136896 = 0xffff8000008ca980 = _sysTableEnd + 0xffff8000004bc980
-> 0000002829: EcMonitorDemo V3.1.4.99 (Unrestricted) for VxWorks70_x64 Copyright
 acontis technologies GmbH @ 2022
0000002829: Full command line: -i8254x 2 1 -f eni.xml -v 3
0000002830: EC-Monitor V3.1.4.99 (Unrestricted) (Aug 15 2022 11:05:22) for VxWork
s70_x64 Copyright acontis technologies GmbH @ 2022
0000002830: emllI8254x V3.1.4.99 (Unrestricted) (Aug 12 2022 12:12:02) Copyright
acontis technologies GmbH @ 2022
0000006513: EtherCAT network adapter MAC: 00-19-99-26-08-37
0000006552: EcMonitorDemo will stop in 600s...
0000008091: Master state changed from <UNKNOWN> to <INIT>
0000008214: Bus scan successful - 6 slaves found
0000008214: Topology Change Performed No Error (0x0)
0000009627: Master state changed from <INIT> to <PREOP>
0000009705: Master state changed from <PREOP> to <SAFEOP>
0000009737: Master state changed from <SAFEOP> to <OP>
```

See also:

Running EcMonitorDemo

6.3.4 OS Compiler settings

Besides the general settings from *Compiling the EcMonitorDemo* 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

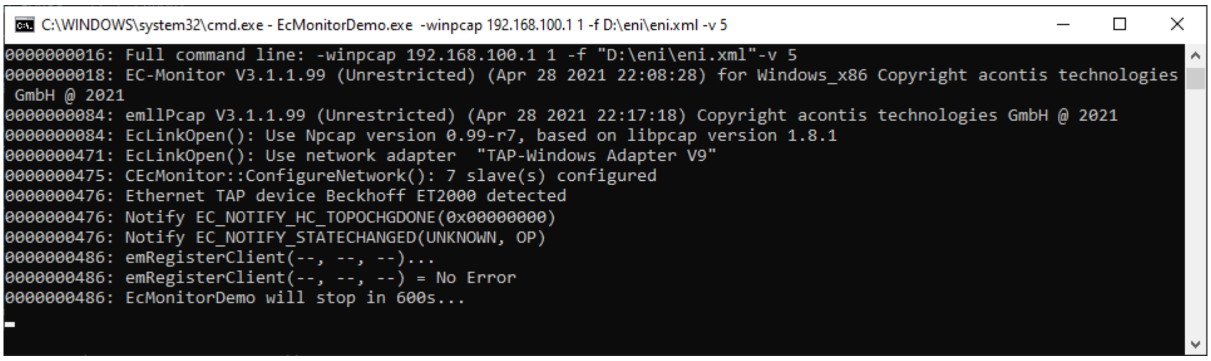
```
<InstallPath>/SDK/LIB/VxWorks/<ARCH>
```

6.4 Microsoft Windows

6.4.1 EcMonitorDemo

1. **Install EC-Monitor**
   Run `setup.exe` from EC-Monitor package, which will guide you through the installation process.
2. **Determine the network interface**
   For example the option *-winpcap 192.168.1.1 1* will be using the network adapter card with the IP address 192.168.1.1.
3. Insert a TAP device after the Master Controller to capture the EtherCAT traffic and start the EtherCAT master
4. **Run the example application**
   Execute `<InstallPath>/Bin/Windows/<Arch>/EcMonitorDemo.exe`. At least a Link Layer option has to be given.

   ```
   C:
   > EcMonitorDemo -winpcap 192.168.100.1 1 -f eni.xml
   ```



```
C:\WINDOWS\system32\cmd.exe - EcMonitorDemo.exe  -winpcap 192.168.100.1 1 -f D:\eni\eni.xml -v 5
0000000016: Full command line: -winpcap 192.168.100.1 1 -f "D:\eni\eni.xml"-v 5
0000000018: EC-Monitor V3.1.1.99 (Unrestricted) (Apr 28 2021 22:08:28) for Windows_x86 Copyright acontis technologies
 GmbH @ 2021
0000000084: emllPcap V3.1.1.99 (Unrestricted) (Apr 28 2021 22:17:18) Copyright acontis technologies GmbH @ 2021
0000000084: EcLinkOpen(): Use Npcap version 0.99-r7, based on libpcap version 1.8.1
0000000471: EcLinkOpen(): Use network adapter  "TAP-Windows Adapter V9"
0000000475: CEcMonitor::ConfigureNetwork(): 7 slave(s) configured
0000000476: Ethernet TAP device Beckhoff ET2000 detected
0000000476: Notify EC_NOTIFY_HC_TOPOCHGDONE(0x00000000)
0000000476: Notify EC_NOTIFY_STATECHANGED(UNKNOWN, OP)
0000000486: emRegisterClient(--, --, --)...
0000000486: emRegisterClient(--, --, --) = No Error
0000000486: EcMonitorDemo will stop in 600s...
```

See also:

Running EcMonitorDemo for a detailed description of the demo application.

6.4.2 OS Compiler settings

Besides the general settings from *Compiling the EcMonitorDemo* the following settings are necessary to build the example application for Windows:

Library path:

- `<InstallPath>/SDK/LIB/Windows/<Arch>`

Include path:

- `<InstallPath>/SDK/INC/Windows`

7 Link Layer

7.1 Link Layer selection

The EC-Monitor currently supports a variety of different Link Layer modules, each of which contained in a single library file, which is loaded by the core library dynamically. The EC-Monitor shipment consist of a core library and one (or more) libraries each containing support for one specific Link Layer module (type of hardware card). Which library actually is loaded, is depending on the Link Layer parameters at runtime.

The principle of Link Layer selection is that the name of the Link Layer (Link Layer Identification) is used to determine the location and name of a registration function, which is called by the EC-Monitor and registers function pointers which allow access to the Link Layer functional entries.

The EtherCAT Link Layer will be initialized using a Link Layer specific configuration parameter set. A pointer to this parameter set is part of the EC-Monitor initialization settings.

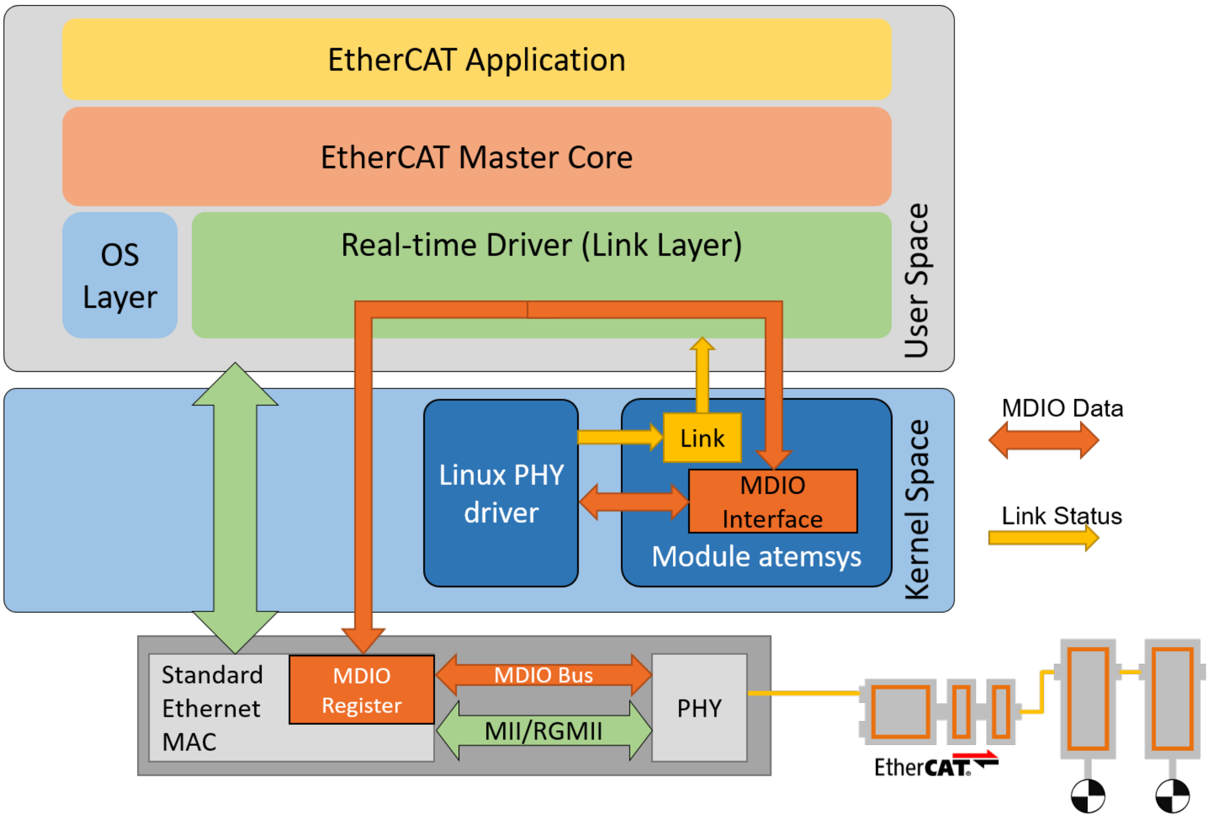
The EC-Monitor supports two Link Layer operating modes. If the Link Layer operates in interrupt mode all received Ethernet frames will be processed immediately in the context of the Link Layer receiver task. When using the polling mode the EC-Monitor will call the Link Layer receiver polling function prior to processing received frames.

7.1.1 Optimized Link Layer drivers

Optimized means operating directly on the network device’s register set instead of using the operating system’s native driver.

7.1.2 Optimized Link Layer drivers and PHY OS Driver

Some operating systems, e.g. Linux and Xenomai, provide drivers for most common Ethernet controllers and their related physical transceivers (PHY). The manufacturer specific PHY circuits can be handled by a dedicated driver. Using the PHY OS Driver interface it is possible to use the manufacturer’s dedicated PHY driver without modification of the acontis optimized Link Layer driver. Depending on the hardware architecture, an additional module from acontis, e.g. atemsys for Linux, grants access to the MDIO bus to the OS drivers, or request MDIO operations from the OS drivers.



Note: Link Layer modules not listed here may be available if purchased additionally

7.1.3 Link Layer selection and initialization

The different Link Layer modules are selected and parameterized by a common structure *EC_T_LINK_PARMS* shared by all Link Layers and a Link Layer specific structure, pointed to by an element within the common structure. This parameter set is given to `EC_T_INIT_MONITOR_PARMS::pLinkParms` with the call of *emInitMonitor()*.

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)

struct **EC_T_LINK_TTS**

Public Members

EC_T_BOOL **bEnabled**
 [in] Use Time-Triggered Send

EC_T_DWORD **dwCycleTimeUsec**
 [in] Cycle time between 2 pfnStartCycle calls

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

EC_T_LINK_TTS_CALLBACK **pfnStartCycle**
 [in] Callback function called cyclically according dwCycleTimeUsec

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

7.1.4 Link Layer instance selection via PCI location

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

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

```
$ lspci | grep Ethernet
```

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

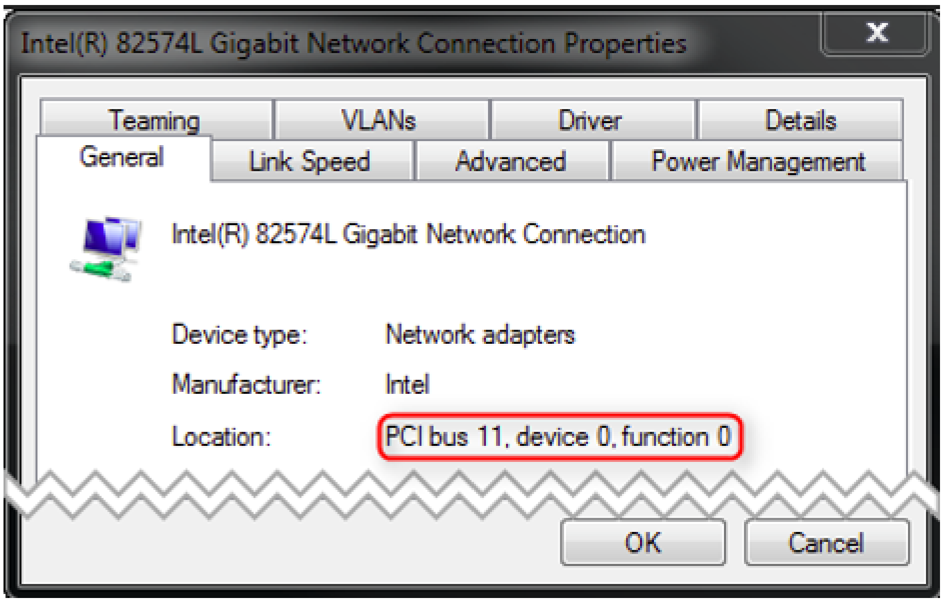
The format of *EC_T_LINK_PARMS::dwInstance* using PCI bus address is:

0x01bbddff

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

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

On Windows the integer value displayed in 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*).



Intel(R) 82574L Gigabit Network Connection

Device type: Network adapters

Manufacturer: Intel

Location: PCI bus 11, device 0, function 0

7.2 Windows NDIS - emllNdis

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

The acontis ECAT Protocol Driver is needed to use the NDIS Link Layer 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 NDIS Link Layer uses the IP address to identify the network adapter.

The parameters to the NDIS Link Layer are setup-specific. The function `CreateLinkParmsFromCmdLineNDIS()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Link Layer 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 Link layer, 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 entry named DontSetPromiscuousMode
- Change the value of DontSetPromiscuousMode to 1
- Close the registry editor and restart your computer

7.3 Windows WinPcap - emllPcap

A Link Layer based on the WinPcap library is shipped with the EC-Monitor. This Link Layer is implemented using a network filter driver that enables the software to send and receive raw Ethernet frames. Using this Link Layer 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 Link Layer 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 WinPcap Link Layer are setup-specific. The function `CreateLinkParmsFromCmdLineWinPcap()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Link Layer 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

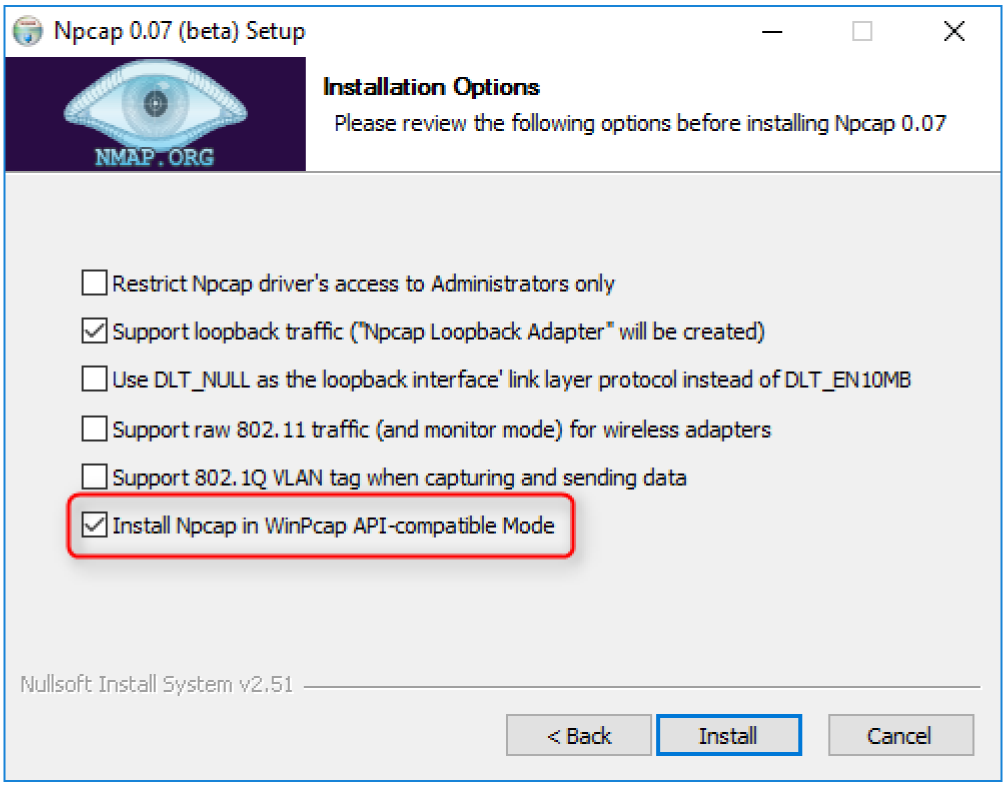
EC_T_BYTE **abyIpAddress**[4]
IP address

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

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



Npcap 0.07 (beta) Setup

Installation Options

Please review the following options before installing Npcap 0.07

- [ ] Restrict Npcap driver's access to Administrators only
- [✓] Support loopback traffic ("Npcap Loopback Adapter" will be created)
- [ ] Use DLT_NULL as the loopback interface' link layer protocol instead of DLT_EN10MB
- [ ] Support raw 802.11 traffic (and monitor mode) for wireless adapters
- [ ] Support 802.1Q VLAN tag when capturing and sending data
- [✓] Install Npcap in WinPcap API-compatible Mode

8 Application programming interface, reference

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

Fundamental types

typedef void ***EC_T_PVOID**
    Pointer of type void

typedef int **EC_T_BOOL**
    Boolean

typedef char **EC_T_CHAR**
    Character, 8 bit

typedef unsigned short **EC_T_WCHAR**
    Wide-character, 16 bit

typedef unsigned char **EC_T_BYTE**
    Byte, unsigned integer 8 bit

typedef unsigned char ***EC_T_PBYTE**
    Pointer of type EC_T_BYTE

typedef unsigned short **EC_T_WORD**
    Word, unsigned integer 16 bit

typedef unsigned int **EC_T_DWORD**
    Double word, unsigned integer 32 bit

typedef signed char **EC_T_SBYTE**
    Signed-Byte, signed integer 8 bit

typedef signed short **EC_T_SWORD**
    Signed-Word, signed integer 16 bit

typedef signed int **EC_T_SDWORD**
    Signed-Double-Word, signed integer 32 bit

typedef int **EC_T_INT**
    Integer

typedef unsigned int **EC_T_UINT**
    Unsigned-Integer

typedef short **EC_T_SHORT**
    Short

typedef unsigned short **EC_T_USHORT**
    Unsigned-Short

typedef float **EC_T_REAL**
    Real, floating point

typedef double **`EC_T_LREAL`**
long Real, floating point

typedef unsigned long long **`EC_T_UINT64`**
Unsigned integer 64 bits

typedef signed long long **`EC_T_INT64`**
Signed integer 64 bits

`EC_T_VOID`
Void type

8.1 General functions

8.1.1 emInitMonitor

EC_T_DWORD **`emInitMonitor`** (
EC_T_DWORD dwInstanceId,
EC_T_MONITOR_INIT_PARMS *pParms
)

Initialize EC-Monitor.

Parameters

- **`dwInstanceId`** – [in] Instance ID
- **`pParms`** – [in] Monitor initialization parameters

Returns
EC_E_NOERROR or error code

struct **`EC_T_MONITOR_INIT_PARMS`**

Public Members

EC_T_DWORD **`dwSignature`**
[in] Set to MONITOR_SIGNATURE

EC_T_DWORD **`dwSize`**
[in] Set to sizeof(EC_T_MONITOR_INIT_PARMS)

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

EC_T_OS_PARMS ***`pOsParms`**
[in] Operation system layer parameters

EC_T_LINK_PARMS ***`pLinkParms`**
[in] Link layer parameters

EC_T_ETHERNET_TAP_TYPE **`eEthTapType`**
[in] Type of Ethernet TAP

EC_T_DWORD **dwBusCycleTimeUsec**
[in] Bus cycle time [usec]

EC_T_DWORD **dwMaxBusSlaves**
[in] Maximum pre-allocated bus slave objects

EC_T_DWORD **dwBacktraceFrames**
[in] Number of frames held in backtrace buffer. Total memory requirements of the buffer: $2 \times dwBacktraceFrames \times 1536 bytes$

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

EC_T_WORKER_THREAD_PARMS **WorkerThreadParms**
[in] Internal worker thread parameters

EC_T_DWORD **dwCommunicationTimeoutMsec**
[in] Timeout [msec] for communication on the Ethernet TAP. 0: defaults to 3 sec, EC_WAITINFINITE: disables monitoring

EC_T_BOOL **bApiLockByApp**
[in] EC_TRUE: Don't lock pending API calls to increase performance

EC_T_CHAR **szFileStoragePath**[EC_FILESTORAGE_PATH_SIZE]
[in] Path used to store records and files, e.g. FoE transfers. EC_NULL: defaults to ""

EC_T_MBX_PARMS **MbxParms**
[in] Mailbox monitoring parameters

enum **EC_T_ETHERNET_TAP_TYPE**
Values:

enumerator **eEthTap_Unknown**
Unknown type

enumerator **eEthTap_AutoDetect**
Auto detect TAP type. If no suitable type is detected, eEthTap_Generic is used

enumerator **eEthTap_Generic**
Generic Ethernet switch

enumerator **eEthTap_Beckhoff_ET2000**
Beckhoff ET2000 Ethernet probe

enumerator **eEthTap_Kunbus_TapCurious**
Kunbus TAP Curious Ethernet probe

enumerator **eEthTap_Hilscher_netANALYZER**
Hilscher netANALYZER Ethernet probe

enumerator **eEthTap_Dummy**

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

EC_PF_HW_TIMER_GET_INPUT_FREQUENCY **`pfHwTimerGetInputFrequency`**
[in] Function to get input frequency of HW timer. This function is needed by some DCM modes described in the Class A manual

EC_PF_HW_TIMER_MODIFY_INITIAL_COUNT **`pfHwTimerModifyInitialCount`**
[in] Function to modify initial count of HW timer. This function is needed by some DCM modes described in the Class A manual

EC_PF_HW_TIMER_GET_CURRENT_COUNT **`pfHwTimerGetCurrentCount`**
[in] Function to get current count of HW timer. This function is needed by some DCM modes described in the Class A manual

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

group **`EC_LOG_LEVELS`**

Defines

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

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]` – Context pointer

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

struct **`EC_T_WORKER_THREAD_PARMS`**

Public Members

EC_T_DWORD **`dwPrio`**
[in] Priority to use for the worker thread

EC_T_CPUSET **`cpuAffinityMask`**
[in] CPU affinity to use for the worker thread

struct **`EC_T_MBX_PARMS`**

Public Members

EC_T_DWORD **`dwMemoryPoolSize`**
[in] Memory for each slave supporting mailbox communication to record e.g. the CoE dictionary. The memory is asynchronously increased by dwBufferSize by the WorkerThread when it is over 80% full. 0: defaults to 1kb

struct **`_EC_T_MBX_PARMS_COE`**

Public Members

EC_T_BOOL **`bDisableNotifications`**
[in] Disable all CoE related EC_NOTIFY_MBOXRCV notifications

EC_T_BOOL **`bDisableODStorage`**
[in] Disable storage of CoE objects in the internal object dictionary

struct **`_EC_T_MBX_PARMS_FOE`**

Public Members

EC_T_BOOL **`bDisableNotifications`**
[in] Disable all FoE related EC_NOTIFY_MBOXRCV notifications

EC_T_BOOL **`bDisableFileStorage`**
[in] Disable storage of FoE transfers as a file on the file system

EC_T_DWORD **`dwMaxQueuedMbxTransfers`**
[in] Maximum number of queued single FoE mailbox transfers that be used as a file write buffer. 0: defaults to 32

8.1.2 emDeinitMonitor

EC_T_DWORD **`emDeinitMonitor`** (*EC_T_DWORD* dwInstanceId)
Deinitialize EC-Monitor.

Parameters
`dwInstanceId` – [in] Instance ID

Returns
EC_E_NOERROR or error code

8.1.3 emConfigureNetwork

EC_T_DWORD **`emConfigureNetwork`** (
EC_T_DWORD dwInstanceID,
EC_T_CNF_TYPE eCnfType,
EC_T_PBYTE pbyCnfData,
EC_T_DWORD dwCnfDataLen
)
Configure EtherCAT network.

Parameters

- **`dwInstanceID`** – [in] Instance ID
- **`eCnfType`** – [in] Type of configuration data provided
- **`pbyCnfData`** – [in] Configuration data
- **`dwCnfDataLen`** – [in] Length of configuration data in byte

Returns
EC_E_NOERROR or error code

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

8.1.4 emGetMonitorStatus

EC_T_DWORD **emGetMonitorStatus** (
EC_T_DWORD dwInstanceID,
EC_T_MONITOR_STATUS *pStatus
)

Get current Monitor status.

Information about the current status of the EtherCAT frame / cycle processing

Parameters

- **dwInstanceID** – [in] Instance ID
- **pStatus** – [out] Monitor status descriptor

Returns

- *EC_E_NOERROR* on success
- *EC_E_INVALIDSTATE* if Monitor isn't initialized
- *EC_E_INVALIDPARM* if pStatus invalid

struct **EC_T_MONITOR_STATUS**

Public Members

EC_T_BOOL **bNextFramesReceived**
[out] Indicates whether further unprocessed frames form the next EtherCAT cycle were received

EC_T_DWORD **dwCyclesProcessed**
[out] Number of EtherCAT cycles processed

EC_T_WORD **wEthTapPositionAutoIncAddr**
[out] Ethernet tap position as auto increment address

EC_T_BOOL **bNextCyclicEntryReceived**
[out] Indicates whether all frames from the next EtherCAT cycle have been received and have not yet been processed

8.1.5 emSetLicenseKey

EC_T_DWORD **emSetLicenseKey** (
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_INVALIDPARM* 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

Example

```
dwRes = emSetLicenseKey(dwInstanceId, "DA1099F2-15C249E9-54327FBC");
if (dwRes != EC_E_NOERROR)
{
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR, "ERROR: 
↪Cannot set license key: %s (0x%lx))\n",
        ecatGetText(dwRes), dwRes));
}
```

See also:

- *emInitMonitor()*
- *emConfigureNetwork()*

8.1.6 emRegisterClient

EC_T_DWORD **emRegisterClient** (
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_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_NOMEMORY* if some memory cannot be allocated

typedef *EC_T_DWORD* (***EC_PF_NOTIFY**)(*EC_T_DWORD* dwCode, *EC_T_NOTIFYPARMS* *pParms)

Parameters

- **dwCode** – [in] Notification code.
- **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)

8.1.7 emUnregisterClient

EC_T_DWORD **emUnregisterClient** (*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

8.1.8 emGetSrcMacAddress

EC_T_DWORD **emGetSrcMacAddress** (
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

See also:

`EC_T_MONITOR_INIT_PARMS::pLinkParms`

8.1.9 emExecJob

EC_T_DWORD **emExecJob** (
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_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_LINK_DISCONNECTED* if the link is disconnected
- *EC_E_FEATURE_DISABLED* for eUsrJob_SwitchEoeFrames if EC_IOCTL_SET_EOE_DEFFERED_SWITCHING_ENABLED hasn't be called before
- *EC_E_ADS_IS_RUNNING* if ADS server is running

Brief job overview:

enum **EC_T_USER_JOB**

Values:

enumerator **eUsrJob_Undefined**

enumerator **eUsrJob_ProcessAllRxFrames**

enumerator **eUsrJob_SendAllCycFrames**

enumerator **eUsrJob_RunMcSm**

enumerator **eUsrJob_MasterTimer**

enumerator **eUsrJob_FlushQueuedCmds**

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***

   When the Link Layer operates in polling mode this call will process all currently received frames, when the Link Layer operates 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_MonitorTimer***

   To trigger the monitor and slave state machines as well as the mailbox handling this call has to be executed cyclically. The monitor cycle time is determined by the period between calling *emExecJob()* (*eUsrJob_MonitorTimer*). The state-machines are handling the EtherCAT state change transfers.

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

8.1.10 emGetMonitorParms

EC_T_DWORD **emGetMonitorParms** (
  EC_T_DWORD dwInstanceID,
  EC_T_MONITOR_INIT_PARMS *pParms,
  EC_T_DWORD dwParmsBufSize
)

Gets current Monitor Init Parameters.

If the given buffer is larger than the actual size of struct *EC_T_MONITOR_INIT_PARMS*, the parameters of *EC_T_MONITOR_INIT_PARMS.pOsParms*, *EC_T_MONITOR_INIT_PARMS.pLinkParms* are appended.

Parameters

- **dwInstanceID** – [in] Instance ID
- **pParms** – [out] Buffer to store parameters
- **dwParmsBufSize** – [in] Size of buffer in bytes

Returns

- *EC_E_NOERROR* on success
- *EC_E_INVALIDSTATE* if Monitor isn't initialized
- *EC_E_INVALIDPARM* if buffer pParms is too small

Example

```
/* Read all monitor init parameters, including OS and Link parameters */
EC_T_BYTE abyBuffer[sizeof(EC_T_MONITOR_INIT_PARMS) + sizeof(EC_T_OS_PARMS) + 512 /
↪* LinkLayer parameters */];
EC_T_MONITOR_INIT_PARMS* pParms = (EC_T_MONITOR_INIT_PARMS*)abyBuffer;
```

(continues on next page)

(continued from previous page)

```
OsMemset(abyBuffer, 0, sizeof(abyBuffer));

dwRes = emGetMonitorParms(dwInstanceId, pParms, sizeof(abyBuffer));
if (EC_E_NOERROR != dwRes)
{
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR, "Cannot get␣
↪monitor parameters: %s (0x%lx))\n",
        ecatGetText(dwRes), dwRes));
}
```

See also:

emInitMonitor()

8.1.11 emSetMonitorParms

EC_T_DWORD **emSetMonitorParms** (
EC_T_DWORD dwInstanceID,
EC_T_MONITOR_INIT_PARMS *pParms
)

Change Monitor Init Parameters.

OS parms, Main Link parms cannot be changed.

Parameters

- **dwInstanceID** – [in] Instance ID
- **pParms** – [in] New Monitor init parameters

Returns

- *EC_E_NOERROR* on success
- *EC_E_INVALIDSTATE* if Monitor isn't initialized

See also:

emInitMonitor()

8.1.12 emGetVersion

EC_T_DWORD **emGetVersion** (*EC_T_DWORD* dwInstanceID, *EC_T_DWORD* *pdwVersion)

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

Returns

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

8.1.13 emGetText

const *EC_T_CHAR* ***emGetText*** (*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

Since the texts are instance independent, a variant without instance id is also available:

const *EC_T_CHAR* ***ecatGetText*** (*EC_T_DWORD* dwTextId)

8.1.14 emGetMemoryUsage

```
EC_T_DWORD emGetMemoryUsage(
    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

8.1.15 emGetMasterState

EC_T_STATE **emGetMasterState** (*EC_T_DWORD* dwInstanceID)

Get the EtherCAT master current state.

Parameters

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

Returns

EtherCAT master state

enum **EC_T_STATE**

Values:

enumerator **eEcatState_UNKNOWN**

enumerator **eEcatState_INIT**

enumerator **eEcatState_PREOP**

enumerator **eEcatState_SAFEOP**

enumerator **eEcatState_OP**

enumerator **eEcatState_BOOTSTRAP**

8.1.16 emGetMasterStateEx

EC_T_DWORD **emGetMasterStateEx** (
    EC_T_DWORD dwInstanceID,
    EC_T_WORD *pwCurrState,
    EC_T_WORD *pwReqState
)

Get the EtherCAT master current and requested state. Possible return values for current and requested state:

- DEVICE_STATE_UNKNOWN
- DEVICE_STATE_INIT
- DEVICE_STATE_PREOP
- DEVICE_STATE_SAFEOP
- DEVICE_STATE_OP

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pwCurrState** – [out] Current master state.
- **pwReqState** – [out] Requested master state

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master isn’t initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the output pointers are EC_NULL

Limitation

Since it is not possible to determine the actual requested master state, the highest slave state of all slaves is assumed to be the requested master state.

8.1.17 emFindInpVarByName - “Inputs.DevicesState”

The device status of all slaves (OR-linked) is part of the process data with name “Inputs.DevicesState”.

EC_T_DWORD emFindInpVarByName(*EC_T_DWORD* dwInstanceID, const *EC_T_CHAR* *szVariableName, *EC_T_PROCESS_VAR_INFO* *pProcessVarInfoEntry)

8.1.18 emFindInpVarByName - "Inputs.BusTime"

The DC system time (written to ESC register 0x0910) is part of the process data with name "Inputs.BusTime".

EC_T_DWORD emFindInpVarByName(*EC_T_DWORD* dwInstanceID, const *EC_T_CHAR* *szVariableName, *EC_T_PROCESS_VAR_INFO* *pProcessVarInfoEntry)

8.1.19 emIoControl

With `emIoControl` a generic control interface exists between the application and the EC-Monitor and its Link Layers.

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

8.1.20 emIoControl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB

This function call registers an callback function which is called after the cyclic frame is received. Typically this is used when the Link Layer 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 `emInitMonitor()` before starting the job task.

emIoControl - 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

Return
EC_E_NOERROR or error code

struct **EC_T_CYCFRAME_RX_CBDESC**

Public Members

EC_T_VOID ***pCallbackContext**
: [in] Context pointer. This pointer is used as parameter every time 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] Context pointer. This pointer is used as parameter every time when the callback function is called.

8.1.21 emIoControl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO

Get cyclic configuration details from ENI configuration file.

emIoControl - 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.

Return
: EC_E_NOERROR or error code

struct **EC_T_CYC_CONFIG_DESC**

Public Members

EC_T_DWORD **dwNumCycEntries**
: [out] Total number of cyclic entries

EC_T_DWORD **dwTaskId**
: [out] Task id of selected cyclic entry

EC_T_DWORD **dwPriority**
: [out] Priority of selected cyclic entry

EC_T_DWORD **dwCycleTime**
: [out] Cycle time of selected cyclic entry

8.1.22 emIoControl - EC_IOCTL_IS_SLAVETOSLAVE_COMM_CONFIGURED

Determine if any slave-to-slave communication is configured.

emIoControl - EC_IOCTL_IS_SLAVETOSLAVE_COMM_CONFIGURED

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 slave-to-slave communication is configured, if EC_FALSE it is not.
- `dwOutBufSize`: [in] Size of the output buffer in bytes.
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Return
EC_E_NOERROR or error code

8.2 Packet Capture

8.2.1 emOpenPacketCapture

EC_T_DWORD **`emOpenPacketCapture`** (
EC_T_DWORD dwInstanceID,
EC_T_PACKETCAPTURE_PARMS *pParms
)

Open packet capture file (PCAP).

Opens a PCAP trace for further processing within the JobTask. No LinkLayer must have been loaded.

Parameters

- **`dwInstanceID`** – [in] Instance ID
- **`pParms`** – [in] Packet capture parameter

Returns

- *EC_E_NOERROR* on success
- *EC_E_INVALIDSTATE* if Monitor isn't initialized or link layer loaded
- *EC_E_INVALIDPARM* if parameter file name invalid
- *EC_E_OPENFAILED* if file could not be opened
- *EC_E_NOMEMORY* if not enough memory available

struct **`EC_T_PACKETCAPTURE_PARMS`**

Public Members

EC_T_CHAR **szFileName**[EC_PACKETCAPTURE_FILE_NAME_SIZE]
[in] File name. Supported formats are *.pcap or *.pcapng

EC_T_BOOL **bReadMultipleFiles**
[in] Read multiple contiguous files. File name format must be "fileName.nnnnn.pcap[ng]", e.g. wireshark.00000.pcap

EC_T_DWORD **dwMaxFrameCnt**
[in] Creates a new file every time the number of frames written exceeds this limit. Disabled with a value set to 0.

EC_T_DWORD **dwMaxFileSize**
[in] Creates a new file every time the number of bytes written exceeds this limit. Disabled with a value set to 0.

EC_T_DWORD **dwRingBufferFileCnt**
[in] Form a ring buffer of the capture files with the given number of files. Only if *EC_T_PACKETCAPTURE_PARMS::dwMaxFrameCnt* or *EC_T_PACKETCAPTURE_PARMS::dwMaxFileSize* are set. Disabled with a value set to 0.

Example

```
EC_T_PACKETCAPTURE_PARMS PacketCaptureParms;
OsMemset(&PacketCaptureParms, dwInstanceId, sizeof(EC_T_PACKETCAPTURE_PARMS));

OsStrcpy(PacketCaptureParms.szFileName, "C:\\ecat.pcap");
dwRes = emOpenPacketCapture(0, &PacketCaptureParms);
if (EC_E_NOERROR != dwRes)
{
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR, "Cannot open␣
↪packet capture: %s (0x%lx))\n",
        ecatGetText(dwRes), dwRes));
}
```

8.2.2 emClosePacketCapture

EC_T_DWORD **emClosePacketCapture**(*EC_T_DWORD* dwInstanceID)
Close packet capture file (PCAP).

Parameters
dwInstanceID – [in] Instance ID

Returns

- *EC_E_NOERROR* on success
- *EC_E_INVALIDSTATE* if Monitor isn't initialized or link layer loaded

8.2.3 emGetPacketCaptureInfo

EC_T_DWORD **emGetPacketCaptureInfo** (
    EC_T_DWORD dwInstanceID,
    EC_T_PACKETCAPTURE_INFO *pInfo
)

Get packet capture file processing status information.

Parameters

- **dwInstanceID** – [in] Instance ID
- **pInfo** – [out] Packet capture info descriptor

Returns

- *EC_E_NOERROR* on success
- *EC_E_INVALIDSTATE* if Monitor isn't initialized or link layer loaded

struct **EC_T_PACKETCAPTURE_INFO**

Public Members

EC_T_PACKETCAPTURE_STATUS **eStatus**
[out] Status of packet capture processing

EC_T_CHAR **szFileName**[EC_PACKETCAPTURE_FILE_NAME_SIZE]
[out] File name of current processed capture

EC_T_UINT64 **qwFrameNumberTotal**
[out] Total number of processed frames from all capture files

EC_T_UINT64 **qwFrameNumberCur**
[out] Last processed frame number from the current packet capture file

EC_T_UINT64 **qwBytesProcessed**
[out] Number of processed bytes from the current packet capture file

EC_T_UINT64 **qwFileSize**
[out] File size[bytes] of the current packet capture

EC_T_UINT64 **qwTimeStamp**
[out] Time stamp[ns] of the last processed frame from the current packet capture file

EC_T_DWORD **dwCyclesProcessed**
[out] Number of EtherCAT cycles processed

enum **EC_T_PACKETCAPTURE_STATUS**
Values:

enumerator **ePcapStatus_Unknown**
Unknown packet capture status

enumerator **`ePcapStatus_NotLoaded`**
No packet capture loaded

enumerator **`ePcapStatus_Running`**
Packet capture processing running

enumerator **`ePcapStatus_Finished`**
Packet capture processing finished

enumerator **`ePcapStatus_Dummy`**

8.2.4 emStartLivePacketCapture

EC_T_DWORD **`emStartLivePacketCapture`** (
EC_T_DWORD dwInstanceID,
EC_T_PACKETCAPTURE_PARMS *pParms
)

Start live packet capture (PCAP).

Starts a live recording of the EtherCAT frames in a specified PCAP file.

Note: Only the PCAP file format is currently supported.

Parameters

- **`dwInstanceID`** – [in] Instance ID
- **`pParms`** – [in] Packet capture parameter

Returns

- *EC_E_NOERROR* on success
- *EC_E_INVALIDPARM* if parameter file name invalid
- *EC_E_OPENFAILED* if file could not be opened
- *EC_E_NOMEMORY* if not enough memory available

8.2.5 emStopLivePacketCapture

EC_T_DWORD **`emStopLivePacketCapture`** (*EC_T_DWORD* dwInstanceID)

Stop live packet capture (PCAP).

Stops a previously started live recording of the EtherCAT frames.

Parameters
`dwInstanceID` – [in] Instance ID

Returns

- *EC_E_NOERROR* on success
- *EC_E_INVALIDSTATE* if Monitor isn't initialized or no recording is in progress

8.2.6 emBacktracePacketCapture

EC_T_DWORD **emBacktracePacketCapture** (
    EC_T_DWORD dwInstanceID,
    EC_T_PACKETCAPTURE_PARMS *pParms
)

Dump packet capture (PCAP) from backtrace buffer.

Writes a backtrace of the received frames in a specified PCAP file. The number of frames in the backtrace buffer is parameterized via *EC_T_MONITOR_INIT_PARMS::dwBacktraceFrames*.

Note: Only the PCAP file format is currently supported.

Parameters

- **dwInstanceID** – [in] Instance ID
- **pParms** – [in] Packet capture parameter

Returns

- *EC_E_NOERROR* on success
- *EC_E_BUSY* if another dump is in progress
- *EC_E_INVALIDSTATE* if backtrace buffer is not initialized
- *EC_E_INVALIDPARM* if parameter file name invalid
- *EC_E_OPENFAILED* if file could not be opened
- *EC_E_NOMEMORY* if not enough memory available

See also:

emInitMonitor()

8.3 Process Data functions

8.3.1 emGetProcessData

EC_T_DWORD **emGetProcessData** (
    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 transfered data
- **`dwDataLen`** – [in] Buffer length [bytes]
- **`dwTimeout`** – [in] Timeout [ms]

Returns
EC_E_NOERROR or error code

8.3.2 emGetProcessDataBits

EC_T_DWORD **`emGetProcessDataBits`** (
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 transfered 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

See also:

`emGetProcessData()`

8.3.3 emGetProcessImageInputPtr

EC_T_BYTE ***emGetProcessImageInputPtr*** (*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

8.3.4 emGetProcessImageOutputPtr

EC_T_BYTE ***emGetProcessImageOutputPtr*** (*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

8.3.5 emFindInpVarByName

EC_T_DWORD **emFindInpVarByName** (
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`

8.3.6 emFindInpVarByNameEx

EC_T_DWORD **emFindInpVarByNameEx** (
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_EX

8.3.7 emFindOutpVarByName

```
EC_T_DWORD emFindOutpVarByName (
    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

8.3.8 emFindOutpVarByNameEx

```
EC_T_DWORD emFindOutpVarByNameEx (
    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

8.3.9 emIoControl - EC_IOCTL_GET_PDMEMORYSIZE

Get the process data image size. This information may be used to provide process data image storage from outside the EC-Monitor core. This IOCTL is to be called after *emConfigureNetwork()*.

emIoControl - 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 (type: EC_T_MEMREQ_DESC).
- `dwOutBufSize`: [in] Size of the output buffer in bytes.
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Return
EC_E_NOERROR or error code

struct **`EC_T_MEMREQ_DESC`**

8.3.10 Process Data access functions

8.3.10.1 EC_COPYBITS

`EC_COPYBITS` (pbyDst, nDstBitOffs, pbySrc, nSrcBitOffs, nBitSize)
Copies a block of bits from a source buffer to a destination buffer.

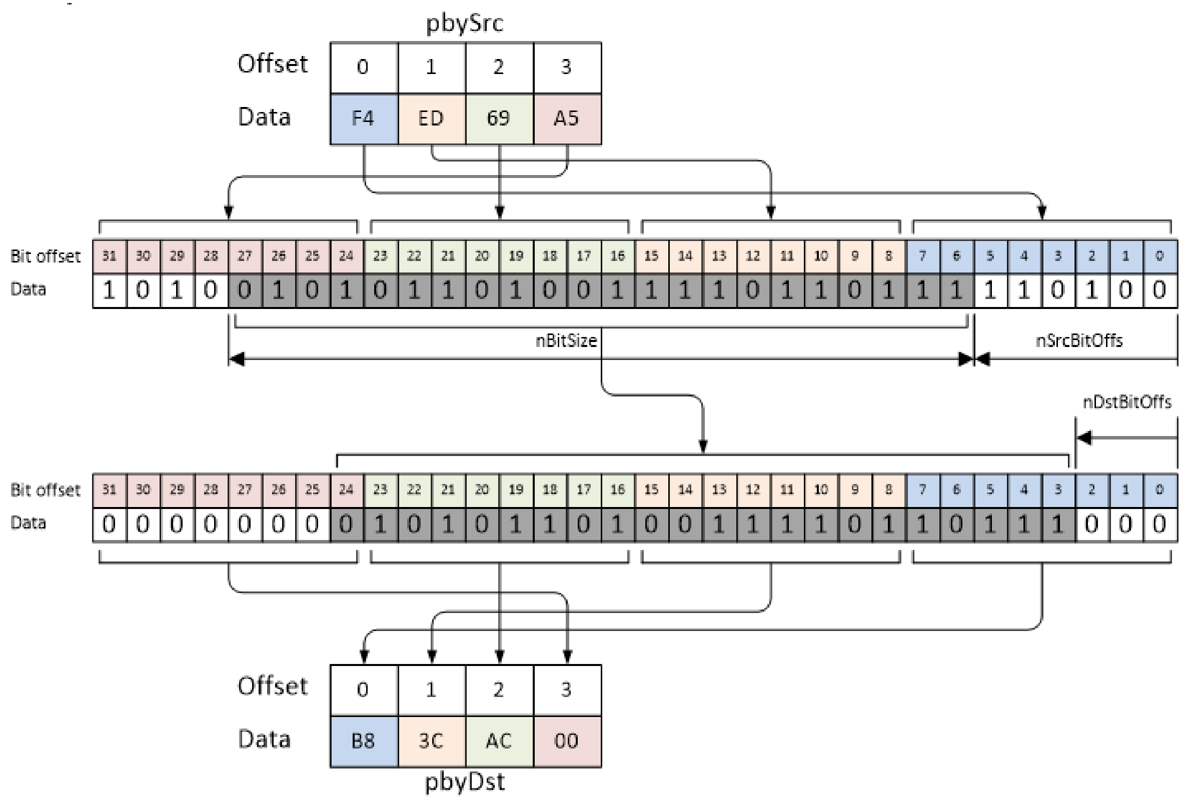
Note: The memory buffers must be allocated before. The buffers must be big enough to hold the block starting at the given offsets! The buffers are not checked for overrun.

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 */
```

8.3.10.2 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 */
```

8.3.10.3 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 */
```

8.3.10.4 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 */
```

8.3.10.5 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*

8.3.11 emIoControl - EC_IOCTL_SET_IGNORE_INPUTS_ON_WKC_ERROR

Set ignore inputs on WKC error

emIoControl - EC_IOCTL_SET_IGNORE_INPUTS_ON_WKC_ERROR

Parameter

- `pbyInBuf`: [in] Pointer to value of EC_T_BOOL. EC_TRUE: inputs are ignored on WKC error.
- `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

Return
EC_E_NOERROR or error code

Calling this IOCTL with `EC_TRUE` as parameter will ignore the inputs data of cyclic commands on WKC error. The default behavior will copy the input data if WKC is non zero and below the expected value. If WKC is not matching the expected value a notification *emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR* is generated and the application must consider this status for the current cycle.

8.3.12 emIoControl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ERROR

Set zero inputs on WKC error

emIoControl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ERROR

Parameter

- `pbyInBuf`: [in] Pointer to value of EC_T_BOOL. EC_TRUE: inputs are set to zero on WKC error.
- `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

Return
EC_E_NOERROR or error code

Calling this IOCTL with `EC_TRUE` as parameter will set the inputs data of cyclic commands to zero on WKC error. The default behavior will copy the input data if WKC is non zero and below the expected value. If WKC is not matching the expected value a notification *emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR* is generated and the application must consider this status for the current cycle.

8.3.13 emIoControl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ZERO

Set zero inputs on WKC is zero

emIoControl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ZERO

Parameter

- `pbyInBuf`: [in] Pointer to value of EC_T_BOOL. EC_TRUE: inputs are set to zero on WKC is zero.
- `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

Return
EC_E_NOERROR or error code

Calling this IOCTL with `EC_TRUE` as parameter will ignore the inputs data of cyclic commands on WKC error. At default behavior it will ignore the input data if WKC is zero, and keep the previous state.

8.4 Slave status functions

8.4.1 emGetNumConfiguredSlaves

EC_T_DWORD **`emGetNumConfiguredSlaves`** (*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

8.4.2 emGetNumConnectedSlaves

EC_T_DWORD **`emGetNumConnectedSlaves`** (*EC_T_DWORD* dwInstanceID)

Get amount of currently connected slaves.

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

Returns
Number of connected slaves

8.4.3 emGetSlaveId

EC_T_DWORD **`emGetSlaveId`** (*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

8.4.4 emGetSlaveIdAtPosition

EC_T_DWORD **`emGetSlaveIdAtPosition`** (
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

8.4.5 emGetSlaveState

EC_T_DWORD **`emGetSlaveState`** (
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_INVALIDPARM* 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

Limitation

Since it is not possible to determine the actual requested slave state from the master, the highest slave state of all slaves is assumed to be the requested state.

See also:

- *`emGetSlaveId()`*
- *emNotify - EC_NOTIFY_SLAVE_STATECHANGED*

8.4.6 emIsSlavePresent

EC_T_DWORD **`emIsSlavePresent`** (
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

See also:

- *`emGetSlaveId()`*
- *emNotify - EC_NOTIFY_SLAVE_PRESENCE*

8.4.7 emGetSlaveProp

EC_T_BOOL **emGetSlaveProp** (
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

struct **EC_T_SLAVE_PROP**

Public Members

EC_T_WORD **wStationAddress**
station address or INVALID_FIXED_ADDR

EC_T_WORD **wAutoIncAddr**
auto increment address or INVALID_AUTO_INC_ADDR

EC_T_CHAR **achName**[MAX_STD_STRLEN]
name of the slave device (NULL terminated string)

See also:

emGetSlaveId()

8.4.8 emGetSlaveInpVarInfoNumOf

EC_T_DWORD **emGetSlaveInpVarInfoNumOf** (
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_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found

See also:

- *emGetSlaveInpVarInfo()*
- *emGetSlaveInpVarInfoEx()*

8.4.9 emGetSlaveInpVarInfo

EC_T_DWORD **emGetSlaveInpVarInfo** (
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

8.4.10 emGetSlaveInpVarInfoEx

```
EC_T_DWORD emGetSlaveInpVarInfoEx(
    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_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found

struct **EC_T_PROCESS_VAR_INFO_EX**

Public Members

EC_T_CHAR **szName**[MAX_PROCESS_VAR_NAME_LEN_EX]
[out] Name of the found process variable

EC_T_WORD **wDataType**
[out] Data type of the found process variable (according to ETG.1000, section 5). See also 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)

MAX_PROCESS_VAR_NAME_LEN_EX
Maximum length of a extended process variable name: 127 characters

8.4.11 emGetSlaveOutpVarInfoNumOf

EC_T_DWORD **emGetSlaveOutpVarInfoNumOf** (
  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

See also:

- *emGetSlaveOutpVarInfo()*
- *emGetSlaveOutpVarInfoEx()*

8.4.12 emGetSlaveOutpVarInfo

EC_T_DWORD **emGetSlaveOutpVarInfo** (
  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

8.4.13 emGetSlaveOutpVarInfoEx

EC_T_DWORD **emGetSlaveOutpVarInfoEx** (
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

8.4.14 emReadSlaveRegister

EC_T_DWORD **emReadSlaveRegister** (
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 that have so far been transfered to a slave and received by the EC-Monitor.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wRegisterOffset** – [in] Register offset. I.e. use 0x0130 to read the AL Status register.
- **pbyData** – [out] Buffer receiving transfered data
- **wLen** – [in] Number of bytes to receive

- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC_NOWAIT
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC_E_TIMEOUT* if dwTimeout elapsed during the API call
- *EC_E_BUSY* 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

8.4.15 emGetCfgSlaveInfo

EC_T_DWORD **emGetCfgSlaveInfo** (
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

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) 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) 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

Flags EC_MBX_PROTOCOL_

`EC_MBX_PROTOCOL_AOE`

`EC_MBX_PROTOCOL_EOE`

`EC_MBX_PROTOCOL_COE`

`EC_MBX_PROTOCOL_FOE`

`EC_MBX_PROTOCOL_SOE`

`EC_MBX_PROTOCOL_VOE`

8.4.16 emGetBusSlaveInfo

EC_T_DWORD **emGetBusSlaveInfo** (
    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_INVALIDPARM* if dwInstanceID is out of range
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found

struct **EC_T_BUS_SLAVE_INFO**

Public Members

EC_T_DWORD **dwSlaveId**
[out] The slave's ID to bind bus slave and config slave information

EC_T_DWORD **adwPortSlaveIds**[ESC_PORT_COUNT]
[out] The slave's ID of the slaves connected to ports. See Port slave ID's

EC_T_WORD **wPortState**

[out] Port link state. Format: wwww xxxx yyyy zzzz (each nibble : port 3210)

wwww : Signal detected 1=yes, 0=no

xxxx : Loop closed 1=yes, 0=no

yyyy : Link established 1=yes, 0=no

zzzz : Slave connected 1=yes, 0=no (zzzz = logical result of w,x,y)

EC_T_WORD **wAutoIncAddress**
[out] The slave's auto increment address

EC_T_BOOL **bDcSupport**
[out] Slave supports DC (Bus Topology Scan)

EC_T_BOOL **bDc64Support**
[out] Slave supports 64 Bit DC (Bus Topology Scan)

EC_T_DWORD **dwVendorId**
[out] Vendor Identification stored in the EEPROM at offset 0x0008

EC_T_DWORD **dwProductCode**
[out] Product Code stored in the EEPROM at offset 0x000A

EC_T_DWORD **dwRevisionNumber**
[out] Revision number stored in the EEPROM at offset 0x000C

EC_T_DWORD **dwSerialNumber**
[out] Serial number stored in the EEPROM at offset 0x000E

EC_T_BYTE **byESCType**
[out] Type of ESC (Value of slave ESC register 0x0000)

EC_T_BYTE **byESCRevision**
[out] Revision number of ESC (Value of slave ESC register 0x0001)

EC_T_WORD **wESCBuild**
[out] Build number of ESC (Value of slave ESC register 0x0002)

EC_T_BYTE **byPortDescriptor**
[out] Port descriptor (Value of slave ESC register 0x0007)

EC_T_WORD **wFeaturesSupported**
[out] Features supported (Value of slave ESC register 0x0008)

EC_T_WORD **wStationAddress**
[out] The slave's station address (Value of slave ESC register 0x0010)

EC_T_WORD **wAliasAddress**
[out] The slave's alias address (Value of slave ESC register 0x0012)

EC_T_WORD **wAlStatus**
[out] AL status (Value of slave ESC register 0x0130)

EC_T_WORD **wAlStatusCode**
[out] AL status code. (Value of slave ESC register 0x0134 during last error acknowledge). This value is reset after a slave state change

EC_T_DWORD **dwSystemTimeDifference**
[out] System time difference. (Value of slave ESC register 0x092C)

EC_T_WORD **wMbxSupportedProtocols**
[out] Supported Mailbox Protocols stored in the EEPROM at offset 0x001C

EC_T_WORD **wDlStatus**
[out] DL status (Value of slave ESC register 0x0110)

EC_T_WORD **wPrevPort**
[out] Connected port of the previous slave

EC_T_WORD **wIdentifyData**
[out] Last read identification value see *EC_T_CFG_SLAVE_INFO.wIdentifyAdo*

EC_T_BOOL **`bLineCrossed`**
: [out] Line crossed was detected at this slave

EC_T_DWORD **`dwSlaveDelay`**
: [out] Delay behind slave [ns]. 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

Port Slave ID's

`MASTER_SLAVE_ID`

`SIMULATOR_SLAVE_ID`

`MASTER_RED_SLAVE_ID`

`EL9010_SLAVE_ID`

`FRAMELOSS_SLAVE_ID`

`JUNCTION_RED_FLAG`

Flags EC_LINECROSSED_

`EC_LINECROSSED_NOT_CONNECTED_PORTA`

`EC_LINECROSSED_UNEXPECTED_INPUT_PORT`

`EC_LINECROSSED_UNEXPECTED_JUNCTION_RED`

`EC_LINECROSSED_UNRESOLVED_PORT_CONNECTION`

`EC_LINECROSSED_HIDDEN_SLAVE_CONNECTED`

`EC_LINECROSSED_PHYSIC_MISMATCH`

`EC_LINECROSSED_INVALID_PORT_CONNECTION`

8.5 Diagnosis

In case of errors on the bus or in one or multiple slaves the EtherCAT monitor stack will notify the application about such an event.

The error notifications can be separated into two classes:

Slave unrelated errors
: Notifications don't contain this information even if one specific slave has caused an error. For example if one or multiple slaves are powered off the working counter of the cyclic commands would be wrong. In that case the *emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR* error notification will be generated.

Slave related errors
: Notifications will also contain the information about which slave has generated an error.

Example Error Scenario

Slave is powered off or disconnected while bus is operational

If the monitor is operational it cyclically sends EtherCAT commands to read and write the slave's process data. It expects the working counter to be incremented to the appropriate value. If one slave is powered off the monitor will generate the *emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR* to indicate such an event. Also the monitor detects a DL status event and performs a bus scan as reaction on this. For the not reachable slaves (powered off or disconnected) the monitor generates the notification *emNotify - EC_NOTIFY_SLAVE_PRESENCE*.

A possible error recovery scenario would be to stay operational and in parallel wait until the slave is powered on again. The next step would be to determine the slave's state and set it operational again:

Monitor calls *emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR*

- Application gets informed
- WKC State in Diagnosis Image changes

Use cases

1. **Slave is disconnected or powered off:**
    - EtherCAT Master detects a DL status event interrupt an performs a bus scan.
    - Monitor calls *emNotify - EC_NOTIFY_SLAVE_PRESENCE*
2. **Slave is re-connected or powered on:**
    - EtherCAT Master detects a DL status event interrupt an performs a bus scan.
    - Monitor calls *emNotify - EC_NOTIFY_SLAVE_PRESENCE*.
    - Application could wait until all slaves are re-connected by calling the functions `emGetNumConnectedSlaves()` and `emGetNumConfiguredSlaves()`.

8.5.1 emIoControl - EC_IOCTL_SB_STATUS_GET

This call will get the status of the last bus scan.

emIoControl - EC_IOCTL_SB_STATUS_GET

Parameter

- `pbyInBuf`: [in] Should be set to EC_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to EC_T_SB_STATUS_NTFY_DESC.
- `dwOutBufSize`: [in] Size of the output buffer in bytes.
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Return
EC_E_NOERROR or error code

See also:

emNotify - EC_NOTIFY_SB_STATUS

8.5.2 emIoControl - EC_IOCTL_GET_SLVSTATISTICS

Get Slave's statistics counter. Counters are collected on a regularly base (default: off) and show errors on Ethernet Layer.

emIoControl - EC_IOCTL_GET_SLVSTATISTICS

Parameter

- `pbyInBuf`: [in] Pointer to a EC_T_DWORD type variable containing the slave id.
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Pointer to struct EC_T_SLVSTATISTICS_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.

Return
EC_E_NOERROR or error code

struct **EC_T_SLVSTATISTICS_DESC**

Public Members

EC_T_BYTE **abyInvalidFrameCnt**[ESC_PORT_COUNT]
[out] Invalid Frame Counters per Slave Port

EC_T_BYTE **abyRxErrorCnt**[ESC_PORT_COUNT]
[out] RX Error Counters per Slave Port

EC_T_BYTE **abyFwdRxErrorCnt**[ESC_PORT_COUNT]
[out] Forwarded RX Error Counters per Slave Port

EC_T_BYTE **byProcessingUnitErrorCnt**
[out] Processing Unit Error Counter

EC_T_BYTE **byPdiErrorCnt**
[out] PDI Error Counter

EC_T_WORD **wAlStatusCode**
[out] AL Status Code

EC_T_BYTE **abyLostLinkCnt**[ESC_PORT_COUNT]
[out] Lost Link Counters per Slave Port

EC_T_UINT64 **qwReadTime**
[out] Timestamp of the last read [ns]

EC_T_UINT64 **qwChangeTime**
[out] Timestamp of the last counter change [ns]

8.5.3 emGetSlaveStatistics

EC_T_DWORD **`emGetSlaveStatistics`** (
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

See also:

- *emIoControl - EC_IOCTL_GET_SLVSTATISTICS*
- `emGetSlaveId()`

8.5.4 emIoControl - EC_IOCTL_CLR_SLVSTATISTICS

Clear all buffered error registers for all slaves. The actual counters on the slaves remain unchanged.

emIoControl - EC_IOCTL_CLR_SLVSTATISTICS

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

Return
EC_E_NOERROR or error code

8.5.5 emClearSlaveStatistics

EC_T_DWORD **`emClearSlaveStatistics`** (
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

Note: Only the buffered error register values are deleted. The actual counters on the slaves remain unchanged.

See also:

emGetSlaveId()

8.5.6 emGetDiagnosisImagePtr

EC_T_BYTE ***emGetDiagnosisImagePtr**(*EC_T_DWORD* dwInstanceID)

Gets the diagnosis image pointer.

Parameters

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

Returns

Diagnosis image pointer

See also:

- *EC_T_CFG_SLAVE_INFO::wWkcStateDiagOffsIn*
- *EC_T_CFG_SLAVE_INFO::wWkcStateDiagOffsOut*

8.5.7 emGetMasterSyncUnitInfoNumOf

EC_T_DWORD **emGetMasterSyncUnitInfoNumOf**(*EC_T_DWORD* dwInstanceID)

Get number of Master Sync Units info entries.

Parameters

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

Returns

Number of Master Sync Units info entries

8.5.8 emGetMasterSyncUnitInfo

EC_T_DWORD **emGetMasterSyncUnitInfo**(
EC_T_DWORD dwInstanceID,
EC_T_WORD wMsuId,
EC_T_MSU_INFO *pMsuInfo
)

Get information about specific Master Sync Unit.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **wMsuId** – [in] Master Sync Unit to get the information from
- **pMsuInfo** – [out] Pointer to an *EC_T_MSU_INFO* structure receiving the Master Sync Unit information

Returns

EC_E_NOERROR or error code

`MSU_ID_ALL_INFO_ENTRIES` retrieves the information from all master sync units at once. The application must ensure that pMsuInfo is capable for all entries.

struct **EC_T_MSU_INFO**

Public Members

EC_T_WORD **wMsuId**
[out] master sync unit ID

EC_T_DWORD **dwBitOffsIn**
[out] input bit offset of master sync unit in process data image

EC_T_DWORD **dwBitSizeIn**
[out] input bit size of master sync unit

EC_T_DWORD **dwBitOffsOut**
[out] output bit offset of master sync unit in process data image

EC_T_DWORD **dwBitSizeOut**
[out] output bit size of master sync unit

EC_T_WORD **wWkcStateDiagOffsIn**
[out] Offset of WkcState bit in diagnosis image WkcState bit values: 0 = Data Valid, 1 = Data invalid

EC_T_WORD **wWkcStateDiagOffsOut**
[out] Offset of WkcState bit in diagnosis image WkcState bit values: 0 = Data Valid, 1 = Data invalid

EC_T_DWORD **adwReserved**[16]
reserved

See also:

emGetMasterSyncUnitInfoNumOf()

8.6 Link Layer Control Interface

8.6.1 emIoControl - EC_IOCTL_ISLINK_CONNECTED

emIoControl - EC_IOCTL_ISLINK_CONNECTED

Parameter

- `pbyInBuf`: [in] Should be set to EC_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to buffer of type struct EC_T_LINK_CONNECTED_INFO
- `dwOutBufSize`: [in] Size of the output buffer in bytes, sizeof(EC_T_LINK_CONNECTED_INFO)
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer

Return
EC_E_NOERROR or error code

struct **EC_T_LINK_CONNECTED_INFO**

Public Members

EC_T_BOOL **bConnected**
[out] MAIN or RED link detected

EC_T_BOOL **bSendEnabled**
[out] send enabled on MAIN or RED

EC_T_BOOL **bMainConnected**
[out] MAIN link detected

EC_T_BOOL **bMainMasked**
[out] MAIN link not used for sending, because topology changed delay not elapsed yet

EC_T_BOOL **bRedConnected**
[out] RED link detected

EC_T_BOOL **bRedMasked**
[out] RED link not used for sending, because topology changed delay not elapsed yet

8.6.2 emIoControl - EC_IOCTL_GET_LINKLAYER_MODE

emIoControl - EC_IOCTL_GET_LINKLAYER_MODE

Parameter

- `pbyInBuf`: [in] Should be set to EC_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to buffer of type struct EC_T_LINKLAYER_MODE_DESC
- `dwOutBufSize`: [in] Size of the output buffer in bytes, sizeof(EC_T_LINKLAYER_MODE_DESC)
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer

Return
EC_E_NOERROR or error code

struct **EC_T_LINKLAYER_MODE_DESC**

Public Members

EC_T_LINKMODE **eLinkMode**
[out] Operation mode of main interface

EC_T_LINKMODE **eLinkModeRed**
[out] Operation mode of redundancy interface

8.6.3 emIoControl - EC_LINKIOCTL...

The generic control interface provides access to the main network adapter when adding `EC_IOCTL_LINKLAYER_MAIN` to the `EC_LINKIOCTL` parameter at dwCode.

```
EC_T_DWORD dwCode = (EC_IOCTL_LINKLAYER_MAIN | EC_LINKIOCTL_GET_ETHERNET_ADDRESS);
```

8.6.4 emIoControl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS

Provides MAC addresses of main or red line.

emIoControl - 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 in bytes (at least 6).
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Return
EC_E_NOERROR or error code

8.6.5 emIoControl - EC_LINKIOCTL_GET_SPEED

emIoControl - 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 Link Layer driver to 10/100/1000.
- `dwOutBufSize`: [in] Size of the output buffer in bytes.
- `pdwNumOutData`: [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Return
EC_E_NOERROR or error code

8.7 EtherCAT Mailbox Transfer

To be able to initiate a mailbox transfer the client has to create a mailbox transfer object first. This mailbox transfer object also contains the memory where the data to be transferred is stored. The one client that initiated the mailbox transfer will be notified about a mailbox transfer completion by the `emNotify()` callback function.

To be able to identify the transfer which was completed the client has to assign a unique transfer identifier for each mailbox transfer. The mailbox transfer object can only be used for one single mailbox transfer. If multiple transfers shall be initiated in parallel the client has to create one transfer object for each. The transfer object can be re-used after mailbox transfer completion.

Typical mailbox transfer sequence:

1. Record a mailbox transfer.
2. **Create a transfer object (for example a SDO download transfer object).**

```
MbxTferDesc.dwMaxDataLen = 10

MbxTferDesc.pbyMbxTferDescData = (EC_T_PBYTE)OsMalloc(MbxTferDesc.
↪dwMaxDataLen)

pMbxTfer = emMbxTferCreate(&MbxTferDesc)
    state of the transfer object = Idle
```

3. **Set the location to write the transferred data to, determine the transfer ID, store the client ID in the object and initiate the transfer (e.g. a SDO upload). A transfer may only be initiated if the state of the transfer object is Idle.**

```
pMbxTfer->dwDataLen = MbxTferDesc.dwMaxDataLen;

pMbxTfer->pbyMbxTferData = MbxTferDesc.pbyMbxTferDescData

pMbxTfer->dwTferId = 1;

pMbxTfer->dwClntId = dwClntId;


dwResult = emCoeSdoUplodadReq(pMbxTfer, dwSlaveId, wObIndex, ...);
    state of the transfer object = Pend or TferReqError
```

The state will then be set to Pend to indicate that this mailbox transfer object currently is in use and the transfer is not completed. If the mailbox transfer cannot be initiated the master will set the object into the state TferReqError - in such cases the client is responsible to set the state back into Idle.

4. **If the mailbox transfer is completed the notification callback function of the corresponding client (`emNotify()`) will be called with a pointer to the mailbox transfer object. The state of the transfer object is set to TferDone prior to calling `emNotify()`.**

```
if( dwResult != EC_E_NOERROR ) { ... }

emNotify( EC_NOTIFY_MBOXRCV, pParms )
    state of the transfer object = TferDone
```

5. **In case of errors the appropriate error handling has to be executed. Application must set the transfer object state to Idle.**

```
if( pMbxTfer->dwErrorCode != EC_E_NOERROR ) { ... }
    In emNotify: application may set transfer object state to Idle
```

6. **Delete the transfer object. Alternatively this object can be used for the next transfer.**

```
emMbxTferDelete(pMbxTfer);
```

8.7.1 Mailbox transfer object states

The following states exist for a mailbox transfer object:

enum **EC_T_MBXTFER_STATUS**
Values:

enumerator **eMbxTferStatus_Idle**
Mailbox transfer object not in use

enumerator **eMbxTferStatus_Pend**
Mailbox transfer in process

enumerator **eMbxTferStatus_TferDone**
Mailbox transfer completed

enumerator **eMbxTferStatus_TferReqError**
Mailbox transfer request error

enumerator **eMbxTferStatus_TferWaitingForContinue**
Mailbox transfer waiting for continue, object owned by application

A mailbox transfer will be processed by the monitor independently from the client's timeout setting. Some types of mailbox transfers can be cancelled by the client, e.g. if the client's timeout elapsed.

After completion of the mailbox transfer (with timeout and the client may finally set the transfer object into the state *EC_T_MBXTFER_STATUS::eMbxTferStatus_Idle*. New mailbox transfers can only be requested if the object is in the state *EC_T_MBXTFER_STATUS::eMbxTferStatus_Idle*.

8.7.2 emMbxTferCreate

EC_T_MBXTFER ***emMbxTferCreate** (
EC_T_DWORD dwInstanceID,
EC_T_MBXTFER_DESC *pMbxTferDesc
)

Creates a mailbox transfer object.

While a mailbox transfer is in process the related transfer object and the corresponding memory may not be accessed. After a mailbox transfer completion the object may be used for the next transfer. The mailbox transfer object has to be deleted by calling ecatMbxTferDelete if it is not needed any more.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTferDesc** – [in] Pointer to the mailbox transfer descriptor. Determines details of the mailbox transfer.

Returns

- Pointer to the created mailbox transfer object if successful
- EC_NULL on error (No memory left)

struct **EC_T_MBXTFER_DESC**

Public Members

EC_T_DWORD **dwMaxDataLen**
Maximum amount of data bytes that shall be transferred using this object. A mailbox transfer type without data transfer will ignore this parameter

EC_T_BYTE ***pbyMbxTferDescData**
Pointer to byte stream carrying in and out data of mailbox content

struct **EC_T_MBXTFER**

Public Members

EC_T_DWORD **dwClntId**
[] Client ID

EC_T_MBXTFER_DESC **MbxTferDesc**
[out] Mailbox transfer descriptor. All elements of pMbxTferDesc will be stored here

EC_T_MBXTFER_TYPE **eMbxTferType**
[] This type information is written to the Mailbox Transfer Object by the last call to a mailbox command function. It may be used as an information, and is required to fan out consecutive notifications. This value is only valid until next mailbox relevant API call, where this value may be overwritten

EC_T_DWORD **dwDataLen**
[] Amount of data bytes for the next mailbox transfer. If the mailbox transfer does not transfer data from or to the slave this parameter will be ignored. This element has to be set to an appropriate value every time prior to initiate a new request. When the transfer is completed (emNotify) this value will contain the amount of data that was actually transferred

EC_T_BYTE ***pbyMbxTferData**
[in/out] Pointer to data. In case of a download transfer the client has to store the data in this location. In case of an upload transfer this element points to the received data. Access to data that was uploaded from a slave is only valid within the notification function because the buffer will be re-used by the master "this data has to be copied into a separate buffer in case it has to be used later by the client

EC_T_MBXTFER_STATUS **eTferStatus**
[out] Transfer state. After a new transfer object is created the state will be set to eMbxTferStatus_Idle

EC_T_DWORD **dwErrorCode**
[out] Error code of a mailbox transfer that was terminated with error

EC_T_DWORD **dwTferId**
[] Transfer ID. For every new mailbox transfer a unique ID has to be assigned. This ID can be used after mailbox transfer completion to identify the transfer

EC_T_MBX_DATA **MbxData**
[] Mailbox data. This element contains mailbox transfer data, e.g. the CoE object dictionary list.

enum **EC_T_MBXTFER_TYPE**
Values:

enumerator **eMbxTferType_COE_SDO_DOWNLOAD**
CoE SDO download

enumerator **eMbxTferType_COE_SDO_UPLOAD**
CoE SDO upload

enumerator **eMbxTferType_COE_GETODLIST**
CoE Get object dictionary list

enumerator **eMbxTferType_COE_GETOBDESC**
CoE Get object description

enumerator **eMbxTferType_COE_GETENTRYDESC**
CoE Get object entry description

enumerator **eMbxTferType_COE_EMERGENCY**
CoE emergency request

enumerator **eMbxTferType_COE_RX_PDO**
CoE RxPDO

enumerator **eMbxTferType_FOE_FILE_UPLOAD**
FoE upload

enumerator **eMbxTferType_FOE_FILE_DOWNLOAD**
FoE download

enumerator **eMbxTferType_SOE_READREQUEST**
SoE read request

enumerator **eMbxTferType_SOE_READRESPONSE**
SoE read response

enumerator **eMbxTferType_SOE_WRITEREQUEST**
SoE write request

enumerator **eMbxTferType_SOE_WRITERESPONSE**
SoE write response

enumerator **eMbxTferType_SOE_NOTIFICATION**
SoE notification

enumerator **eMbxTferType_SOE_EMERGENCY**
SoE emergency

enumerator **eMbxTferType_VOE_MBX_READ**
VoE read

enumerator **eMbxTferType_VOE_MBX_WRITE**
VoE write

enumerator **eMbxTferType_AOE_READ**
AoE read

enumerator **eMbxTferType_AOE_WRITE**
AoE write

enumerator **`eMbxTferType_AOE_READWRITE`**
AoE read/write

enumerator **`eMbxTferType_AOE_WRITECONTROL`**
AoE write control

enumerator **`eMbxTferType_RAWMBX`**
Raw mbx

enumerator **`eMbxTferType_FOE_SEG_DOWNLOAD`**
FoE segmented download

enumerator **`eMbxTferType_FOE_SEG_UPLOAD`**
FoE segmented upload

enumerator **`eMbxTferType_S2SMBX`**
S2S mbx

enumerator **`eMbxTferType_FOE_UPLOAD_REQ`**
FoE upload request

enumerator **`eMbxTferType_FOE_DOWNLOAD_REQ`**
FoE download request

union **`EC_T_MBX_DATA`**

Public Members

EC_T_AOE_CMD_RESPONSE **`AoE_Response`**
AoE

EC_T_MBX_DATA_COE **`CoE`**
CoE

EC_T_COE_ODLIST **`CoE_ODList`**
CoE Object Dictionary list

EC_T_COE_OBDESC **`CoE_ObDesc`**
CoE object description

EC_T_COE_ENTRYDESC **`CoE_EntryDesc`**
CoE entry description

EC_T_COE_EMERGENCY **`CoE_Emergency`**
CoE emergency data

EC_T_MBX_DATA_COE_INITCMD **`CoE_InitCmd`**
CoE InitCmd

EC_T_MBX_DATA_FOE **`FoE`**
FoE

EC_T_MBX_DATA_FOE_REQ **`FoE_Request`**
FoE request

EC_T_MBX_DATA_SOE **`SoE`**
SoE

EC_T_SOE_NOTIFICATION **`SoE_Notification`**
SoE notification request

EC_T_SOE_EMERGENCY **`SoE_Emergency`**
SoE emergency request

8.7.3 emMbxTferAbort

EC_T_DWORD **`emMbxTferAbort`** (*EC_T_DWORD* dwInstanceID, *EC_T_MBXTFER* *pMbxTfer)
Abort a running mailbox transfer.

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 created with emMbxTferCreate

Returns
EC_E_NOERROR if successful

Currently only supported for FoE Transfer, CoE Download and CoE Upload.

8.7.4 emMbxTferDelete

EC_T_VOID **`emMbxTferDelete`** (*EC_T_DWORD* dwInstanceID, *EC_T_MBXTFER* *pMbxTfer)
Deletes a mailbox transfer object.

A transfer object may only be deleted if it is in the Idle state.

Parameters

- **`dwInstanceID`** – [in] Instance ID (Multiple EtherCAT Network Support)
- **`pMbxTfer`** – [in] Mailbox transfer object created with emMbxTferCreate

Returns
EC_E_NOERROR or error code

8.7.5 emNotify - EC_NOTIFY_MBOXRCV

Indicates a mailbox transfer completion.

emNotify - EC_NOTIFY_MBOXRCV

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER, contains the corresponding mailbox transfer object.

- `dwInBufSize`: [in] Size of the transfer object 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

The element `EC_T_MBXTFER::dwClntId` contains the corresponding ID of the client that is notified, the corresponding transfer ID can be found in `EC_T_MBXTFER::dwTferId`. The transfer result is stored in `EC_T_MBXTFER::dwErrorCode`.

On error `EC_T_MBXTFER::eTferStatus` is `eMbxTferStatus_TferReqError`, on success `eMbxTferStatus_TferDone`. In order to reuse the transfer object the application must set it back to `eMbxTferStatus_Idle`.

The `EC_T_MBXTFER::eMbxTferType` element determines the mailbox transfer type (e.g. `eMbxTferType_COE_SDO_DOWNLOAD` for a completion of a CoE SDO download transfer).

8.8 CAN application protocol over EtherCAT (CoE)

The EC-Monitor can forward CoE transfers to the application in real time via the notifications *emNotify - eMbxTferType_COE_SDO_DOWNLOAD*, *emNotify - eMbxTferType_COE_SDO_UPLOAD* and *emNotify - eMbxTferType_COE_EMERGENCY*.

There is also the option of storing the recorded data from the CoE transfers in an internal object dictionary. This object dictionary is structured analogously to that from the slaves and can be read out via the functions `emCoeSdoUpload()` / `emCoeSdoUploadReq()` and `emCoeGetODList()`.

The notifications for CoE can be deactivated using the `EC_T_MBX_PARMS::_EC_T_MBX_PARMS_COE::bDisableNotifica` parameter if they are not required or to save computing time. In order to reduce memory consumption, the internal memory for the CoE data can be deactivated using the `EC_T_MBX_PARMS::_EC_T_MBX_PARMS_COE::bDisableODStorage` parameter.

If both parameters `EC_T_MBX_PARMS::_EC_T_MBX_PARMS_COE::bDisableNotifications` and `EC_T_MBX_PARMS::_EC_T_MBX_PARMS_COE::bDisableODStorage` are set, the CoE monitoring is completely deactivated.

8.8.1 emNotify - eMbxTferType_COE_SDO_DOWNLOAD

SDO download transfer completion.

emNotify - eMbxTferType_COE_SDO_DOWNLOAD

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.
- `dwInBufSize`: [in] Size of the transfer object 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

The transfer result is stored in `EC_T_MBXTFER::dwErrorCode`. The request parameters stored in element `EC_T_MBX_DATA::CoE` of type `EC_T_MBX_DATA_COE` are part of `EC_T_MBXTFER::MbxData`. The SDO data stored in `EC_T_MBXTFER::pbyMbxTferData`.

struct **EC_T_MBX_DATA_COE**

Public Members

EC_T_WORD **wStationAddress**
Station address of the slave

EC_T_WORD **wIndex**
Object index

EC_T_BYTE **bySubIndex**
Object subindex

EC_T_BOOL **bCompleteAccess**
Complete access

8.8.2 emNotify - eMbxTferType_COE_SDO_UPLOAD

SDO upload transfer completion.

emNotify - eMbxTferType_COE_SDO_UPLOAD

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER, contains the corresponding mailbox transfer object.
- `dwInBufSize`: [in] Size of the transfer object 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

The transfer result is stored in *`EC_T_MBXTFER::dwErrorCode`*. The request parameters stored in element *`EC_T_MBX_DATA::CoE`* of type *`EC_T_MBX_DATA_COE`* are part of *`EC_T_MBXTFER::MbxData`*. The SDO data stored in *`EC_T_MBXTFER::pbyMbxTferData`*.

8.8.3 CoE Emergency (emNotify - eMbxTferType_COE_EMERGENCY)

Indication of a CoE emergency request. A *emNotify - EC_NOTIFY_MBOXRCV* is given with *`EC_T_MBXTFER::eMbxTferType`* = *`EC_T_MBXTFER_TYPE::eMbxTferType_COE_EMERGENCY`*.

emNotify - eMbxTferType_COE_EMERGENCY

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER, contains the corresponding mailbox transfer object.
- `dwInBufSize`: [in] Size of the transfer object 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

In case of an emergency notification all registered clients will get this notification. The corresponding mailbox transfer object will be created. *`EC_T_MBXTFER::dwTferId`* is undefined as it is not needed by the client. The transfer result is stored in *`EC_T_MBXTFER::dwErrorCode`*.

The emergency data stored in element *`EC_T_MBX_DATA::CoE_Emergency`* of type *`EC_T_COE_EMERGENCY`* is part of *`EC_T_MBXTFER::MbxData`* and may have to be buffered by the client. Access to the memory area *`EC_T_MBXTFER::MbxData`* outside of the notification caller context is illegal and the results are undefined.

struct **`EC_T_COE_EMERGENCY`**

Public Members

EC_T_WORD **`wErrorCode`**
Error code according to EtherCAT specification

EC_T_BYTE **`byErrorRegister`**
Error register

EC_T_BYTE **`abyData`**[EC_COE_EMERGENCY_DATASIZE]
Error data

EC_T_WORD **`wStationAddress`**
Slave node address of the faulty slave

See also:

A more detailed description of the values can be found in the EtherCAT specification ETG.1000, section 5.

8.8.4 emCoeSdoUpload

EC_T_DWORD **`emCoeSdoUpload`**(
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 transfered 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

Limitation

- Only CoE entries which have been received by the EcMonitor can be retrieved
- CoE objects read via complete access, can only be retrieved as complete access.
- When switching between complete access and access via subindexes the corresponding CoE object is overwritten.

See also:

emGetSlaveId()

8.8.5 emCoeSdoUploadReq

EC_T_DWORD **emCoeSdoUploadReq** (
EC_T_DWORD dwInstanceID,
EC_T_MBXTFER *pMbxTfer,
EC_T_DWORD dwSlaveId,
EC_T_WORD wObIndex,
EC_T_BYTE byObSubIndex,
EC_T_DWORD dwTimeout,
EC_T_DWORD dwFlags
)

Initiates a CoE SDO upload from an EtherCAT slave device to the master and returns immediately.

The length of the data to be uploaded must be set in *EC_T_MBXTFER.dwDataLen*. A unique transfer ID must be written into *EC_T_MBXTFER.dwTferId*. EC_NOTIFY_MBOXRCV is given on completion.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object created with emMbxTferCreate
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **byObSubIndex** – [in] Object sub index. 0 or 1 if Complete Access.
- **dwTimeout** – [in] Timeout [ms]
- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Returns

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

- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave if full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running

Limitation

- Only CoE entries which have been received by the EC-Monitor can be retrieved
- CoE objects read via complete access, can only be retrieved as complete access.
- When switching between complete access and access via subindexes, the corresponding CoE object is overwritten.

See also:

- *emNotify - eMbxTferType_COE_SDO_UPLOAD*
- `emGetSlaveId()`

8.8.6 emCoeGetODList

EC_T_DWORD **`emCoeGetODList`** (
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 have so far been transfered to a slave and received by the EC-Monitor. 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

enum **`EC_T_COE_ODLIST_TYPE`**

Values:

enumerator **eODListType_Lengths**
Lengths of each list type

enumerator **eODListType_ALL**
List contains all objects

enumerator **eODListType_RxPdoMap**
List with PDO mappable objects

enumerator **eODListType_TxPdoMap**
List with objects that can be changed

enumerator **eODListType_StoredFRepl**
Only stored for a device replacement objects

enumerator **eODListType_StartupParm**
Only startup parameter objects

See also:

- *emMbxTferCreate()*
- *emGetSlaveId()*

8.8.7 emNotify - eMbxTferType_COE_GETODLIST

Notification of a detected CoE SDO information service transfer for a object dictionary list.

emNotify - eMbxTferType_COE_GETODLIST

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER.
- `dwInBufSize`: [in] Size of the transfer object 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

The transfer result is stored in *EC_T_MBXTFER::dwErrorCode*.

The object list stored in element *EC_T_MBX_DATA::CoE_ODList* of type *EC_T_COE_ODLIST* is part of *EC_T_MBXTFER::MbxData* and may have to be buffered by the client. Access to the memory area *EC_T_MBXTFER::MbxData* outside of the notification caller context is illegal and the results are undefined.

struct **EC_T_COE_ODLIST**

Public Members

EC_T_COE_ODLIST_TYPE **eOdListType**
list type

EC_T_WORD **wLen**
amount of object IDs

EC_T_WORD **wStationAddress**
Station address of the slave

EC_T_WORD ***pwOdList**
array containing object IDs

8.8.8 emNotify - eMbxTferType_COE_GETENTRYDESC

Notification of a detected CoE SDO information service transfer for a object entry description.

emNotify - eMbxTferType_COE_GETENTRYDESC

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER.
- `dwInBufSize`: [in] Size of the transfer object 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

The transfer result is stored in *`EC_T_MBXTFER::dwErrorCode`*.

The object entry description stored in element *`EC_T_MBX_DATA::CoE_EntryDesc`* of type *`EC_T_COE_ENTRYDESC`* is part of *`EC_T_MBXTFER::MbxData`* and may have to be buffered by the client. Access to the memory area *`EC_T_MBXTFER::MbxData`* outside of the notification caller context is illegal and the results are undefined.

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)

Value info flags

group **EC_COE_ENTRY_VALUEINFO**
EtherCat CoE entry description value information bit definitions.

Defines

EC_COE_ENTRY_ObjAccess
Object access

EC_COE_ENTRY_ObjCategory
Object category

EC_COE_ENTRY_PdoMapping
PDO mapping

EC_COE_ENTRY_UnitType
Unit type

EC_COE_ENTRY_DefaultValue
Default value

EC_COE_ENTRY_MinValue
Minimum value

`EC_COE_ENTRY_MaxValue`
Maximum value

Object access flags

group **`EC_COE_ENTRY_OBJACCESS`**
EtherCat CoE entry access bit definitions.

Defines

`EC_COE_ENTRY_Access_R_PREOP`
Read access in Pre-Operational state

`EC_COE_ENTRY_Access_R_SAFEOP`
Read access in Safe-Operational state

`EC_COE_ENTRY_Access_R_OP`
Read access in Operational state

`EC_COE_ENTRY_Access_W_PREOP`
Write access in Pre-Operational state

`EC_COE_ENTRY_Access_W_SAFEOP`
Write access in Safe-Operational state

`EC_COE_ENTRY_Access_W_OP`
Write access in Operational state

See also:

A more detailed description of the values can be found in the EtherCAT specification ETG.1000, section 5 and 6.

8.9 File access over EtherCAT (FoE)

The EC-Monitor can record file transfers via the FoE protocol between an EtherCAT master and a slave. These FoE transfers can be forwarded to the application as segmented packets in real time via the notifications *emNotify - eMbxTferType_FOE_SEG_DOWNLOAD* and *emNotify - eMbxTferType_FOE_SEG_UPLOAD*.

In addition, the FoE transfers can be stored as a file on the file system. The files are automatically created and stored in *`EC_T_MONITOR_INIT_PARMS::szFileStoragePath`*. The file name consists of the following:

```
<TimeStamp[msec]>_Slave<StationAddress>_<FoeFileName>
```

For example:

```
0123456789_Slave1001_firmware.bin
```

The notifications for FoE can be deactivated using the *`EC_T_MBX_PARMS::_EC_T_MBX_PARMS_FOE::bDisableNotifica`* parameter if they are not required or to save computing time. If no file system is available or file storage is not desired, it can be disabled using the *`EC_T_MBX_PARMS::_EC_T_MBX_PARMS_FOE::bDisableFileStorage`* parameter.

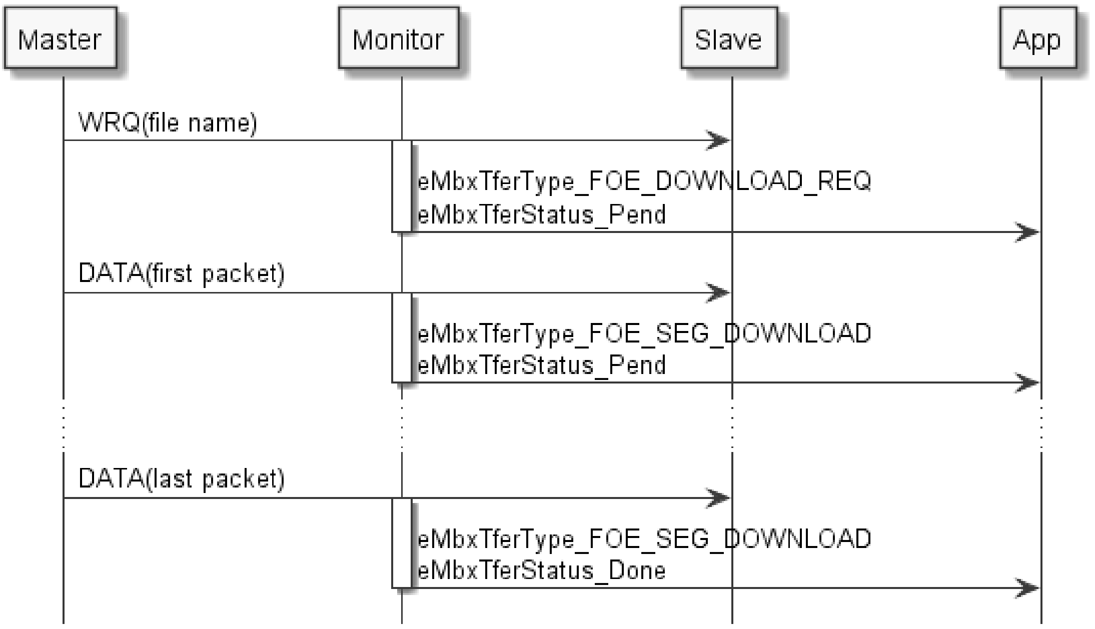
If both parameters `EC_T_MBX_PARMS::_EC_T_MBX_PARMS_FOE::bDisableNotifications` and `EC_T_MBX_PARMS::_EC_T_MBX_PARMS_FOE::bDisableFileStorage` are set, the FoE monitoring is completely deactivated.

8.9.1 Notification sequence

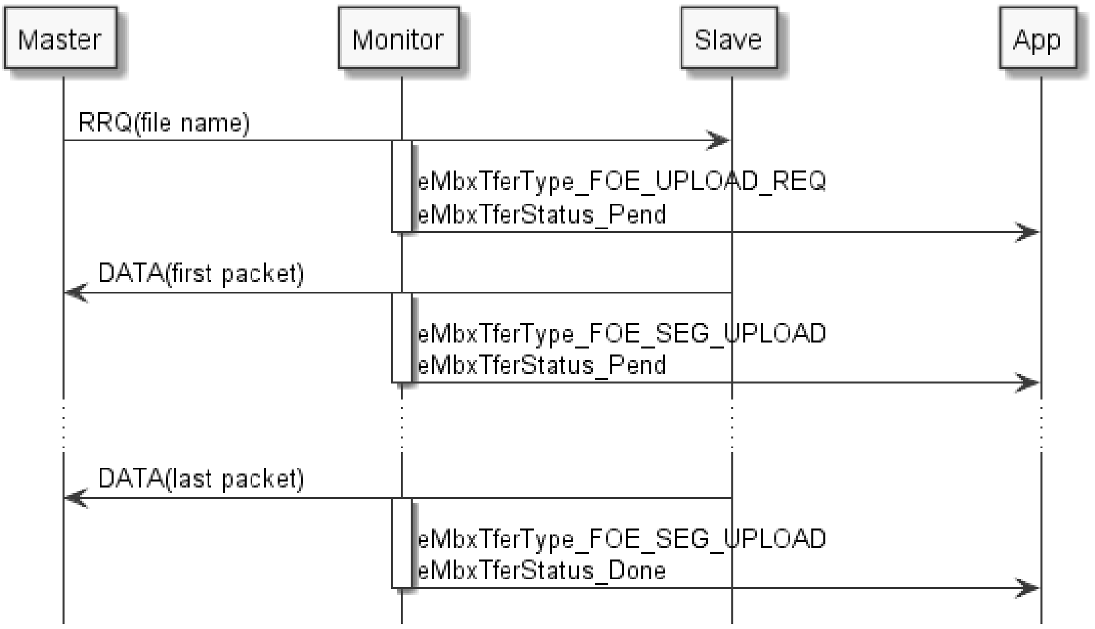
Once the EC-Monitor detects an FoE transfer, the application is notified via an *emNotify - eMbxTferType_FOE_DOWNLOAD_REQ* or *emNotify - eMbxTferType_FOE_UPLOAD_REQ* notification. This notification contains basic information about the upcoming transfer, e.g. requested file name.

After that, each individual packet is transmitted via an *emNotify - eMbxTferType_FOE_SEG_DOWNLOAD* or *emNotify - eMbxTferType_FOE_SEG_UPLOAD* notification. The end of the transfer is set via the `EC_T_MBXTFER::eTferStatus` = `eMbxTferStatus_TferDone`.

Download



Upload



8.9.2 emNotify - eMbxTferType_FOE_DOWNLOAD_REQ

Notifies a FoE download request from the EtherCAT master to a slave.

emNotify - eMbxTferType_FOE_DOWNLOAD_REQ

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.
- `dwInBufSize`: [in] Size of the transfer object 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

The parameters that the master has requested from the slave are stored in the structure *`EC_T_MBX_DATA::FoE_Request`* which is part of *`EC_T_MBXTFER::MbxData`*.

struct **`EC_T_MBX_DATA_FOE_REQ`**

Public Members

EC_T_WORD **`wStationAddress`**
[out] Station address of the slave

EC_T_DWORD **`dwPassword`**
[out] FoE read/write request password

EC_T_CHAR **`szFileName`**[EC_MAX_FILE_NAME_SIZE]
[out] Name of the file to be read/write

8.9.3 emNotify - eMbxTferType_FOE_SEG_DOWNLOAD

Transmits a data segment of the ongoing FoE download.

emNotify - eMbxTferType_FOE_SEG_DOWNLOAD

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.
- `dwInBufSize`: [in] Size of the transfer object 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

The FoE download data segment is stored at *`EC_T_MBXTFER::pbyMbxTferData`* with size *`EC_T_MBXTFER::dwDataLen`* and may have to be buffered by the application. Access to the memory area *`EC_T_MBXTFER::pbyMbxTferData`* outside of the notification caller context is illegal and the results are undefined.

Information about the current transfer are in structure `EC_T_MBX_DATA::FoE` which is part of `EC_T_MBXTFER::MbxData`. Among other things, it contains the slave station address `EC_T_MBX_DATA_FOE::wStationAddress` and the number of bytes already transmitted `EC_T_MBX_DATA_FOE::dwTransferredBytes`.

struct **`EC_T_MBX_DATA_FOE`**

Public Members

EC_T_DWORD **`dwTransferredBytes`**
[out] amount of transferred bytes

EC_T_DWORD **`dwRequestedBytes`**
[out] amount of bytes to be provided by application

EC_T_DWORD **`dwBusyDone`**
[out] If slave is busy: 0 … dwBusyEntire

EC_T_DWORD **`dwBusyEntire`**
[out] If dwBusyEntire > 0: Slave is busy

EC_T_CHAR **`szBusyComment`**[EC_FOE_BUSY_COMMENT_SIZE]
[out] Busy Comment from slave

EC_T_DWORD **`dwFileSize`**
[out] File size

EC_T_WORD **`wStationAddress`**
[out] Station address of the slave

Note: The elements `EC_T_MBX_DATA_FOE::dwRequestedBytes` and `EC_T_MBX_DATA_FOE::dwFileSize` are not used by the EC-Monitor because they are not known at runtime.

8.9.4 emNotify - eMbxTferType_FOE_UPLOAD_REQ

Notifies a FoE upload request from the EtherCAT master to a slave.

emNotify - eMbxTferType_FOE_DOWNLOAD_REQ

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.
- `dwInBufSize`: [in] Size of the transfer object 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

The parameters that the master has requested from the slave are stored in the structure `EC_T_MBX_DATA::FoE_Request` which is part of `EC_T_MBXTFER::MbxData`.

struct *EC_T_MBX_DATA_FOE_REQ*

> *EC_T_WORD wStationAddress*
> *EC_T_DWORD dwPassword*
> *EC_T_CHAR szFileName*[EC_MAX_FILE_NAME_SIZE]

8.9.5 emNotify - eMbxTferType_FOE_SEG_UPLOAD

Transmits a data segment of the ongoing FoE upload.

emNotify - eMbxTferType_FOE_SEG_UPLOAD

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.
- `dwInBufSize`: [in] Size of the transfer object 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

The FoE upload data segment is stored at *`EC_T_MBXTFER::pbyMbxTferData`* with size *`EC_T_MBXTFER::dwDataLen`* and may have to be buffered by the application. Access to the memory area *`EC_T_MBXTFER::pbyMbxTferData`* outside of the notification caller context is illegal and the results are undefined.

Information about the current transfer are in structure *`EC_T_MBX_DATA::FoE`* which is part of *`EC_T_MBXTFER::MbxData`*. Among other things, it contains the slave station address *`EC_T_MBX_DATA_FOE::wStationAddress`* and the number of bytes already transmitted *`EC_T_MBX_DATA_FOE::dwTransferredBytes`*.

struct *EC_T_MBX_DATA_FOE*

> *EC_T_DWORD dwTransferredBytes*
> *EC_T_DWORD dwRequestedBytes*
> *EC_T_DWORD dwBusyDone*
> *EC_T_DWORD dwBusyEntire*
> *EC_T_CHAR szBusyComment*[EC_FOE_BUSY_COMMENT_SIZE]
> *EC_T_DWORD dwFileSize*
> *EC_T_WORD wStationAddress*

Note: The elements *`EC_T_MBX_DATA_FOE::dwRequestedBytes`* and *`EC_T_MBX_DATA_FOE::dwFileSize`* are not used by the EC-Monitor because they are not known at runtime.

8.9.6 emNotify - EC_NOTIFY_FOE_MBSLAVE_ERROR

This error will be indicated in case a FoE mailbox slave send an error message. Detailed error information is stored in structure `EC_T_MBOX_FOE_ABORT_DESC` which is part of `EC_T_ERROR_NOTIFICATION_DESC`.

struct **EC_T_MBOX_FOE_ABORT_DESC**

Public Members

EC_T_SLAVE_PROP **SlaveProp**
Slave properties

EC_T_DWORD **dwErrorCode**
Error code

EC_T_CHAR **achErrorString**[MAX_STD_STRLEN]
FoE error string

8.9.7 emConvertEcErrorToFoeError

EC_T_DWORD **emConvertEcErrorToFoeError** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwErrorCode
)

Convert master error code to FoE error code.

Returns
FoE error code according to ETG1000.6 Table 92 - Error codes of FoE

8.10 Hot Connect

8.10.1 emHCGetNumGroupMembers

EC_T_DWORD **emHCGetNumGroupMembers** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwGroupIndex
)

Get number of slaves belonging to a specific Hot-Connect group.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwGroupIndex** – [in] Index of Hot-Connect group, 0 is the mandatory group

Returns
Number of slaves

8.10.2 emHCGetSlaveIdsOfGroup

EC_T_DWORD **emHCGetSlaveIdsOfGroup** (
EC_T_DWORD dwInstanceID,
EC_T_DWORD dwGroupIndex,
EC_T_DWORD *adwSlaveId,
EC_T_DWORD dwMaxNumSlaveIds
)

Get a list of Slave ID's belonging to a specific Hot-Connect group.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwGroupIndex** – [in] Index of Hot-Connect group, 0 is the mandatory group
- **adwSlaveId** – [out] Preallocated Slave ID list buffer
- **dwMaxNumSlaveIds** – [in] Size of Slave ID list buffer

Returns

EC_E_NOERROR or error code

8.10.3 emNotify - EC_NOTIFY_HC_DETECTADDGROUPS

This notification is raised when HotConnect group detection is finished, after slave addition.

emNotify - EC_NOTIFY_HC_DETECTADDGROUPS

Parameter

- `pbyInBuf`: [in] pointer to notification descriptor EC_T_HC_DETECTALLGROUP_NTFY_DESC
- `dwInBufSize`: [in] sizeof(EC_T_HC_DETECTALLGROUP_NTFY_DESC)
- `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_HC_DETECTALLGROUP_NTFY_DESC**

Public Members

EC_T_DWORD **dwResultCode**
Result of Group detection

EC_T_DWORD **dwGroupCount**
Total number of Groups

EC_T_DWORD **dwGroupsPresent**
Number of connected groups

EC_T_DWORD **dwGroupMask**
Bitmask of first 32 Groups. 1 = present, 0 = absent

EC_T_DWORD **adwGroupMask**[100]
Bitmask of first 3200 Groups.

8.10.4 emNotify - EC_NOTIFY_HC_PROBEALLGROUPS

This notification is raised when HotConnect Group Detection is finished, after Slave Disappearance.

emNotify - EC_NOTIFY_HC_PROBEALLGROUPS

Parameter

- `pbyInBuf`: [in] pointer to notification descriptor EC_T_HC_DETECTALLGROUP_NTFY_DESC
- `dwInBufSize`: [in] sizeof(EC_T_HC_DETECTALLGROUP_NTFY_DESC)
- `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_HC_DETECTALLGROUP_NTFY_DESC*

EC_T_DWORD dwResultCode
EC_T_DWORD dwGroupCount
EC_T_DWORD dwGroupsPresent
EC_T_DWORD dwGroupMask
EC_T_DWORD adwGroupMask[100]

8.10.5 emNotify - EC_NOTIFY_HC_TOPOCHGDONE

This notification is raised when HotConnect has completely processed a topology change.

emNotify - EC_NOTIFY_HC_TOPOCHGDONE

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD (EC_E_NOERROR on success, Error code otherwise)
- `dwInBufSize`: [in] sizeof(EC_T_DWORD)
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

The notification is raised when the slaves reached the current bus state.

9 Generic notification interface

One of the parameters the client has to set when registering with the EC-Monitor is a generic notification callback function (`emNotify()`). This function is called every time an event occurs about which the client needs to be informed.

Within this callback function the client must not call any active EtherCAT functions which finally would lead to send EtherCAT commands (e.g. initiation of mailbox transfers, starting/stopping the master, sending raw commands). In such cases the behavior is undefined. Only EtherCAT functions which are explicitly marked to be callable within `emNotify()` may be called.

This callback function is usually called in the context of the EC-Monitor timer thread or the EtherCAT Link Layer receiver thread. To avoid dead-lock situations the notification callback handler may not use mutex semaphores.

As the whole EtherCAT operation is blocked while calling this function the error handling must not use much CPU time or even call operating system functions that may block. Usually the error handling will be done in a separate application thread.

9.1 Notification callback

typedef *EC_T_DWORD* (**EC_PF_NOTIFY*)(*EC_T_DWORD* dwCode, *EC_T_NOTIFYPARMS* *pParms)

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

9.2 emNotifyApp

By calling this function the generic notification callback function setup by `emRegisterClient()` is called.

EC_T_DWORD **`emNotifyApp`**(
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwCode,
    EC_T_NOTIFYPARMS *pParms
)

Calls the notification callback functions of all registered clients.

Note: EC_E_ERROR and EC_E_INVALIDPARM from registered clients' callback functions are ignored.

Parameters

- **`dwInstanceID`** – [in] Instance ID (Multiple EtherCAT Network Support)
- **`dwCode`** – [in] Application specific notification code. dwCode must be <= EC_NOTIFY_APP_MAX_CODE. The callback functions get "EC_NOTIFY_APP | dwCode" as parameter.
- **`pParms`** – [in] Parameter to all callback functions. Note: Output parameters are not transferred from RAS client to RAS server.

Returns

EC_E_ERROR or first error code different from EC_E_ERROR and EC_E_INVALIDPARM of registered clients' callback functions

The maximum value for dwCode is defined by `EC_NOTIFY_APP_MAX_CODE`

9.3 Enable/Disable notifications

All notifications can be enabled or disabled. By default, all notifications are enabled except for:

EC_NOTIFY_SLAVE_STATECHANGED

EC_NOTIFY_SLAVES_STATECHANGED

EC_NOTIFY_SLAVES_PRESENCE

EC_NOTIFY_REFCLOCK_PRESENCE

EC_NOTIFY_SLAVES_UNEXPECTED_STATE

EC_NOTIFY_SLAVES_ERROR_STATUS

EC_NOTIFY_COE_INIT_CMD

EC_NOTIFY_SLAVE_REGISTER_TRANSFER

9.3.1 emIoControl - EC_IOCTL_SET_NOTIFICATION_ENABLED

Set notification enabled state. With *EC_T_SET_NOTIFICATION_ENABLED_PARMS::dwCode* set to *EC_ALL_NOTIFICATIONS*, all notifications can be changed at once. *EC_T_SET_NOTIFICATION_ENABLED_PARMS::dwEnabled* set to *EC_NOTIFICATION_DEFAULT*, resets to default.

emIoControl - EC_IOCTL_SET_NOTIFICATION_ENABLED

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SET_NOTIFICATION_ENABLED_PARMS.
- `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

Return
EC_E_NOERROR or error code

struct **EC_T_SET_NOTIFICATION_ENABLED_PARMS**

Public Members

EC_T_DWORD **dwClientId**
[in] Client ID, 0: Master

EC_T_DWORD **dwCode**
[in] Notification code or *EC_ALL_NOTIFICATIONS*

EC_T_DWORD **dwEnabled**
[in] Enable, disable or reset to default notification. See EC_NOTIFICATION_ flags

EC_NOTIFICATION_DISABLED
Disable notification

EC_NOTIFICATION_ENABLED
Enable notification

EC_NOTIFICATION_DEFAULT
Reset notification to default

EC_ALL_NOTIFICATIONS
Notification code to change all notifications

9.3.2 emIoControl - EC_IOCTL_GET_NOTIFICATION_ENABLED

Get notification enabled state.

emIoControl - EC_IOCTL_GET_NOTIFICATION_ENABLED

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_GET_NOTIFICATION_ENABLED_PARMS.
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Pointer to EC_T_BOOL to carry out current enable set.
- `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.

Return

EC_E_NOERROR or error code

struct **`EC_T_GET_NOTIFICATION_ENABLED_PARMS`**

Public Members

EC_T_DWORD **`dwClientId`**
[in] Client ID, 0: Master

EC_T_DWORD **`dwCode`**
[in] Notification code

9.4 Status notifications

9.4.1 emNotify - EC_NOTIFY_STATECHANGED

Notification about a change in the master’s operational state.

emNotify - EC_NOTIFY_STATECHANGED

Parameter

- `pbyInBuf`: [in] Pointer to data of type EC_T_STATECHANGE which contains the old and the new master operational state.
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **`EC_T_STATECHANGE`**

Public Members

EC_T_STATE **oldState**
old operational state

EC_T_STATE **newState**
new operational state

9.4.2 emNotify - EC_NOTIFY_SB_STATUS

Scan bus status notification.

emNotify - EC_NOTIFY_SB_STATUS

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SB_STATUS_NTFY_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **EC_T_SB_STATUS_NTFY_DESC**

Public Members

EC_T_DWORD **dwResultCode**
[in] EC_E_NOERROR: success EC_E_NOTREADY: no bus scan executed EC_E_BUSCONFIG_MISMATCH: bus configuration mismatch Result of scanbus

EC_T_DWORD **dwSlaveCount**
[in] number of slaves connected to the bus

9.4.3 emNotify - EC_NOTIFY_SB_MISMATCH

This notification is triggered when the bus scan detects a discrepancy between connected slaves and configuration due to unexpected slaves or missing mandatory slaves. In case of permanent frame loss no slaves can be found although the slaves are connected.

emNotify - EC_NOTIFY_SB_MISMATCH

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SB_MISMATCH_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_SB_MISMATCH_DESC`**

Public Members

EC_T_WORD **`wPrevFixedAddress`**
[in] Previous slave station address

EC_T_WORD **`wPrevPort`**
[in] Previous slave station address

EC_T_WORD **`wPrevAIncAddress`**
[in] Previous slave auto-increment address

EC_T_WORD **`wBusAIncAddress`**
[in] Unexpected slave (bus) auto-inc address

EC_T_DWORD **`dwBusVendorId`**
[in] Unexpected slave (bus) vendor ID

EC_T_DWORD **`dwBusProdCode`**
[in] Unexpected slave (bus) product code

EC_T_DWORD **`dwBusRevisionNo`**
[in] Unexpected slave (bus) revision number

EC_T_DWORD **`dwBusSerialNo`**
[in] Unexpected slave (bus) serial number

EC_T_WORD **`wBusFixedAddress`**
[in] Unexpected slave (bus) station address

EC_T_WORD **`wIdentificationVal`**
[in] last identification value read from slave according to the last used identification method

EC_T_WORD **`wCfgFixedAddress`**
[in] Missing slave (config) station Address

EC_T_WORD **`wCfgAIncAddress`**
[in] Missing slave (config) Auto-Increment Address

EC_T_DWORD **`dwCfgVendorId`**
[in] Missing slave (config) Vendor ID

EC_T_DWORD **`dwCfgProdCode`**
[in] Missing slave (config) Product code

EC_T_DWORD **`dwCfgRevisionNo`**
[in] Missing slave (config) Revision Number

EC_T_DWORD **`dwCfgSerialNo`**
[in] Missing slave (config) Serial Number

EC_T_BOOL **`bIdentValidationError`**
[in] Hot-Connect Identification command sent to slave but failed

EC_T_WORD **`oIdentCmdHdr`**[5]
[in] Last Hot-Connect Identification command header (if bIdentValidationError)

EC_T_DWORD **`dwCmdData`**
[in] First DWORD of Data portion of last identification command

EC_T_DWORD **`dwCmdVMask`**
[in] First DWORD of Validation mask of last identification command

EC_T_DWORD **`dwCmdVData`**
[in] First DWORD of Validation data of last identification command

9.4.4 emNotify - EC_NOTIFY_HC_TOPOCHGDONE

This notification is triggered when topology change has completely processed.

emNotify - EC_NOTIFY_HC_TOPOCHGDONE

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_DWORD (EC_E_NOERROR on success, Error code otherwise)
- `dwInBufSize`: [in] sizeof(EC_T_DWORD).
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

9.4.5 emNotify - EC_NOTIFY_SLAVE_PRESENCE

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

emNotify - EC_NOTIFY_SLAVE_PRESENCE

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SLAVE_PRESENCE_NTFY_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

Disconnecting the slave from the network, powering it off or a bad connection can produce this notification.

struct **`EC_T_SLAVE_PRESENCE_NTFY_DESC`**

Public Members

EC_T_WORD **`wStationAddress`**
Slave station address

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

9.4.6 emNotify - EC_NOTIFY_SLAVE_STATECHANGED

This notification is triggered when a slave has changed its EtherCAT state. This notification is disabled by default.

emNotify - EC_NOTIFY_SLAVE_STATECHANGED

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SLAVE_STATECHANGED_NTFY_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **`EC_T_SLAVE_STATECHANGED_NTFY_DESC`**

Public Members

EC_T_SLAVE_PROP **`SlaveProp`**
Slave properties

EC_T_STATE **`newState`**
New slave state

See also:

emIoControl - EC_IOCTL_SET_NOTIFICATION_ENABLED to enable notification.

9.4.7 emNotify - EC_NOTIFY_SLAVE_REGISTER_TRANSFER

This notification is triggered when a slave register transfer is completed.

To avoid excessive triggering of the notification, registers that are read by the EtherCAT master at regular intervals are not notified. These are the following registers:

AL-Status (0x0130)

RX Error Counter, Forwarded RX Error Counter, ECAT Processing Unit Error Counter, PDI Error Counter, PDI Error Code, Lost Link Counter (0x0300:0x0314)

SII EEPROM Interface (0x0500:0x050F)

Registers above 0x1000

This notification is disabled by default.

emNotify - EC_NOTIFY_SLAVE_REGISTER_TRANSFER

Parameter

- `pbyInBuf`: [in] Pointer to EC_T_SLAVEREGISTER_TRANSFER_NTFY_DESC
- `dwInBufSize`: [in] Size of the input buffer provided at pbyInBuf in bytes.
- `pbyOutBuf`: [out] Should be set to EC_NULL
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to EC_NULL

struct **`EC_T_SLAVEREGISTER_TRANSFER_NTFY_DESC`**

Public Members

EC_T_DWORD **`dwTferId`**
Transfer ID. For every new slave register transfer a unique ID has to be assigned. This ID can be used after completion to identify the transfer

EC_T_DWORD **`dwResult`**
Result of Slave register transfer

EC_T_BOOL **`bRead`**
EC_TRUE: Read register, EC_FALSE: Write register transfer

EC_T_WORD **`wFixedAddr`**
Station address of slave

EC_T_WORD **`wRegisterOffset`**
Register offset

EC_T_WORD **`wLen`**
Length of slave register transfer

EC_T_BYTE ***`pbyData`**
Pointer to the data read

EC_T_WORD **`wWkc`**
Received working counter

See also:

emIoControl - EC_IOCTL_SET_NOTIFICATION_ENABLED to enable notification.

9.5 Error notifications

For each error an error ID (error code) will be defined. This error ID will be used as the notification code when `emNotify()` is called. In addition to this notification code the second parameter given to `emNotify()` contains a pointer to an error notification descriptor of type *`EC_T_ERROR_NOTIFICATION_DESC`*. This error notification descriptor contains detailed information about the error.

struct **`EC_T_ERROR_NOTIFICATION_DESC`**

Public Members

EC_T_DWORD **`dwNotifyErrorCode`**
Error ID (same value as the notification code)

EC_T_CHAR **`achErrorInfo`**[MAX_ERRINFO_STRLEN]
Additional error string (may be empty)

union **`_EC_T_ERROR_NOTIFICATION_PARM`**

Public Members

EC_T_WKCERR_DESC **`WkcErrDesc`**
WKC error descriptor

EC_T_FRAME_RSPERR_DESC **`FrameRspErrDesc`**
Frame response error descriptor

EC_T_INITCMD_ERR_DESC **`InitCmdErrDesc`**
Master/Slave init command error descriptor

EC_T_SLAVE_ERROR_INFO_DESC **`SlaveErrInfoDesc`**
Slave Error Info Descriptor

EC_T_SLAVES_ERROR_DESC **`SlavesErrDesc`**
Slaves Error Descriptor

EC_T_MBOX_SDO_ABORT_DESC **`SdoAbortDesc`**
SDO Abort

EC_T_RED_CHANGE_DESC **`RedChangeDesc`**
Redundancy Descriptor

EC_T_MBOX_FOE_ABORT_DESC **`FoeErrorDesc`**
FoE error code and string

EC_T_MBXRCV_INVALID_DATA_DESC **`MbxRcvInvalidDataDesc`**
Invalid mailbox data received descriptor

EC_T_PDIWATCHDOG_DESC **`PdiWatchdogDesc`**
PDI watchodg expired

EC_T_SLAVE_NOTSUPPORTED_DESC **`SlaveNotSupportedDesc`**
Slave not supported

EC_T_SLAVE_UNEXPECTED_STATE_DESC **`SlaveUnexpectedStateDesc`**
Slave in unexpected state

EC_T_SLAVES_UNEXPECTED_STATE_DESC **`SlavesUnexpectedStateDesc`**
Slaves in unexpected state

EC_T_EEPROM_CHECKSUM_ERROR_DESC **`EEPROMChecksumErrorDesc`**
EEPROM checksum error

EC_T_JUNCTION_RED_CHANGE_DESC **`JunctionRedChangeDesc`**
Junction redundancy change descriptor

EC_T_FRAMELOSS_AFTER_SLAVE_NTFY_DESC **`FramelossAfterSlaveDesc`**
Frameloss after Slave descriptor

EC_T_S2SMBX_ERROR_DESC **`S2SMbxErrorDesc`**
S2S Mailbox Error descriptor

EC_T_BAD_CONNECTION_NTFY_DESC **`BadConnectionDesc`**
Bad connection descriptor

EC_T_COMMUNICATION_TIMEOUT_NTFY_DESC **`CommunicationTimeoutDesc`**
Communication timeout descriptor

EC_T_TAP_LINK_STATUS_NTFY_DESC **`TapLinkStatusDesc`**
Tap link status

If the pointer to this descriptor exists the detailed error information (e.g. information about the slave) is stored in the appropriate structure of a union. These error information structures are described in the following sections.

9.5.1 emNotify - EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL

When processing cyclic frames the EtherCAT master checks whether all slaves are still in OPERATIONAL state. If at least one slave device is not OPERATIONAL this error will be indicated.

9.5.2 emNotify - EC_NOTIFY_ALL_DEVICES_OPERATIONAL

When processing cyclic frames the EtherCAT master checks whether all slaves are still in OPERATIONAL state. This will be notified after *emNotify - EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL* and all the slaves are back in OPERATIONAL state.

9.5.3 emNotify - EC_NOTIFY_CLIENTREGISTRATION_DROPPED

This notification will be indicated if the client registration was dropped because `emConfigureNetwork()` was called by another thread.

```
EC_T_DWORD dwDeinitForConfiguration; /* 0 = terminating Master, 1 = restarting␣
↪Master */
```

9.5.4 emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR

To update the process data some EtherCAT commands will be sent cyclically by the external master. These commands will address one or multiple slaves. These EtherCAT commands contain a working counter which has to be incremented by each slave that is addressed. The working counter will be checked after the EtherCAT command is received by the monitor. If the expected working counter will not match to the working counter of the received command the error *EC_NOTIFY_CYCCMD_WKC_ERROR* will be indicated. The working counter value expected by the monitor is determined by the EtherCAT configuration (XML) file for each cyclic EtherCAT command (section Config/Cyclic/Frame/Cmd/Cnt). Detailed error information are stored in structure *EC_T_WKCERR_DESC* of *EC_T_ERROR_NOTIFICATION_DESC*.

struct **EC_T_WKCERR_DESC**

Public Members

EC_T_SLAVE_PROP **SlaveProp**
Slave properties, content is undefined in case of cyclic WKC_ERROR

EC_T_BYTE **byCmd**
EtherCAT command type

EC_T_DWORD **dwAddr**
Logical address or physical address (ADP/ADO)

EC_T_WORD **wWkcSet**
Working counter set value

EC_T_WORD **wWkcAct**
Working counter actual value

struct *EC_T_SLAVE_PROP*

EC_T_WORD wStationAddress
EC_T_WORD wAutoIncAddr
EC_T_CHAR achName[MAX_STD_STRLEN]

9.5.5 emNotify - EC_NOTIFY_FRAME_RESPONSE_ERROR

This notification will be indicated if the actually received Ethernet frame does not match to the frame expected or if a expected frame was not received.

struct **EC_T_FRAME_RSPERR_DESC**

Public Members

EC_T_BOOL **bIsCyclicFrame**
Indicates whether the lost frame was a cyclic frame

EC_T_FRAME_RSPERR_TYPE **EErrorType**
Frame response error type

EC_T_BYTE **byEcCmdHeaderIdxSet**
Expected IDX value, this value is valid only for acyclic frames in case EErrorType is not equal to eRspErr_UNEXPECTED

EC_T_BYTE **byEcCmdHeaderIdxAct**
Actually received IDX value, this value is only valid for acyclic frames in case of EErrorType is equal to: eRspErr_WRONG_IDX and eRspErr_UNEXPECTED

EC_T_WORD **wCycFrameNum**
Number of the lost cyclic frame from the ENI

EC_T_DWORD **dwTaskId**
Cyclic Task Id from the ENI. Only valid if bIsCyclicFrame is set

enum **EC_T_FRAME_RSPERR_TYPE**
Values:

enumerator **eRspErr_UNDEFINED**
undefined

enumerator **eRspErr_NO_RESPONSE**
No Ethernet frame received (timeout, frame loss)

enumerator **eRspErr_WRONG_IDX**
Wrong IDX value in acyclic frame

enumerator **eRspErr_UNEXPECTED**
Unexpected frame was received

enumerator **eRspErr_FRAME_RETRY**
Ethernet frame will be re-sent (timeout, frame loss)

enumerator **eRspErr_RETRY_FAIL**
all retry mechanism fails to re-sent acyclic frames

enumerator **eRspErr_FOREIGN_SRC_MAC**
Frame with MAC from other Master received

enumerator **eRspErr_NON_ECAT_FRAME**
Non EtherCAT frame received

enumerator **eRspErr_CRC**
Ethernet frame with CRC error received

9.5.6 emNotify - EC_NOTIFY_STATUS_SLAVE_ERROR

When processing cyclic frames, the EC-Monitor checks whether the ERROR bit in the AL-STATUS register is set for at least one slave. In this case, this notification is triggered. If the EtherCAT master determines the error information of the slave(s) signal an error, another notification *emNotify - EC_NOTIFY_SLAVE_ERROR_STATUS_INFO* with more precise error information is triggered.

9.5.7 emNotify - EC_NOTIFY_SLAVE_ERROR_STATUS_INFO

This notification will be indicated if the EtherCAT master reads the AL-STATUS and AL-STATUS-CODE registers and the slave signals an error in them. Detailed error information is stored in structure *EC_T_SLAVE_ERROR_INFO_DESC* of *EC_T_ERROR_NOTIFICATION_DESC*.

struct **EC_T_SLAVE_ERROR_INFO_DESC**

Public Members

EC_T_SLAVE_PROP **SlaveProp**
Slave properties

EC_T_WORD **wStatus**
Slave Status (AL Status)

EC_T_WORD **wStatusCode**
Error status code (AL STATUS CODE)

9.5.8 emNotify - EC_NOTIFY_PDIWATCHDOG

This notification will be indicated every time a PDI watchdog error is detected. Detailed error information is stored in structure *EC_T_PDIWATCHDOG_DESC* of *EC_T_ERROR_NOTIFICATION_DESC*.

struct **EC_T_PDIWATCHDOG_DESC**

Public Members

EC_T_SLAVE_PROP **SlaveProp**
Slave properties

9.5.9 emNotify - EC_NOTIFY_COMMUNICATION_TIMEOUT

This notification will be indicated if the EC-Monitor does not detect any EtherCAT communication on the Ethernet tap for a parameterizable timeout. The descriptor of the notification contains information on which port of the Ethernet tap the timeout occurred.

struct **`EC_T_COMMUNICATION_TIMEOUT_NTFY_DESC`**

Public Members

EC_T_BOOL **`bMainTapPortIn`**
EC_TRUE: Timeout occurred at the input port of the Ethernet TAP for the EtherCAT main line

EC_T_BOOL **`bMainTapPortOut`**
EC_TRUE: Timeout occurred at the output port of the Ethernet TAP for the EtherCAT main line

See also:

`EC_T_MONITOR_INIT_PARMS::dwCommunicationTimeoutMsec`

9.5.10 emNotify - EC_NOTIFY_TAP_LINK_STATUS

This notification will be indicated if the link status between EC-Monitor and Ethernet TAP device has changed.

struct **`EC_T_TAP_LINK_STATUS_NTFY_DESC`**

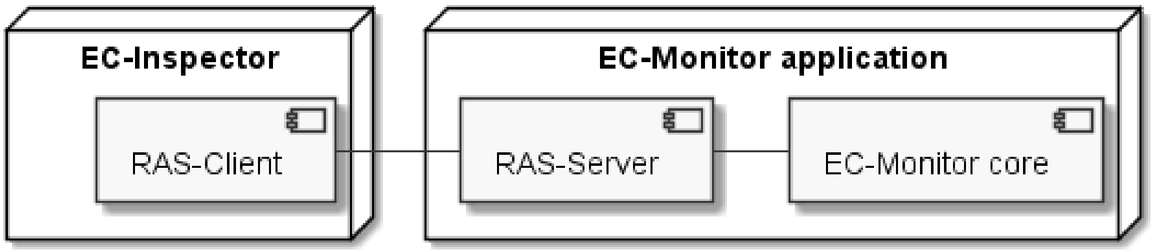
Public Members

EC_T_BOOL **`bLinkConnected`**
Link status of EC-Monitor - Ethernet Tap connection

10 RAS-Server for EC-Inspector and EC-Engineer

10.1 Integration Requirements

To use the diagnosis tool EC-Inspector with a customer application, some modifications have to be done during integration of the EC-Monitor. 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-Inspector.



An example on how to integrate the Remote API Server within the application is given with the example application, which in case is pre-configured to listen for EC-Inspector on TCP Port 6000 when command line 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 “EcMonitorRasSrv.lib” (or respectively “EcMonitor-RasSrv.a”) must be linked.

10.2 Application programming interface

10.2.1 emRasSrvStart

EC_T_DWORD EC_NAMESPACE::**emRasSrvStart**(
ATEMRAS_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 **ATEMRAS_T_SRVPARMS**

Public Members

EC_T_DWORD **`dwSignature`**
: [in] Set to ATEMRASSRV_SIGNATURE

EC_T_DWORD **`dwSize`**
: [in] Set to sizeof(ATEMRAS_T_SRVPARMS)

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

ATEMRAS_T_IPADDR **`oAddr`**
: [in] Server Bind IP Address

EC_T_WORD **`wPort`**
: [in] Server Bind IP 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

10.2.2 emRasSrvStop

EC_T_DWORD `EC_NAMESPACE::`**`emRasSrvStop`**`(`
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

10.2.3 emRasNotify

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

typedef *EC_T_DWORD* (**EC_PF_NOTIFY*)(*EC_T_DWORD* dwCode, *EC_T_NOTIFYPARMS* *pParms)

10.2.4 emRasNotify - ATEMRAS_NOTIFY_CONNECTION

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

emRasNotify - ATEMRAS_T_CONNOTIFYDESC

Parameter

- `pbyInBuf`: [in] Pointer to data of type ATEMRAS_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

struct **`ATEMRAS_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.

10.2.5 emRasNotify - ATEMRAS_NOTIFY_REGISTER

Notification about a connected application registered a client to the EC-Monitor.

emRasNotify - ATEMRAS_NOTIFY_REGISTER

Parameter

- `pbyInBuf`: [in] Pointer to data of type ATEMRAS_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 **ATEMRAS_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

10.2.6 emRasNotify - ATEMRAS_NOTIFY_UNREGISTER

Notification about a connected application un-registered a client from the EC-Monitor.

emRasNotify - ATEMRAS_NOTIFY_UNREGISTER

Parameter

- `pbyInBuf`: [in] Pointer to data of type ATEMRAS_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

See also:

ATEMRAS_T_REGNOTIFYDESC

10.2.7 emRasNotify - ATEMRAS_NOTIFY_MARSHALERROR

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

emRasNotify - ATEMRAS_NOTIFY_MARSHALERRORDESC

Parameter

- `pbyInBuf`: [in] Pointer to data of type ATEMRAS_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 **`ATEMRAS_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

10.2.8 emRasNotify - ATEMRAS_NOTIFY_ACKERROR

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

emRasNotify - ATEMRAS_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

10.2.9 emRasNotify - ATEMRAS_NOTIFY_NONOTIFYMEMORY

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

emRasNotify - ATEMRAS_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

10.2.10 emRasNotify - ATEMRAS_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.

emRasNotify - ATEMRAS_NOTIFY_STDNOTIFYMEMORYSMALL

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

10.2.11 emRasNotify - ATEMRAS_NOTIFY_MBXNOTIFYMEMORYSMALL

Notification given, when buffer size for Mailbox notifications available in pre-allocated notification store are too small to carry a specific notification. This points to a configuration error. 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. Mailbox notifications should be dimensioned correctly see `emRasSrvStart()`

emRasNotify - ATEMRAS_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

11 Error Codes

11.1 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 erroreous E.g. mailbox command queue not large enough
4	Bus/Slave Error	SLV	Slave error E.g. Working Counter Error
5	Link Layer	LLA	Link Layer errror (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

11.2 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)

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_PARMS.dwMaxAcycFramesQueued too small)

EC_E_LINK_DISCONNECTED
0x9811002D: SLV: Ethernet link cable disconnected (e.g. EtherCAT bus segment not connected to network interface)

EC_E_MASTERCORE_INACCESSIBLE
0x9811002E: RAS: Master core not accessible (e.g. Connection to remote server was terminated or master instance has been stopped on remote side)

EC_E_COE_MBXSND_WKC_ERROR
0x9811002F: SLV: CoE mailbox send: working counter (e.g. CoE mailbox couldn’t be read on slave, slave didn’t read out mailbox since last write)

EC_E_COE_MBXRCV_WKC_ERROR
0x98110030: SLV: CoE mailbox receive: working counter (e.g. CoE mailbox couldn’t be read from slave)

EC_E_NO_MBX_SUPPORT
0x98110031: APP: No mailbox support (e.g. Slave does not support mailbox access)

EC_E_NO_COE_SUPPORT
0x98110032: ENI: CoE protocol not supported (e.g. Configuration error or slave information file doesn’t match slave firmware)

EC_E_NO_EOE_SUPPORT
0x98110033: ENI: EoE protocol not supported (e.g. Configuration error or slave information file doesn’t match slave firmware)

EC_E_NO_FOE_SUPPORT
0x98110034: ENI: FoE protocol not supported (e.g. Configuration error or slave information file doesn’t match slave firmware)

EC_E_NO_SOE_SUPPORT
0x98110035: ENI: SoE protocol not supported (e.g. Configuration error or slave information file doesn’t match slave firmware)

EC_E_NO_VOE_SUPPORT
0x98110036: ENI: VoE protocol not supported (e.g. Configuration error or slave information file doesn't match slave firmware)

EC_E_EVAL_VIOLATION
0x98110037: ENI: Configuration violates Evaluation limits (obsolete)

EC_E_EVAL_EXPIRED
0x98110038: CFG: Evaluation Time limit reached (e.g. License not provided and evaluation period (1 hour) of protected version exceeded)

EC_E_LICENSE_MISSING
0x98110039: License key invalid or missing

EC_E_CFGFILENOTFOUND
0x98110070: CFG: 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 to big

EC_E_AOE_NOTINIT
0x98110156: AoE: Device not initialized

EC_E_AOE_TIMEOUT
0x98110157: AoE: Device has a timeout

EC_E_AOE_NOINTERFACE
0x98110158: AoE: Query interface failed

EC_E_AOE_INVALIDINTERFACE
0x98110159: AoE: Wrong interface required

EC_E_AOE_INVALIDCLSID
0x9811015A: AoE: Class ID invalid

EC_E_AOE_INVALIDOBJID
0x9811015B: AoE: Object ID invalid

EC_E_AOE_PENDING
0x9811015C: AoE: Request pending

EC_E_AOE_ABORTED
0x9811015D: AoE: Request aborted

EC_E_AOE_WARNING
0x9811015E: AoE: Signal warning

EC_E_AOE_INVALIDARRAYIDX
0x9811015F: AoE: Invalid array index

EC_E_AOE_SYMBOLNOTACTIVE
0x98110160: AoE: Symbol not active -> release handle and try again

EC_E_AOE_ACCESSDENIED
0x98110161: AoE: Access denied

EC_E_AOE_INTERNAL
0x98110162: AoE: Internal error

EC_E_AOE_TARGET_PORT_NOT_FOUND
0x98110163: AoE: Target port not found

EC_E_AOE_TARGET_MACHINE_NOT_FOUND
0x98110164: AoE: Target machine not found

EC_E_AOE_UNKNOWN_CMD_ID
0x98110165: AoE: Unknown command ID

EC_E_AOE_PORT_NOT_CONNECTED
0x98110166: AoE: Port not connected

EC_E_AOE_INVALID_AMS_LENGTH
0x98110167: AoE: Invalid AMS length

EC_E_AOE_INVALID_AMS_ID
0x98110168: AoE: invalid AMS Net ID

EC_E_AOE_PORT_DISABLED
0x98110169: AoE: Port disabled

EC_E_AOE_PORT_CONNECTED
0x9811016A: AoE: Port already connected

EC_E_AOE_INVALID_AMS_PORT
0x9811016B: AoE: Invalid AMS port

EC_E_AOE_NO_MEMORY
0x9811016C: AoE: No memory

EC_E_AOE_VENDOR_SPECIFIC
0x9811016D: AoE: Vendor specific AoE device error

EC_E_XML_AOE_NETID_INVALID
0x9811016E: ENI: AoE: Invalid NetID (e.g. Error from Configuration Tool)

EC_E_MAX_BUS_SLAVES_EXCEEDED
0x9811016F: CFG: Error: Maximum number of bus slave has been exceeded (The maximum number of preallocated bus slave objects are to 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_PARMS_SIMULATOR::qwOemKey (SiL))

EC_E_S2SMBX_NOT_CONFIGURED
0x98130020: S2S: Not Configured

EC_E_S2SMBX_NO_MEMORY
0x98130021: S2S: No Memory

EC_E_S2SMBX_NO_DESCRIPTOR
0x98130022: S2S: No Descriptor

EC_E_S2SMBX_DEST_SLAVE_NOT_FOUND
0x98130023: S2S: Destination Slave not found

EC_E_MASTER_RED_STATE_INACTIVE
0x98130024: APP: Master Redundancy State is INACTIVE (e.g. API not allowed in current Master Redundancy State)

EC_E_MASTER_RED_STATE_ACTIVE
0x98130025: APP: Master Redundancy State is ACTIVE (e.g. API not allowed in current Master Redundancy State)

EC_E_JUNCTION_RED_LINE_BREAK
0x98130026: Junction redundancy line break

EC_E_VALIDATION_ERROR
0x98130027: Validation error (validation data mismatch)

EC_E_TIMEOUT_WAITING_FOR_DC
0x98130028: Timeout waiting for DC

EC_E_TIMEOUT_WAITING_FOR_DCM
0x98130029: Timeout waiting for DCM

EC_E_SIGNATURE_MISMATCH
0x98130030: Signature mismatch

EC_E_PDIWATCHDOG
0x98130031: PDI watchdog expired

EC_E_BAD_CONNECTION
0x98130032: Bad connection

EC_E_XML_INCONSISTENT
0x98130033: ENI: Inconsistent content

11.3 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

11.4 ADS over EtherCAT (AoE) Error Codes

EC_E_AOE_NO_RTIME
0x9813000D: AoE: No Rtime

EC_E_AOE_LOCKED_MEMORY
0x9813000E: AoE: Allocation locked memory

EC_E_AOE_MAILBOX
0x9813000F: AoE: Insert mailbox error

EC_E_AOE_WRONG_HMSG
0x98130010: AoE: Wrong receive HMSG

EC_E_AOE_BAD_TASK_ID
0x98130011: AoE: Bad task ID

EC_E_AOE_NO_IO
0x98130012: AoE: No IO

EC_E_AOE_UNKNOWN_AMS_COMMAND
0x98130013: AoE: Unknown ADS command

EC_E_AOE_WIN32
0x98130014: AoE: Win 32 error

EC_E_AOE_LOW_INSTALL_LEVEL
0x98130015: AoE: Low installation level

EC_E_AOE_NO_DEBUG
0x98130016: AoE: No debug available

EC_E_AOE_AMS_SYNC_WIN32
0x98130017: AoE: Sync Win 32 error

EC_E_AOE_AMS_SYNC_TIMEOUT
0x98130018: AoE: Sync Timeout

EC_E_AOE_AMS_SYNC_AMS
0x98130019: AoE: Sync AMS error

EC_E_AOE_AMS_SYNC_NO_INDEX_MAP
0x9813001A: AoE: Sync no index map

EC_E_AOE_TCP_SEND
0x9813001B: AoE: TCP send error

EC_E_AOE_HOST_UNREACHABLE
0x9813001C: AoE: Host unreachable

EC_E_AOE_INVALIDAMSFRAGMENT
0x9813001D: AoE: Invalid AMS fragment

`EC_E_AOE_NO_LOCKED_MEMORY`
0x9813001E: AoE: No allocation locked memory

`EC_E_AOE_MAILBOX_FULL`
0x9813001F: AoE: Mailbox full

11.5 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 an file error) (CoE abort code 0x08000023 of slave)

EC_E_SDO_ABORTCODE_UNKNOWN
0x9811005D: SLV: SDO: Unknown code (Unknown CoE abort code of slave)

EC_E_SDO_ABORTCODE_MODULE_ID_LIST_NOT_MATCH
0x9811005E: Detected Module Ident List (0xF030) and Configured Module Ident list (0xF050) does not match

EC_E_SDO_ABORTCODE_SI_NOT_WRITTEN
: 0x98130004: SLV: SDO: Sub Index cannot be written, SI0 must be 0 for write access (CoE abort code 0x06010003 of slave)

EC_E_SDO_ABORTCODE_CA_TYPE_MISM
: 0x98130005: SLV: SDO: Complete access not supported for objects of variable length such as ENUM object types (CoE abort code 0x06010004 of slave)

EC_E_SDO_ABORTCODE_OBJ_TOO_BIG
: 0x98130006: SLV: SDO: Object length exceeds mailbox size (CoE abort code 0x06010005 of slave)

EC_E_SDO_ABORTCODE_PDO_MAPPED
: 0x98130007: SLV: SDO: Object mapped to RxPDO, SDO Download blocked (CoE abort code 0x06010006 of slave)

11.6 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)

11.7 Servo Drive Profil over EtherCAT (SoE) Error Codes

EC_E_SOE_ERRORCODE_INVALID_ACCESS
0x98110078: ERROR SoE: Invalid access to element 0

EC_E_SOE_ERRORCODE_NOT_EXIST
0x98110079: ERROR SoE: Does not exist

EC_E_SOE_ERRORCODE_INVL_ACC_ELEM1
0x9811007A: ERROR SoE: Invalid access to element 1

EC_E_SOE_ERRORCODE_NAME_NOT_EXIST
0x9811007B: ERROR SoE: Name does not exist

EC_E_SOE_ERRORCODE_NAME_UNDERSIZE
0x9811007C: ERROR SoE: Name undersize in transmission

EC_E_SOE_ERRORCODE_NAME_OVERSIZE
0x9811007D: ERROR SoE: Name oversize in transmission

EC_E_SOE_ERRORCODE_NAME_UNCHANGE
0x9811007E: ERROR SoE: Name unchangeable

EC_E_SOE_ERRORCODE_NAME_WR_PROT
0x9811007F: ERROR SoE: Name currently write-protected

EC_E_SOE_ERRORCODE_UNDERS_TRANS
0x98110080: ERROR SoE: Attribute undersize in transmission

EC_E_SOE_ERRORCODE_OVERS_TRANS
0x98110081: ERROR SoE: Attribute oversize in transmission

EC_E_SOE_ERRORCODE_ATTR_UNCHANGE
0x98110082: ERROR SoE: Attribute unchangeable

EC_E_SOE_ERRORCODE_ATTR_WR_PROT
0x98110083: ERROR SoE: Attribute currently write-protected

EC_E_SOE_ERRORCODE_UNIT_NOT_EXIST
0x98110084: ERROR SoE: Unit does not exist

EC_E_SOE_ERRORCODE_UNIT_UNDERSIZE
0x98110085: ERROR SoE: Unit undersize in transmission

EC_E_SOE_ERRORCODE_UNIT_OVERSIZE
0x98110086: ERROR SoE: Unit oversize in transmission

EC_E_SOE_ERRORCODE_UNIT_UNCHANGE
0x98110087: ERROR SoE: Unit unchangeable

EC_E_SOE_ERRORCODE_UNIT_WR_PROT
0x98110088: ERROR SoE: Unit currently write-protected

EC_E_SOE_ERRORCODE_MIN_NOT_EXIST
0x98110089: ERROR SoE: Minimum input value does not exist

EC_E_SOE_ERRORCODE_MIN_UNDERSIZE
0x9811008A: ERROR SoE: Minimum input value undersize in transmission

EC_E_SOE_ERRORCODE_MIN_OVERSIZE
0x9811008B: ERROR SoE: Minimum input value oversize in transmission

EC_E_SOE_ERRORCODE_MIN_UNCHANGE
0x9811008C: ERROR SoE: Minimum input value unchangeable

EC_E_SOE_ERRORCODE_MIN_WR_PROT
0x9811008D: ERROR SoE: Minimum input value currently write-protected

EC_E_SOE_ERRORCODE_MAX_NOT_EXIST
0x9811008E: ERROR SoE: Maximum input value does not exist

EC_E_SOE_ERRORCODE_MAX_UNDERSIZE
0x9811008F: ERROR SoE: Maximum input value undersize in transmission

EC_E_SOE_ERRORCODE_MAX_OVERSIZE
0x98110090: ERROR SoE: Maximum input value oversize in transmission

EC_E_SOE_ERRORCODE_MAX_UNCHANGE
0x98110091: ERROR SoE: Maximum input value unchangeable

EC_E_SOE_ERRORCODE_MAX_WR_PROT
0x98110092: ERROR SoE: Maximum input value currently write-protected

EC_E_SOE_ERRORCODE_DATA_NOT_EXIST
0x98110093: ERROR SoE: Data item does not exist

EC_E_SOE_ERRORCODE_DATA_UNDERSIZE
0x98110094: ERROR SoE: Data item undersize in transmission

EC_E_SOE_ERRORCODE_DATA_OVERSIZE
0x98110095: ERROR SoE: Data item oversize in transmission

EC_E_SOE_ERRORCODE_DATA_UNCHANGE
0x98110096: ERROR SoE: Data item unchangeable

EC_E_SOE_ERRORCODE_DATA_WR_PROT
0x98110097: ERROR SoE: Data item currently write-protected

EC_E_SOE_ERRORCODE_DATA_MIN_LIMIT
0x98110098: ERROR SoE: Data item less than minimum input value limit

EC_E_SOE_ERRORCODE_DATA_MAX_LIMIT
0x98110099: ERROR SoE: Data item exceeds maximum input value limit

EC_E_SOE_ERRORCODE_DATA_INCOR
0x9811009A: ERROR SoE: Data item incorrect

EC_E_SOE_ERRORCODE_PASWD_PROT
0x9811009B: ERROR SoE: Data item protected by password

EC_E_SOE_ERRORCODE_TEMP_UNCHANGE
0x9811009C: ERROR SoE: Data item temporary unchangeable (in AT or MDT)

EC_E_SOE_ERRORCODE_INVL_INDIRECT
0x9811009D: ERROR SoE: Invalid indirect

EC_E_SOE_ERRORCODE_TEMP_UNCHANGE1
0x9811009E: ERROR SoE: Data item temporary unchangeable (parameter or opmode)

EC_E_SOE_ERRORCODE_ALREADY_ACTIVE
0x9811009F: ERROR SoE: Command already active

EC_E_SOE_ERRORCODE_NOT_INTERRUPT
0x98110100: ERROR SoE: Command not interruptible

EC_E_SOE_ERRORCODE_CMD_NOT_AVAIL
0x98110101: ERROR SoE: Command not available (in this phase)

EC_E_SOE_ERRORCODE_CMD_NOT_AVAIL1
0x98110102: ERROR SoE: Command not available (invalid parameter)

EC_E_SOE_ERRORCODE_DRIVE_NO
0x98110103: ERROR SoE: Response drive number not identical with requested drive number

EC_E_SOE_ERRORCODE_IDN
0x98110104: ERROR SoE: Response IDN not identical with requested IDN

EC_E_SOE_ERRORCODE_FRAGMENT_LOST
0x98110105: ERROR SoE: At least one fragment lost

EC_E_SOE_ERRORCODE_BUFFER_FULL
0x98110106: ERROR SoE: RX buffer full (EtherCAT call with to small data-buffer)

EC_E_SOE_ERRORCODE_NO_DATA
0x98110107: ERROR SoE: No data state

EC_E_SOE_ERRORCODE_NO_DEFAULT_VALUE
0x98110108: ERROR SoE: No default value

EC_E_SOE_ERRORCODE_DEFAULT_LONG
0x98110109: ERROR SoE: Default value transmission too long

EC_E_SOE_ERRORCODE_DEFAULT_WP
0x9811010A: ERROR SoE: Default value cannot be changed, read only

EC_E_SOE_ERRORCODE_INVL_DRIVE_NO
0x9811010B: ERROR SoE: Invalid drive number

EC_E_SOE_ERRORCODE_GENERAL_ERROR
0x9811010C: ERROR SoE: General error

`EC_E_SOE_ERRCODE_NO_ELEM_ADR`
0x9811010D: ERROR SoE: No element addressed

11.8 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)