acontis technologies GmbH

SOFTWARE

EC-Master

EtherCAT® Master Stack Class B

Version 2.8

Edition: 2016-03-15

Content - compact

Content

1 Introduction

1.1 What is EtherCAT?

EtherCAT is an IEEE802.3 Ethernet based fieldbus system. EtherCAT defines a new standard in communication speed and is due to its flexible topology and simple configuration to handle like a conventional Fieldbus. The implementation of EtherCAT is inexpensive to implement which allows the system to use fieldbus technology in applications which had to ommit fieldbus use in the past. EtherCAT is an open technology which is standardized within the IEC (International Electrotechnical Commission). The system itself is supported and powered by the EtherCAT Technology Group, which is an international community of users and vendors where more than 3100 members already joined; among them you may also find acontis technologies GmbH.

Fieldbusses are proved and established in automation and most applications depend on them. The use of PC based control systems in a reasonable way was only made possible by the introduction of fieldbus technology. Since the control CPU's speed is increasing rapidly (especially with IPC's), the conventional fieldbus systems are moreover become the bottle neck and limit the reachable performance of the control systems. Additionaly the control topology becomes multi layered with some subsided cyclic systems:
- the control task himself
- the fieldbus system
- and propably some local extension busses in the I/O system
- or simply the local firmware cycle in the peripheral device

Because of this latency times are generated which are typically a multiple of 3 or 5 of the control cycle time, which is not a satisfying solution in most applications. On top of the fieldbus systems, to interconnect control systems, ethernet is state of the art for a long time. The use of Ethernet to control drives or I/O systems is pretty new and was reserved for the conventional fieldbus systems in the past. In this focus the propability to carry small data, hard real time possibilities and of course low costs are the primary requirements. EtherCAT fulfills those requirements and brings internet technologies to the level of I/O communication.

1.2 EtherCAT protocol

The EtherCAT protocol is optimized for process data transfer and is transported directly within the Ethernet frame thanks to a special Ethertype. It may consist of several EtherCAT telegrams, each serving a particular memory area of the logical process image which can address up to 4 gigabytes in size. The data sequence is independent of the physical order of the Ethernet terminals in the network; addressing can be in any order. Broadcast, Multicast and communication between slaves are possible. Direct Ethernet frame transfer is used in cases where maximum performance is required and the EtherCAT components are operated in the same subnet as the controller.
However, EtherCAT applications are not limited to a single subnet: EtherCAT UDP packages the EtherCAT protocol into UDP/IP datagrams. This enables any control with Ethernet protocol stack to address EtherCAT systems. Even communication across routers into other subnets is possible. In this variant, system performance obviously depends on the real-time characteristics of the control and its Ethernet protocol implementation. The response times of the EtherCAT network itself are hardly restricted at all: the UDP datagram only has to be unpacked in the first station.
In addition to data exchange according to the master/slave principle, EtherCAT is also very suitable for communication between controllers (master/master). Freely addressable network variables for process data and a variety of services for parameterization, diagnosis, programming and remote control cover a wide range of requirements. The data interfaces for master/slave and master/master communication are identical. For slave to slave communication, two mechanisms are available. Upstream devices can communicate to downstream devices within the same cycle and thus extremely fast. Since this method is topology dependent, it is particularly suitable for slave to slave communication relationships given by machine design - e.g. in printing or packaging applications. For freely configurable slave to slave communication, the second mechanism applies: the data is relayed by the master. Here two cycles are needed, but due to the extraordinary performance of EtherCAT this is still faster than any other approach.
EtherCAT only uses standard frames according to IEEE802.3 - the frames are not shortened. EtherCAT frames can thus be sent from any Ethernet MAC, and standard tools (e.g. monitor) can be used.

1.3 EC-Master – Features

Feature ID: Unique identification used in ETG.1500 EtherCAT Master Classes
*1: According to ETG.1500 Master Classes not mandatory for Class A
*2: According to ETG.1500 Master Classes not mandatory for Class B

Feature name	Short description	EC-Master Class A	EC-Master Class B	Feature ID
Basic Features				
Service Commands	Support of all commands	✓	✓	101
IRQ field in datagram	Use IRQ information from Slave in datagram header	✓	✓	102
Slaves with Device Emulation	Support Slaves with and without application controller	✓	✓	103
EtherCAT State Machine	Support of ESM special behavior	✓	✓	104
Error Handling	Checking of network or slave errors, e.g. Working Counter	✓	✓	105
VLAN	Support VLAN Tagging	✓	-- (*2)	106
EtherCAT Frame Types	Support EtherCAT Frames	✓	✓	107
UDP Frame Types	Support UDP Frames	-- (*1)	-- (*2)	108
Process Data Exchange				
Cyclic PDO	Cyclic process data exchange	✓	✓	201
Multiple Tasks	Different cycle tasks Multiple update rates for PDO	✓	✓	202
Frame repetition	Send cyclic frames multiple times to increase immunity	-- (*1)	-- (*2)	203
Network Configuration				
Online scanning	Network configuration functionality included in EtherCAT Master	✓	✓	301
Reading ENI	Network Configuration taken from ENI file			
Compare Network configuration	Compare configured and existing network configuration during boot-up	✓	✓	302
Explicit Device identification	Identification used for Hot Connect and prevention against cable swapping	✓	✓	303
Station Alias Addressing	Support configured station alias in slave, i.e. enable 2nd Address and use it	✓	✓	304
Access to EEPROM	Support routines to access EEPROM via ESC register	✓	✓	305
Mailbox Support				
Support Mailbox	Main functionality for mailbox transfer	✓	✓	401
Mailbox Resilient Layer	Support underlying resilient layer	✓	✓	402
Multiple Mailbox channels		✓	✓	403
Mailbox polling	Polling Mailbox state in slaves	✓	✓	404

Feature name	Short description	EC-Master Class A	EC-Master Class B	Feature ID
CAN application layer over EtherCAT (CoE)				
SDO Up/Download	Normal and expedited transfer	✓	✓	501
Segmented Transfer	Segmented transfer	✓	✓	502
Complete Access	Transfer the entire object (with all sub-indices) at once	✓	✓	503
SDO Info service	Services to read object dictionary	✓	✓	504
Emergency Message	Receive Emergency messages	✓	✓	505
PDO in CoE	PDO services transmitted via CoE	-- (*1)	-- (*2)	506
EoE				
EoE protocol	Services for tunneling Ethernet frames. includes all specified EoE services	✓	✓	601
Virtual Switch	Virtual Switch functionality	✓	✓	602
EoE Endpoint to Operation Systems	Interface to the Operation System on top of the EoE layer	FP	-- (*2)	603
FoE				
FoE Protocol	Support FoE Protocol	✓	-- (*2)	701
Firmware Up-/Download	Password, FileName should be given by the application	✓	-- (*2)	702
Boot State	Support Boot-State for Firmware Up/Download	✓	-- (*2)	703
SoE				
SoE Services	Support SoE Services	✓	✓	801
AoE				
AoE Protocol	Support AoE Protocol	✓	-- (*2)	901
VoE				
VoE Protocol	External Connectivity supported	✓	-- (*2)	1001
Synchronization with Distributed Clock (DC)				
DC support	Support of Distributed Clock	✓	-- (*2)	1101
Continuous Propagation Delay compensation	Continuous Calculation of the propagation delay	V 2.4.2	-- (*2)	1102
Sync window monitoring	Continuous monitoring of the Synchronization difference in the slaves	✓	-- (*2)	1103
Slave-to-Slave Communication				
via Master	Information is given in ENI file or can be part of any other network configuration Copying of the data can be handled by master stack or master's application	✓	✓	1201
Master information				
Master Object Dictionary	Information is given in ENI file or can be part of any other network configuration Copying of the data can be handled by master stack or master's application	FP	-- (*2)	1301

1.4 Restrictions of the protected version

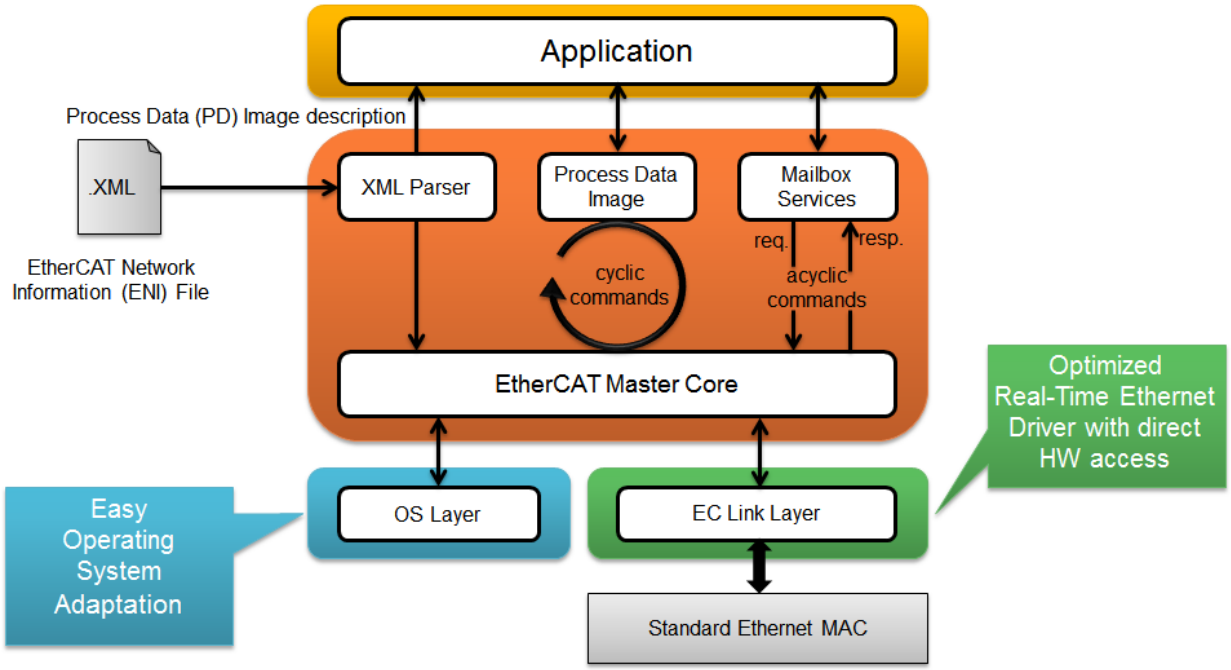
An unlicensed version stops sending ethernet frames after 1 hour (evaluation period expired). Afterwards timeout errors will occur. See 4.3.26 ecatSetLicenseKey.

2 Getting Started

2.1 EC-Master Architecture

The EC-Master EtherCAT Master Stack 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 master can be used in ANSI-C as well as in C++ environments.

The Master Stack is divided into modules, see diagram and descriptions below:



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

2.2 EtherCAT Network Configuration (ENI)

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

Additionally some static configuration parameters have to be defined like the identification of the network adapter card to use, the priority of the EtherCAT master timer task etc.

2.3 Operating system configuration

This chapter is about preparing the operating system for usage with the EC-Master stack. 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. Only network adapters which support at least 100 MBit are supported. First the Link Layer drivers must be chosen as for some, the Operating system must be configured with special steps.

2.3.1 Link Layer selection

2.3.1.1 General

The EtherCAT master stack 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 EtherCAT master stack shipment consist of a master core library (e.g. AtMaster.dll for Windows CE, EcMaster.a for VxWorks) 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 priniciple 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 EtherCAT master 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 master's initialization settings when calling the function ecatInitMaster().

The EtherCAT master 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 EtherCAT master will call the Link Layer receiver polling function prior to processing received frames.

2.3.1.2 Optimized Link Layer drivers

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

Please check in chapter 0 "
" if the optimized Link Layer is available for the target operating system.

2.3.1.3 Supported network controllers

Link Layer Name	Controller / Device ID	Windows XP, 7, …	WinCE	VxWorks	On Time RTOS-32	QNX Linux T-Kernel	RTX INtime	DOS GO32
emllPcap	OS driver	x86	-	-	-	-		-
emllSnarf	OS driver	-	-	x86, PPC	-	-		-
emlll8255x Intel Pro/100	82551QM / 0x1059 82555VE2 / 0x27DC 82557 / 0x1229 82557ER / 0x1209 82559ER / 0x2449 82562 / 0x1039 82801DB / 0x103A 82801EB / 0x1050 Pro/100/M / 0x1229 Pro/100/S / 0x1229 Pro/100/VE /0x1092	-	x86	x86, PPC	x86	x86	x86	-
emlll8254x Intel Pro/1000	82540EM / 0x100E 82541EI / 0x1013 82541ER / 0x1078 82541GI / 0x1076 82541GI / 0x1077 82541PI / 0x107C 82545GM / 0x1026 82546EB / 0x1010 82546GB / 0x1079 82547EI / 0x1075 82547GI / 0x1019 82566DM / 0x104A 82566L / 0x10BD 82566MC / 0x104D 82567V / 0x10CE 82567V / 0x1501 82567LM / 0x10DE 82567LM / 0x10F5 82571GB / 0x10A4 82571GB / 0x10BC 82572GI / 0x10B9 82572PI / 0x107D 82573 / 0x108C 82573E / 0x108B 82573L / 0x109A 82574(L) / 0x10D3 82575 / 0x10A7 82577LM/0x10EA 82577LC / 0x10EB 82576 / 0x10C9 82576 ET2 / 0x1526 82578DM / 0x10EF 82578DC / 0x10F0 82579LM / 0x1502 82579V / 0x1503 82580 / 0x150E 82580 QF / 0x1527 82583V / 0x150C N1E5132 / 0x105E I350 / 0x1521 I210 / 0x1533 I210 CFL / 0x157B I211AT / 0x1539 I217LM / 0x153A I217V / 0x153B I218LM / 0x155A I218V / 0x1559 I219-LM / 0x15B7	x86	x86	x86	x86	x86	x86	-

Link Layer Name	Controller / Device ID	Windows XP, 7, …	WinCE	VxWorks	On Time RTOS-32	QNX Linux T-Kernel	RTX INtime	DOS GO32
emllL9218i SMSC LAN9218i	L9218i	-	ARM STR9	-	-	-		-
emllRTL8139 Realtek RTL8139	8139D / 0x8139 D-Link 8139D / 0x1300	-	x86	x86	-	x86		-
emllRTL8169 Realtek Gigabit NIC (PCIe, PCI)	RTL8110 / 0x8169 RTL8111 / 0x8168 RTL8168 / 0x8168 RTL8169 / 0x8169 D-Link RTL8169 / 0x4300 RTL8169SC/0x8167 RTL8103 / 0x8136	-	x86	x86	x86	x86	x86	-
emllCPSW	Texas Instruments Sitara on board		ARM	ARM		ARM		-
emllETSEC	Freescale PowerPC TSEC/eTSEC controller eTSEC v1 and v2			PPC		PPC		-
emllFslFec	Freescale FEC and ENET controller		ARM	ARM		ARM		-
emllR6040	RDC R6040	-	x86	-	-	-	-	x86

2.3.1.4 Shipment

There are different Link Layer modules available for different platforms. The following table contains the default shipped Link Layer modules.

WINDOWS	WINCE	VXWORKS
emllPcap.dll WinPcap Ident=”Pcap”	*emlll8255x.dll* Intel Pro/100 Ident=”I8255x”	*emllSnarfGpp.out* *emllSnarfPid.out* Snarf Ident=”Snarf”
	emlll8254x.dll Intel Pro/1000 Ident=”I8254x”	*emlll8255x.out* Intel Pro/100 Ident=”I8255x”

ON TIME RTOS-32	QNX NEUTRINO	RTX
emlll8254x.lib Intel Pro/1000 Ident=”I8254x”	*emlll8255x.so* Intel Pro/100 Ident=”I8255x”	*emlll8255x.rtdll* Intel Pro/100 Ident=”I8255x”
	emlll8254x.so Intel Pro/1000 Ident=”I8254x”	*emlll8254x.rtdll* Intel Pro/1000 Ident=”I8254x”

INTIME	T-KERNEL	LINUX
emlll8255x.rsl Intel Pro/100 Ident="I8255x"	*emlll8254x.a* Intel Pro/1000 Ident="I8254x"	*CCAT (x86, x64),* *CPSW (ARM),* *ETSEC (PPC),* *GEM (ARM),* *I8254x (x86, x64),* *I8255x (x86),* *RTL8139 (x86),* *RTL8169 (x86),* *DW3504 (ARM),*
emlll8254x.rsl Intel Pro/1000 Ident="I8254x"		

Link Layer modules not listed here may be available in your installation if purchased additionally.

2.3.1.5 Freescale TSEC / eTSEC

The following MAC's are supported:

TSEC (not tested)
Legacy hardware. Should be supported, because eTSEC is compatible to TSEC if the enhanced functionality is not used.

eTSEC v1 (tested)
This chip is used for QorIQ (i.e. P2020E) and PowerQUICC devices (i.e. MPC8548). It has 4k of IO memory.

eETSEC v2, also called vETSEC, v read as "virtualization" (tested)
This chip is used for newer QorIQ devices (i.e. P1020). It has 12k of IO memory (4k MDIO, 4k Register group0, 4k Register group1)

2.3.1.5.1 Shared MII bus

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

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

Note: the external PHYs are connected physically to the MII bus of the first eTSEC (and/or eTSEC3, depending on SoC type). From SoC reference manuals:

"14.5.3.6.6 MII Management Configuration Register (MIIMCFG)
... Note that MII management hardware is shared by all eTSECs. Thus, only through the MIIM registers of eTSEC1 can external PHYs be accessed and configured."

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

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

   poDrvSpecificParam->dwPhyMdioBase = dwCcsrbar + 0x24000;

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

   poDrvSpecificParam->dwTbiPhyAddr = 16;

2.3.1.5.2 Locking

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

1. poDrvSpecificParam->oMiiBusMtx = EC_NULL;
   /* implement locking by using return value of LinkOsCreateLock(eLockType_DEFAULT); */

2.3.1.5.3 Link check

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

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

```
dwRes = ecatIoControl((EC_IOCTL_LINKLAYER | EC_LINKIOCTL_UPDATE_LINKSTATUS), EC_NULL);
```

EcLinkGetStatus() always retuns the last known link status.

2.3.1.5.4 Fixed Link

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

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

Parameters for fixed link:

```
pETSECParam->dwPhyAddr       = ETSEC_FIXED_LINK;
pETSECParam->dwFixedLinkVal  = ETSEC_LINKFLAG_1000baseT_Full | ETSEC_LINKFLAG_LINKOK;
```

2.3.1.5.5 Build instructions

Linux and VxWorks (PowerPC only) are supported. The GNU GCC compiler is needed to compile.

The following files need to be compiled:

- EcDeviceETSEC.cpp
- LinkOsLayer.cpp (platform-dependend)

2.3.2 DOS GO32-V2

The EC-Master library for DOS 32-bit (GO32) is designed to be used under DPMI extention only, a virtual-memory-disabling DPMI server like `CWSDPR0.exe` has to be used.
It is recommended to set the `CONFIG.SYS` as following:

```
DEVICE=C:\DOS\HIMEM.SYS
DOS=HIGH
FILES=30
```

2.3.3 Greenhills INTEGRITY

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

1. Copy the file `<InstallPath>\SDK\Files\INTEGRITY\ethercat.c` to `<INTEGRITYPath>\modules\acontis\ethercat.c`
2. Open the project `<INTEGRITYPath>\pcx86\default.gpj` according to the used BSP
3. Add `<INTEGRITYPath>\modules\acontis\ethercat.c` to `libbsp.gpj`
4. Recompile the library
5. Recompile kernel space project

2.3.4 IntervalZero RTX

To use Optimized Link Layers under RTX, the network adapter should be assigned to RTX as described in the RTX user manual.

The NIC driver should not use the network adapter for TCP/IP and therefore the network adapter **may not be configured in RtxTcpIp.ini**.

2.3.5 Linux

2.3.5.1 Atemsys for Optimized Link Layers

To use Optimized Link Layers under Linux, the atemsys Kernel module must be compiled and loaded.

Compile the atemsys Kernel module if running EC-Master **natively**:

```
cd Sources/atemsys
make modules
```

Cross-compile the atemsys Kernel module if running EC-Master on a **target**:

```
export ARCH=<…>
export CROSS_COMPILE=<…>
cd Sources/atemsys
make modules KERNELDIR=<path to target kernel dir>
```

Load the atemsys module:

```
sudo insmod Sources/atemsys/atemsys.ko
```

2.3.5.2 Show loaded modules

The following command lists the loaded Kernel modules that may conflict with Optimized Link Layers:
```
lsmod | egrep "<module-name>"
```

E.g.:
```
lsmod | egrep "e1000|e1000e|igb"
```

PCI/PCIe: The command "`lspci`" 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)
...
```

2.3.5.3 Unbind Link Layer instance

Link Layer instances used by Optimized Link Layers may not be bound by Kernel drivers modules!

The following command unbinds an instance without unloading the kernel driver module:
```
echo "<Instance-ID>" > /sys/bus/pci/drivers/<driver-name>/unbind
```

E.g.:
```
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
```

Some optimized Link Layers support PCI bus, device, function codes as instance ID.

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

Modules can be prevented from loading on Ubuntu with the following commands:
```
echo blacklist <module-name> | sudo tee -a /etc/modprobe.d/blacklist.conf
update-initramfs -k all -u
sudo reboot
```

The following table shows the Kernel modules related to the Optimized Link Layers:

Chip	Link Layer Name	Kernel driver (s)	Remarks
Beckhoff CCAT	emllCCAT	ec_bhf	
CPSW	emllCPSW	ti_cpsw	Unbind not supported, see below!
DesignWare 3504	emllDW3504	stmmac	
	emllEG20T		
Freescale TSEC/eTSEC v1/2	emllETSEC	gianfar_driver	
Freescale FEC and ENET controller	emllFslFec	fec, fec_ptp	
Xilinx Zynq-7000	emllGEM		Unbind not supported, see below!
Intel Pro/1000	emlli8254x	igb, e1000, e1000e	
Intel Pro/100	emlli8255x	e100	
RDC R6040	emllR6040		
Realtek RTL8139	emllRTL8139	8139too, 8139cp	

Chip	Link Layer Name	Kernel driver (s)	Remarks
Realtek RTL8169 / RTL8111 / RTL8168	emllRTL8169	r8169	De-initializes PHY, see below!
Generic	emllSockRaw		

2.3.5.4 Optionally modify search location for Optimized Link Layers

Search locations for Optimized Link Layers can be adjusted using `LD_LIBRARY_PATH`.

2.3.5.5 CPSW

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

2.3.5.6 Freescale TSEC / eTSEC

The driver is compiled as dynamic library "libemllETSEC.so" to be loadable by the EC-Master.
The following compiler flags are set:

-te500v2: Generate optimized code for the PPC e500v2 core.

2.3.5.7 Realtek RTL8169 / RTL8111 / RTL8168

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

2.3.5.8 Xilinx Zynq-7000 (GEM) EtherCAT driver

Due to lacking unbind-feature of the GEM driver, the target's Kernel must be patched to not bind the adapter on startup. Therefore the board's Kernel SDK is needed.

The patch is located at `SDK/FILES/GEM/Linux/xilinx_emacps.c.patch`. The patch can be applied using the command "`patch -p1 < /opt/EC-Master/SDK/FILES/GEM/Linux/xilinx_emacps.c.patch`" in the target's Kernel development dir and rebuild the Kernel and its modules.
A correctly patched system shows "xemacps at 0x… disabled (Used for EtherCAT!)" with given address in the kernel logs.

2.3.5.9 SockRaw Link Layer

The SockRaw Link Layer is always part of the EC-Master for Linux package. It does not need the atemsys driver and uses already established Ethernet adapters, e.g. eth0, eth1, etc. It is strongly recommended to use a separate network adapter to connect EtherCAT devices. If the main network adapter is used for both EtherCAT devices and the local area network there may be a main impact on the local area network operation. **The SockRaw cannot be used for real time applications and may need cycle time of 4 ms or higher**.

2.3.6 Microsoft Windows

2.3.6.1 Windows WinPcap based Link Layer

A Link Layer based on the WinPcap library is shipped with the EtherCAT master stack. 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.

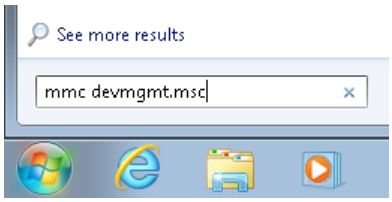
It is recommended to use a separate network adapter to connect EtherCAT devices. If the main network adapter is used for both EtherCAT devices and the local area network there may be a main impact on the local area network operation.
The network adapter card used by EtherCAT has to be set to a fixed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

At least WinPcap version 4.1.2 must be used.

2.3.6.2 EcatDrv for Optimized Link Layers

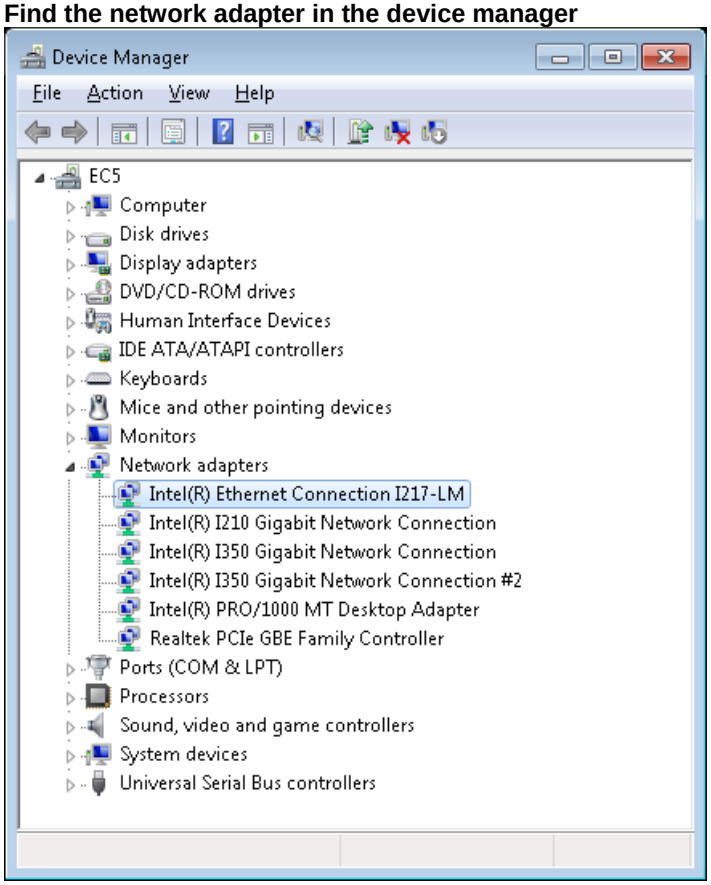
To use the optimized Link Layers under Windows, it is necessary to install the EcatDrv driver included in the optimized Link Layer delivery:

Step 1: Start the "Device Manager"



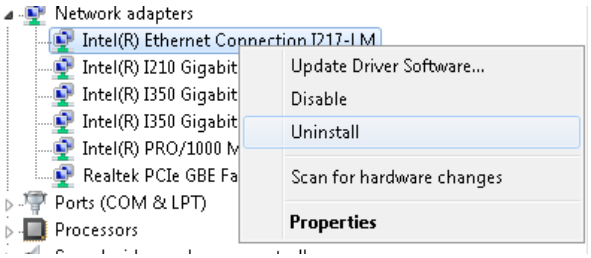
Step 2: Disable the windows driver of the network adapter you want to assign to the ECAT driver

Find the network adapter in the device manager



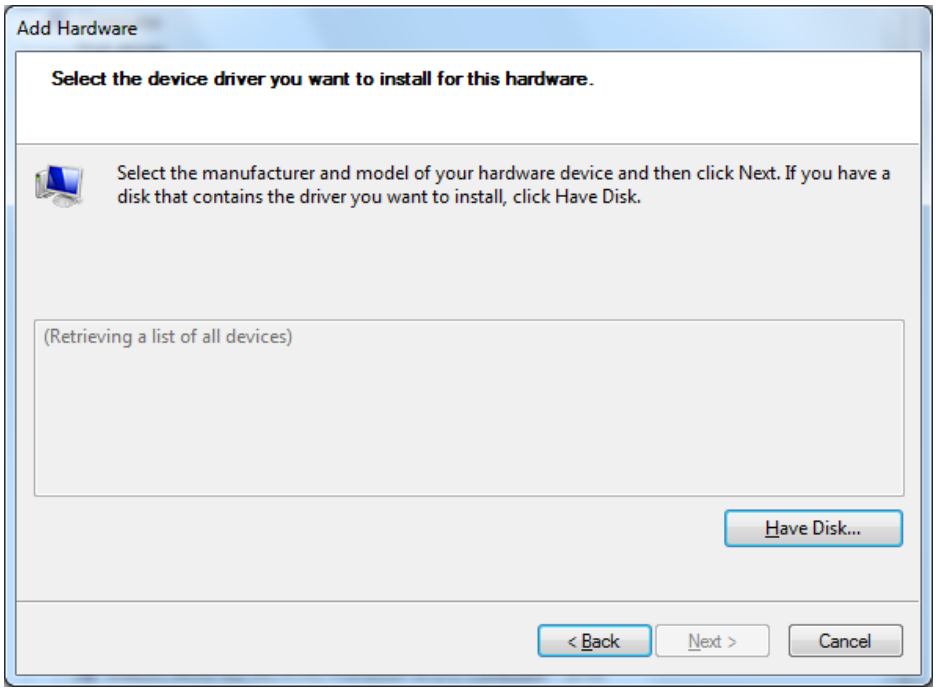
Look in C:\Windows\Inf for .inf files containing the name of the adapter as displayed in the device manager. Rename the .inf files to .bak and delete the assotiated .pnf files.

Step 3: Delete network adapter from the device manager



Restart the system.

Step 4: Assign the ECAT driver to the network adapter

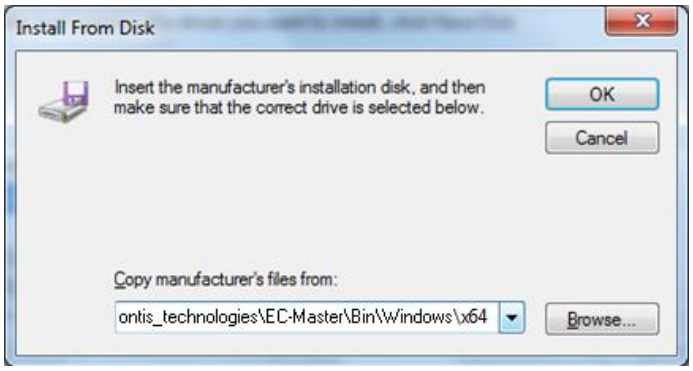


Step 5: Enter the directory to the correct driver version (32 bit or 64 bit)

The default folder if not changed when installing the EC-Master is beneath “C:\Program Files\acontis_technologies\EC-Master\Bin\Windows”.

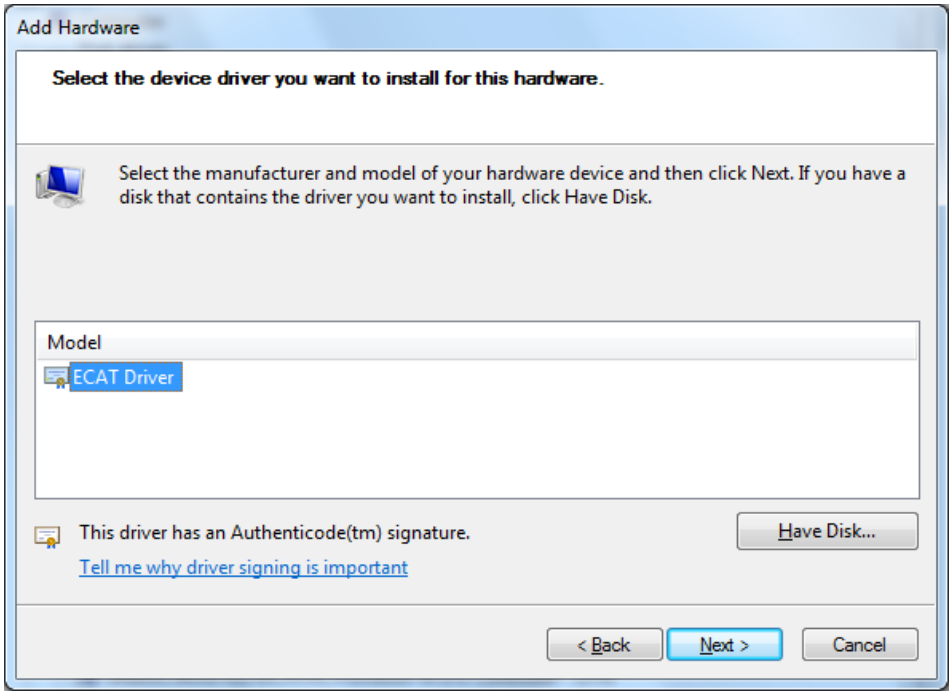
There are two different drivers available: 32 bit (subfolder **x86)** and 64 bit (subfolder **x64)**.

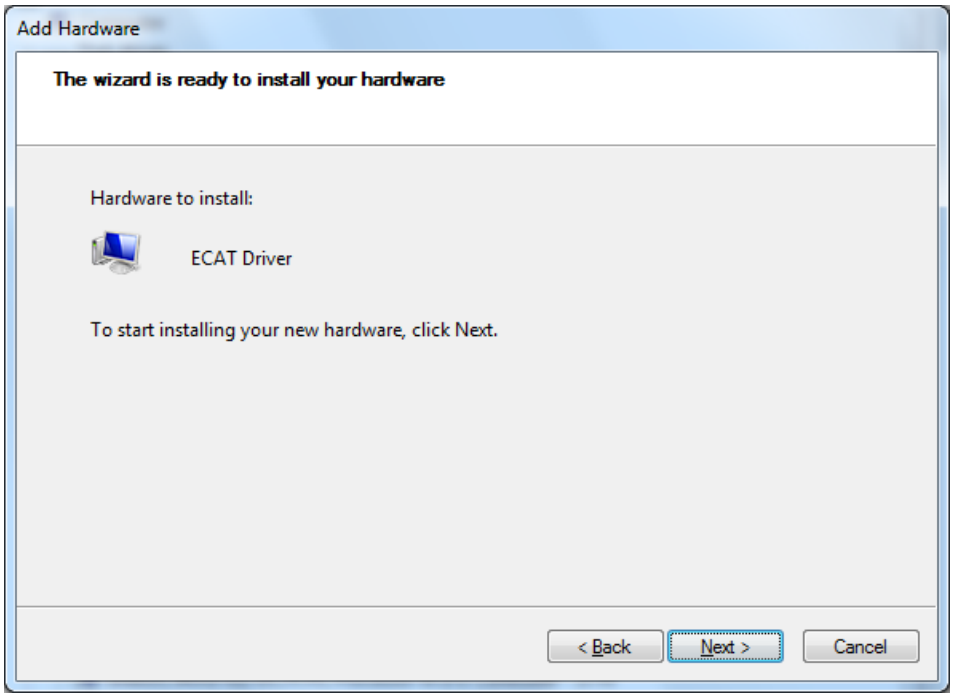
Enter the correct directory at the input box:



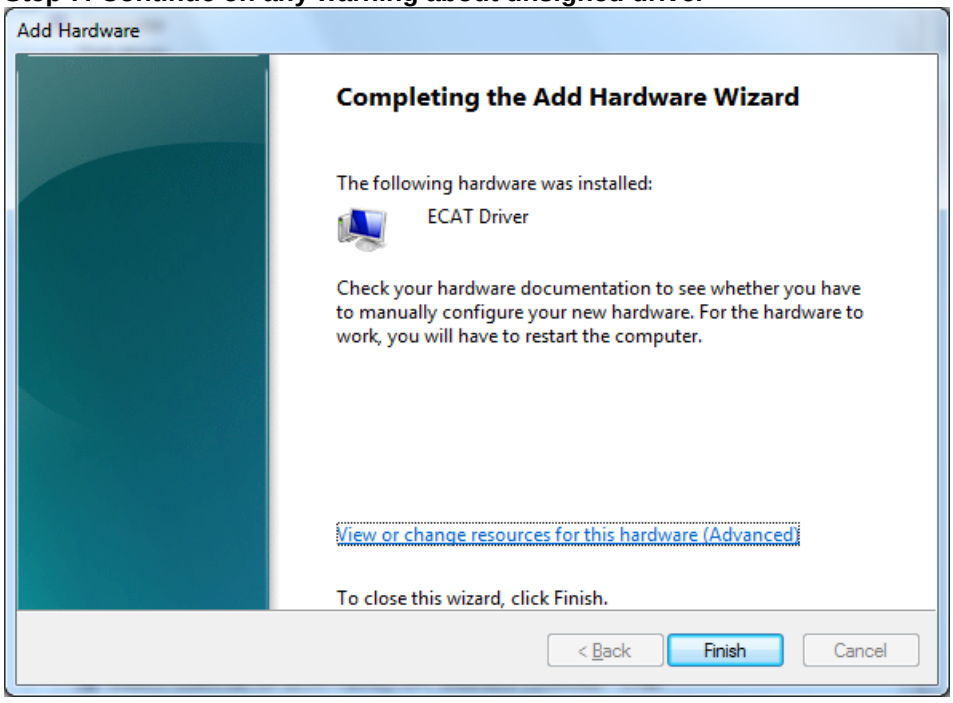
Press OK to proceed.

Step 6: Chose the ECAT Driver and click “Next” and confirm the installation





Step 7: Continue on any warning about unsigned driver



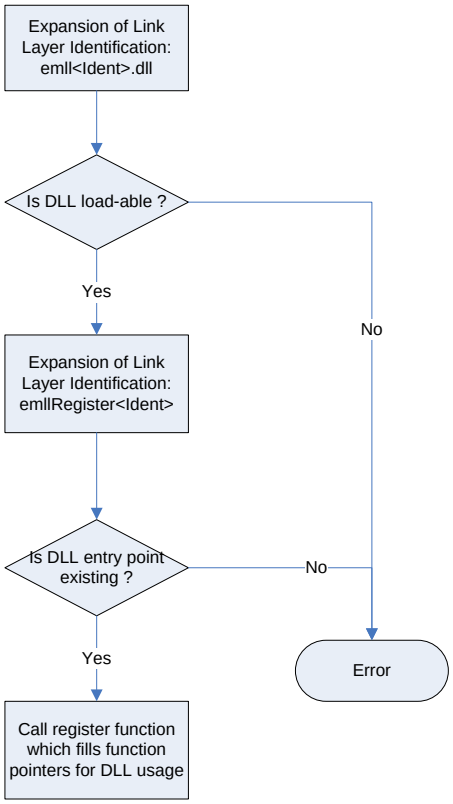
Optionally modify search location for Optimized Link Layers

Search locations for Optimized Link Layers can be adjusted using the PATH environment variable.

2.3.7 Microsoft Windows CE

2.3.7.1.1 Windows and WinCE

The identification of the Link Layer is done like this:



The Link Layer module DLL has to be locatable within the applications DLL search path (local or Windows directory). If it is not, an error is given.

2.3.7.2 NdisUio Link Layer

To be able to use the NDISUIO based Link Layer the following files have to be included to the Windows CE OS-image:

[…]\BIN\WinCE500\NDISUIO*CPU*\AtNdisUio.dll
[…]\BIN \WinCE500\NDISUIO*CPU*\EcMaster.dll
[…]\BIN \WinCE500\NDISUIO*CPU*\emllNdisUio.dll

This is done by use of the files:

[…]\SDK\FILES\EcMaster.bib
[…]\SDK\FILES\Ndisuio\AtNdisUio.bib

The registry entries which have to be added can be taken from:

[…]\SDK\FILES\Ndisuio\AtNdisUio.reg

2.3.7.3 KUKA CeWin

If you are using **KUKA CeWin** (Windows CE runs in parallel with Windows on the same host) the network adapter card to be used has to be assigned to Windows CE.
It is also possible in CeWin to load the NDISUIO filter driver dynamically.
An example how to include the EtherCAT Master using a Realtek RTL8139 Network Interface Card can be found in the directory […]\SDK\FILES\Ndisuio\CeWin (CeWin version 3.3.1):

- Windows INF-File to assign the Realtek NIC to the RTOS (WindowsCE): RTOS_RTL8139.inf
- WinCE image file for Windows CE 4.2 with RTL8139 support: …\3.3.1\WINCE420\RTL8139.zip
- WinCE image file for Windows CE 5.0 with RTL8139 support: …\3.3.1\WINCE500\RTL8139.zip
- Windows CE configuration for the Realtek-NIC: RTL8139.config
- Dynamic start of the NDISUIO-filter driver AtNdisUio.dll via network share: AtNdisUio.config

Note: Due to a bug in Windows CE Version 5.0 you need a workaround to load a DLL (e.g. the NDISUIO driver AtNdisUio.dll) from a network share. This can be done by including the following configuration file into

cewin.config:
[…]\SDK\FILES\Ndisuio\CeWin\CE5_DllLoadFix.config

To create a new Windows CE image which includes the NDISUIO based Link Layer the following files have to be included in the Windows CE OS-image:
[…]\SDK\BIN\NDISUIO\x86\AtNdisUio.dll
[…]\SDK\BIN\NDISUIO*x86*\EcMaster.dll
[…]\SDK\BIN\NDISUIO*x86*\emllNdisUio.dll
This is done by use of the files:
[…]\SDK\FILES\EcMaster.bib
[…]\SDK\FILES\Ndisuio\AtNdisUio.bib
The registry entries which have to be added can be taken from:
[…]\SDK\FILES\Ndisuio\AtNdisUio.reg
The appropriate network adapter card (e.g. the Realtek 8139 adapter card) has to be taken from the Windows CE catalog to include it in the Windows CE image.

2.3.7.4 Windows CE 5.0

To be able to use the optimized Link Layers the following files have to be included to the Windows CE OS-image:
Here the proceedings for Intel PRO/100
[…]\BIN\WinCE500\X86\EcMaster.dll
[…]\Bin\WinCE500\X86\emlII8255x.dll
This is done by use of the files:
[…]\SDK\FILES\EcMaster.bib
The registry entries which have to be added can be taken from:
[…]\SDK\FILES\I8255x\I8255x.reg

Same procedure and settings may be applied for the other optimized Link Layer; i.e. use I8254x instead of I8255x.

Search locations for Optimized Link Layers can be adjusted using the PATH environment variable.

2.3.7.5 Windows CE 6.0

To be able to use the optimized Link Layers the following files have to be included to the Windows CE OS-image:
Here the proceedings for Intel PRO/100
[…]\BIN\WinCE600\EcMaster.dll
[…]\BIN\WinCE600\emlII8255x.dll
[…]\SDK\FILES\I8255x\WinCE600\VirtualDrv.dll
This is done by use of the files:
[…]\SDK\FILES\EcMaster.bib
[…]\SDK\FILES\I8255x\WinCE600\VirtDrv600.bib
The registry entries which have to be added can be taken from:
[…]\SDK\FILES\I8255x\I8255x.600.reg

Same procedure and settings may be applied for the other optimized Link Layer; i.e. use I8254x instead of I8255x.

Search locations for Optimized Link Layers can be adjusted using the PATH environment variable.

2.3.7.6 KUKA CeWin

If using **KUKA CeWin** (Windows CE runs in parallel with Windows on the same host) the network adapter card has to be assigned to Windows CE.
An example how to include the EtherCAT Master using the optimized Intel PRO/100 Network Interface Card can be found in the directory […]\SDK\FILES\I8255x\CeWin (version 3.3.1):

- Windows INF-File to assign the PRO/100 NIC to the RTOS (WindowsCE): RTOS_I8255x.inf
- Windows CE configuration for the PRO/100-NIC: I8255x.config

Note1: Due to a bug in Windows CE Version 5.0 you need a workaround to load a DLL (e.g. for dynamically loading the EtherCAT stack EcMaster.dll) from a network share. This can be done by including the following configuration file into cewin.config:
[…]\SDK\FILES\Ndisuio\CeWin\CE5_DllLoadFix.config
Note2: The images shipped with CeWin can be used together with the Intel PRO/100 optimized Link Layer

For example to create a new Windows CE image which includes the optimized PRO/100 Link Layer the following files have to be included in the Windows CE OS-image:
[…]\BIN\WinCE500\I8255x*x86*\EcMaster.dll
[…]\BIN\WinCE500\I8255x*CPU*\emlII8255x.dll
This is done by use of the file:
[…]\SDK\FILES\EcMaster.bib
The registry entries which have to be added can be taken from:
[…]\SDK\FILES\I8255x\I8255x.reg

2.3.8 QNX Neutrino

2.3.8.1 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 in the “EC-Master EtherCAT Master Stack”, you need to change the allowed priority range for non-root processes with the procnto -P option:
`procnto -P priority`
For more informations about changing the priority range refer to the QNX documentation.

NOTE: Don’t changing the priority range leads to bad timing performance!

2.3.8.2 Optimized Link Layers

The network interface card and Link Layer to be used for the demo application can be set via command line. For example the option `-i8255x` will dynamically load the Link Layer for the Intel Pro/100 interface. The corresponding network interface must be unloaded if in use by a driver using the `umount` (QNX 6.3) or `ifconfig` (QNX 6.5) command in the QNX shell. For example `umount /dev/io-net/en1` or `ifconfig en1 destroy` unloads the driver for the interface `en1`.

The Link Layer drivers for QNX are “so” files (shared object files).
Search locations for Optimized Link Layers can be adjusted using `_CS_LIBPATH`.

2.3.9 RTAI

Step 1: Load required RTAI Kernel modules

Example:

```
> insmod /usr/realtime/modules/rtai_hal.ko
> insmod /usr/realtime/modules/rtai_sched.ko
> insmod /usr/realtime/modules/rtai_shm.ko
```

Step 2: Make sure that the native Linux driver for your EtherCAT network card is not loaded

Note: SockRaw cannot be used with RTAI.

See 2.4.1.3 “Linux” for how to unload drivers that may use network adapter.

2.3.10 RTEMS

Build the RTEMS operating system with least following configuration parameters:

```
--target=i386-rtems4.11 --enable-rtemsbsp=pc386 --enable-cxx
```

2.3.11 T-Engine T-Kernel

2.3.11.1 Optimized Link Layers

Optimized Link Layers are available for T-Kernel.
To be able to use the optimized Link Layers you have to make sure the driver for the ethernet adapter you want to use is not loaded by T-Kernel.

2.3.11.2 System Timer

For T-Kernel the default interval of the system timer interrupt ("time tick") is 10 ms. The EtherCAT Master Stack needs a system timer interrupt of 1 ms.

Changing the "time tick" interval:
The period of the system timer interrupt ("time tick" inteval) is defined by the "TTimPeriod" entry of the "SYSCONF" file.
You can change it by the "sysconf"(lowercase letters) command as follows:

```
[/SYS]% sysconf TTimPeriod      # check current value
+0: TTimPeriod 10               # timer interrrupt interval (in milliseconds)

[/SYS]% sysconf TTimPeriod 1    # change value to 1 ms
+0: TTimPeriod 1

[/SYS]% exit                    # exit CLI
[IMS]% exit -1                  # system reboot
```

Please note that the changed value becomes valid after system reboot.

2.3.12 tenAsys INtime

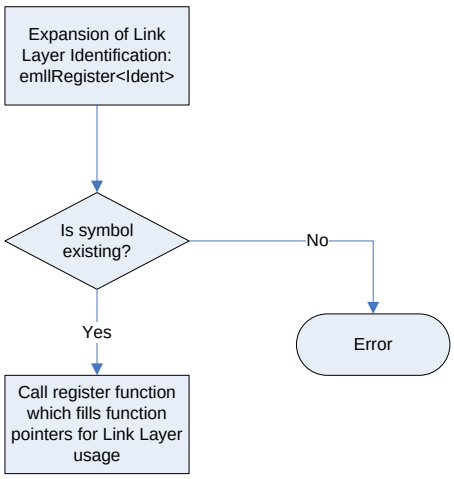
Optimized Link Layers are available for INtime.
If you are using INtime with Windows runs in parallel on the same host the network adapter card has to be assigned to INtime
The network adapters should be passed to INtime using the “INtime Device Manager”. Please refer to the INtime user manual for this.

Search locations for Optimized Link Layers can be adjusted using the PATH environment variable.

2.3.13 Windriver VxWorks

Optimized Link Layers for VxWorks are available.
If none of the optimized Link Layers can be used, the SNARF or EtherLib Link Layer must be selected (see below).

The identification of the Link Layer is done like this:



The Link Layer module has to be downloaded before the master stack is started. If the register function is not found in the global symbol table of VxWorks an error is given.

2.3.13.1 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},
#endif /* INCLUDE_FEI_END */
*/
      :     :     :
      :     :     :
```

2.3.13.2 KUKA VxWin

If using **KUKA VxWin** (Windows runs on top of VxWorks) the network adapter card used for EtherCAT has to be assigned to VxWorks. On APIC systems the interrupt used by the network card has to be activated for VxWorks (configuration file interrupt.config, a detailled description can be found in the VxWin manual).

I.e. for the optimized Intel Pro/100 Link Layer the following VxWin support files are located in …\SDK\FILES\I8255X\VXWIN (VxWin version 3.1.1):

- Windows INF file to assign a Intel Pro/100 network adapter card to VxWorks: RTOS_I82557.inf
- VxWorks 6.3 image file which maps Intel Pro/100 PCI memory into VxWorks space: V3.1.1\VxWorks63\VxWorks63_OPT_PRO100.zip (two files, one with ELF file format to be used by the target server and a second binary file for the VxWin uploader).

The VxWin board support package has to be adjusted prior to using this Link Layer:

1) I.e. to use the optimized PRO/100 Link Layer the adapter memory has to be mapped into VxWorks memory space. This can be achieved by setting the INCLUDE_FEI_END macro in the BSP configuration file config.h.
2) To avoid conflicts with the VxWorks network driver which normally will be loaded when INCLUDE_FEI_END is set the file configNet.h has to be adjusted in a way that the network driver is not loaded. The network driver entry has to be removed from the endDevTbl[]:

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

2.3.13.3 Freescale TSEC / eTSEC

2.3.13.3.1 Build instructions for VxWorks

There is no support for the WindRiver DIAB compiler. In order to use this driver in VxWorks BSP's which are DIAB compiled, this driver should be compiled as C-Code (not C++).
The driver should be compiled with the standard WindRiver Workbench settings for a "Downloadable Kernel Module" project with module name "emllETSEC.out". Recommended compiler flags:

Debug Build:
  -DDEBUG

Debug and Release Build:
  -xc -std=c99 -DVXWORKS -DNO_PCI_SUPPORT -mspe=no

-xc:	Generate C Code, ignore file extension.
-std=c99:	Allows use of "//" as comment.
-DVXWORKS	Hint for the Os-Layer.
-DNO_PCI_SUPPORT	Don't reference pciLib in VxWorks's Os-Layer.
-mspe=no	Forbid generation of code which accesses the SPE unit. EC-Master VxWorks tasks are started without VX_FP_TASK or VX_SPE_TASK flags. Without this option "SPE Unavailable Exception" may raised.

2.3.13.3.2 VxWorks driver interactions

There may be concurrent accesses by the VxWorks and our driver if the "motetsec" VxWorks driver is configured into the BSP. Due to 3.1.1, this may also happen if the VxWorks driver is disabled (i.e. muxDevUnload() for the specified interface).

The mutex handle, specified by poDrvSpecificParam->oMiiBusMtx can be used to synchronize the concurrent accesses to the MDIO registers. Note that this handle must be created by LinkOsCreateLock(). You could also change the LinkOsLock() and LinkOsUnlock() calls in EcDeviceETSEC.cpp, so that a native VxWorks mutex handle (SEM_ID) can be used instead.
Pseudo code for getting the device mutex (see vxbEtsecEnd.c):

```
VXB_DEVICE_ID miiDev = vxbInstByNameFind ("motetsec", 0);
ETSEC_DRV_CTRL *pDrvCtrl = miiDev->pDrvCtrl;
semTake(pDrvCtrl->etsecDevSem);
...
semGive(pDrvCtrl->etsecDevSem);
```

There may also be a VxWorks task running (miiBusMonitor), which periodically
checks the link status of attached PHY's. It is recommended to disable this task in order to avoid potential link problems.

2.3.13.4 SNARF / EtherLib Link Layer

The SNARF or EtherLib Link Layer is only needed if none of the optimized Link Layers can be used (see above).

The appropriate network adapter drivers have to be added to the VxWorks image. The demo requires the VxWorks image to support the Intel Pro/100 network card – FEI device and the Intel Pro/1000 network card – GEI device. The macros INCLUDE_FEI_END and INCLUDE_GEI8254X_END or respectively INCLUDE_..._VXB_END have to be set in the BSP configuration file config.h.

If using **KUKA VxWin** (Windows runs on top of VxWorks) the network adapter card used for EtherCAT has to be assigned to VxWorks. On APIC systems the interrupt used by the network card has to be activated for VxWorks (configuration file interrupt.config, a detailled description can be found in the VxWin manual).

The following VxWin support files are located in …\SDK\FILES\SNARF\VXWIN (VxWin version 3.1.1):

- Windows INF file to assign a Intel Pro/100 network adapter card to VxWorks: RTOS_I82557.inf
- VxWorks 6.1 image file containing Intel Pro/100 driver: V3.1.1\VxWorks61\VxWorks61_SNARF_PRO100.zip (two files, one with ELF file format to be used by the target server and a second binary file for the VxWin uploader).

2.3.14 Xenomai

The system must be setup first the same way as for EC-Master for Linux, see chapter 2.3.5 “Linux”, especially installation of the atemsys module and optimized Link Layer usage preparation.

The binaries are built using the following versions:

- armv6-vfp-eabihf: Xenomai 2.6.3, tested on Linux Kernel 3.8.13-xenomai-2.6.4
- x86: Xenomai 2.6.2.1, tested on Linux Kernel 3.5.7

2.4 Running EcMasterDemo

The EcMasterDemo is an EC-Master example application that handles the following tasks:

- Showing basic EtherCAT communication
- Master stack initialization into OPERATIONAL state
- Process Data operations for e.g. Beckhoff EL2004, EL1004 and EL4132
- Periodic diagnosis task
- Periodic Job Task in polling mode
- Logging to `ecmaster0.log`, `error0.log`

See chapter 3.1 “Application framework and example application” for detailed explanation.

The EcMasterDemo is available “out of the box” for different operating systems. The operating system must be prepared for running EC-Master applications, see 2.3, “Operating system configuration”.

2.4.1 Setting up and running the demo

2.4.1.1 DOS GO32-V2

Step 1: Operating system configuration
See the section Operating system configuration for how to prepare the operating system

Step 2: Starting EcMasterDemo
The file EcMasterDemo.exe has to be executed.The full path and file name of the configuration file has to be given as a command line parameter as well as the appropriate Link Layer. The DPMI server CSWDPR0.exe has to be present in system either in the same folder as EcMasterDemo.exe or has to be accessible over PATH environment variable.

Example

```
EcMasterDemo.exe -f c:\test.xml –r6040 1 1 –t 60000
```

2.4.1.2 IntervalZero RTX

The file EcMasterDemo.rtss has to be executed.The full path and file name of the configuration file has to be given as a command line parameter as well as the appropriate Link Layer.
Example:

```
RTSSrun EcMasterDemo.rtss -f c:\test.xml–i8255x 1 1 –t 60000
```

2.4.1.3 Linux

Step 1: Operating system configuration
See the section Operating system configuration for how to prepare the operating system

Step 2: Starting EcMasterDemo

```
cd /opt/EC-Master-Linux/Bin/Linux/x86
./EcMasterDemo –f MasterENI.xml -i8254x 2 1 -perf
```

2.4.1.4 Microsoft Windows (EcMasterDemo)

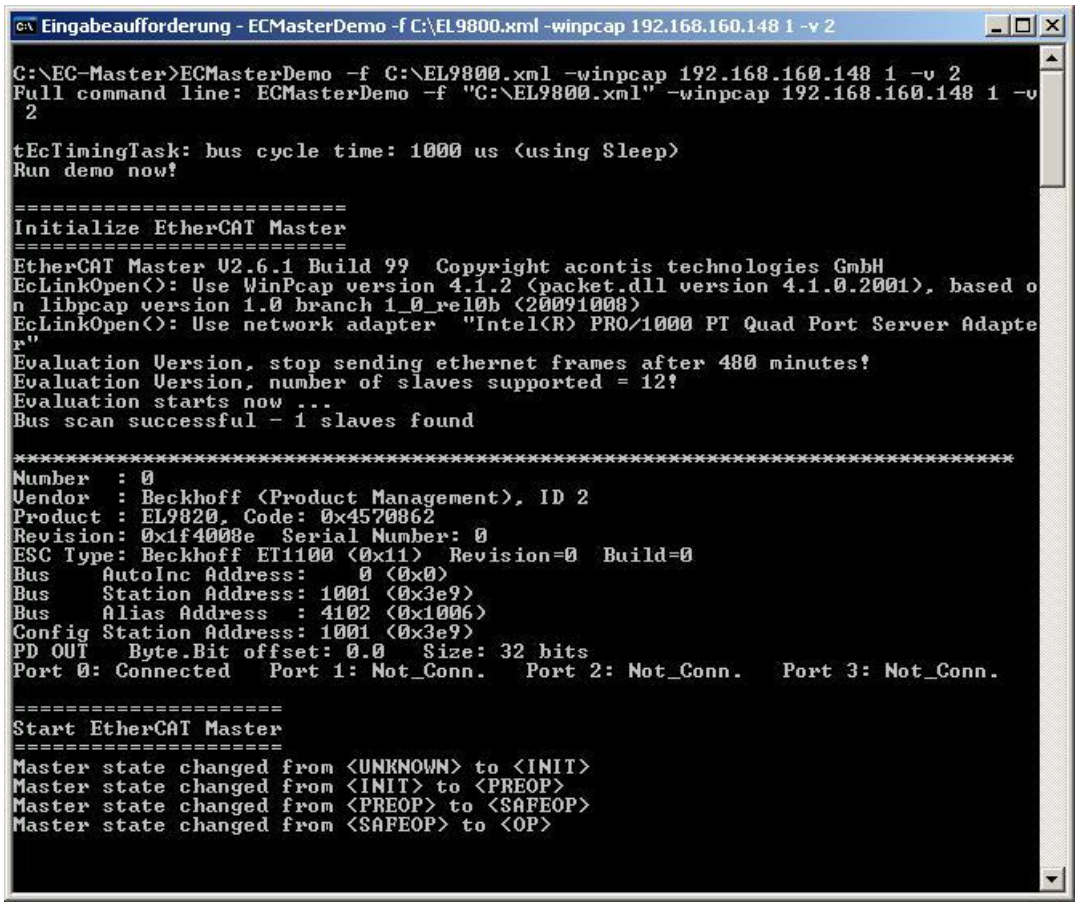
Step 1: Windows configuration
See the section Operating system configuration for how to prepare the operating system

Step 2: Determine the network interface
Using the command line option the network interface card used by the example application can be determined. For example the option `-winpcap 192.168.110.11` will be using the network adapter card with the IP address 192.168.110.11.

Step 3: Connection of the EtherCAT modules
The Evaluation board has to be connected with the target system using an Ethernet switch or a patch cable. You should never connect your local IT infrastructure together with EtherCAT modules at the same switch as the EtherCAT master will send many broadcast packets.
EtherCAT requires a 100Mbit/s connection. If the network adapter card does not support this speed an Ethernet switch has to be used.

Step 4: Copy all of the example application files into one directory
The application EcMasterDemo.exe together with the master stack DLL EcMaster.dll, the link-layer DLL emllPcap.dll and the configuration XML file (for example the file el9800.xml) have to be copied into one directory.

Step 5: Run the example application
The file EcMasterDemo.exe has to be executed. The file name of the configuration file has to be given as a command line parameter as well as the appropriate Link Layer settings.
Example (starting the application with command line parameter `-f C:\EL9800.xml -winpcap 192.168.160.148 1 -v 2`):



2.4.1.5 Microsoft Windows (EcMasterDemoDotNet, .NET)

Step 1: Open the Workspace
Please find the solution including the C#-project for VS2005 at
Workspace\WindowsVS2005\EcMasterDemoDotNet\EcMasterDemoDotNet.sln

Step 2: Compile the application

Step 3: Copy DLLs in the executable's folder
To run the EcMasterDemoDotNet.exe, copy EcMaster.dll, EcMasterDotNet.dll and emllPcap.dll from Bin\Windows\x86 to the executable's folder

Step 4: Run/Debug EcMasterDemoDotNet.exe

2.4.1.6 Microsoft Windows CE

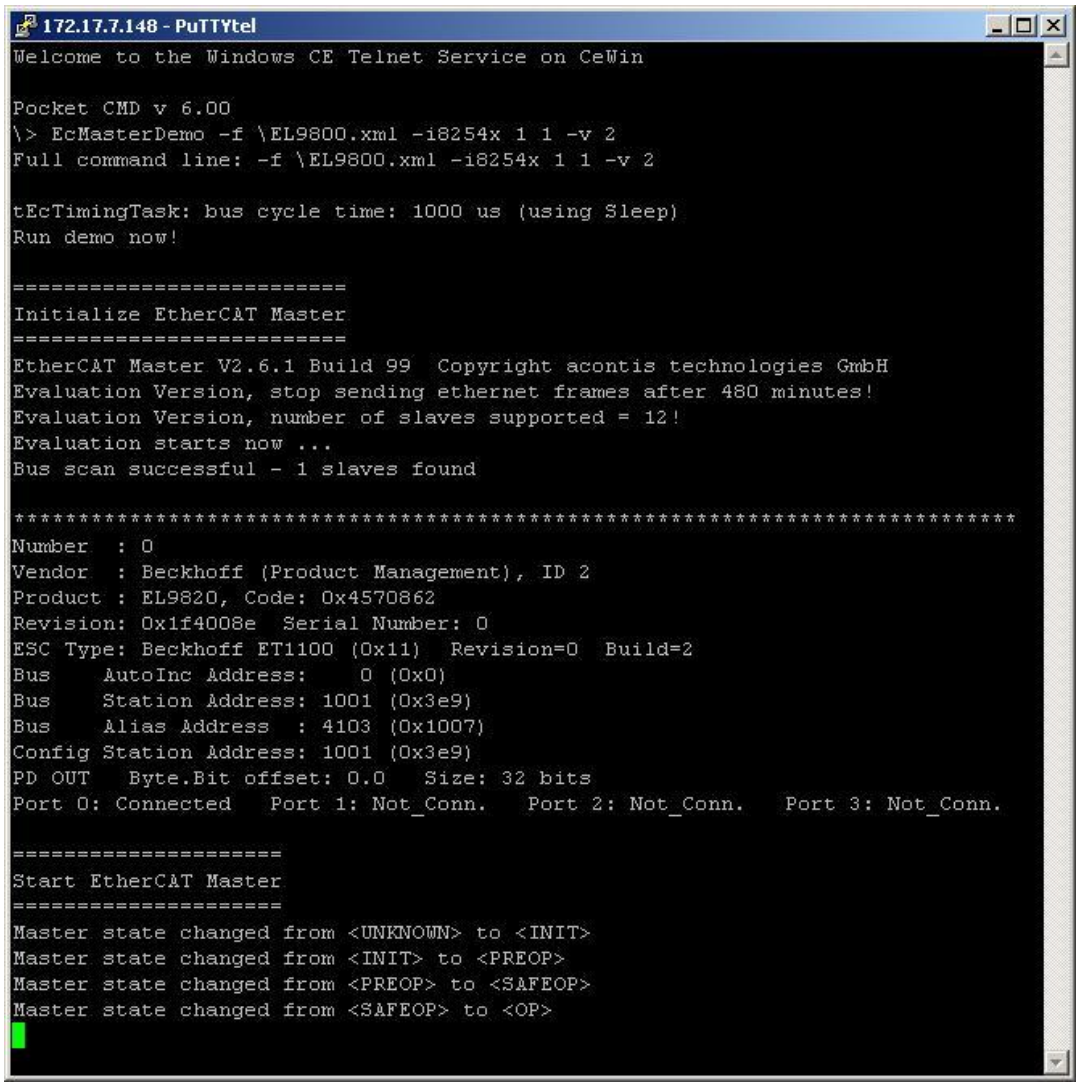
Step 1: Windows CE configuration
See the section Operating system configuration for how to prepare the operating system

Step 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. For example the option `-i8255x 1 1` will dynamically load the optimized Intel Pro/100 Link Layer (the first PCI device instance) and operate in polling mode.

Step 3: Connection of the EtherCAT modules
The Evaluation board has to be connected with the target system using an Ethernet switch or a patch cable. You should never connect your local IT infrastructure together with EtherCAT modules at the same switch as the EtherCAT master will send many broadcast packets.
EtherCAT requires a 100Mbit/s connection. If the network adapter card does not support this speed an Ethernet switch has to be used.

Step 4: Copy the example application on your target
The target has to be started and the application EcMasterDemo.exe together with the master stack DLL EcMaster.dll and the configuration XML file (for example the file el9800.xml) has to be copied onto the target.

Step 5: Run the example application
The file EcMasterDemo.exe has to be executed. The full path and file name of the configuration file has to be given as a command line parameter as well as the appropriate Link Layer.
Example (starting the application on a network share via telnet):



2.4.1.7 QNX Neutrino

Step 1: QNX Neutrino OS configuration
See 2.3.8 “QNX Neutrino” for how to prepare the operating system.
In order to get real-time priority (e.g. 250), see 2.3.8.1 “Thread priority” and also set `JOBS_PRIORITY`. The applications needs root priviledges to increase the priority above 63.

Step 2: Determine the network interface
The network interface card and Link Layer to be used for the demo application can be set via command line. For example the option `-i8255x` will dynamically load the Link Layer for the Intel Pro/100 interface. Possibly you have to un-mount the corresponding network interface driver by using the `umount` command in the QNX shell. For example the command `umount /dev/io-net/en1` unloads the driver for the interface `en1`.

Step 3: Connection of the EtherCAT modules
The Evaluation board has to be connected with the target system using an Ethernet switch or a patch cable. You should never connect your local IT infrastructure together with EtherCAT modules at the same switch as the EtherCAT master will send many broadcast packets.
EtherCAT requires a 100Mbit/s connection. If the network adapter card does not support this speed an Ethernet switch has to be used.

Step 4: Copy the example application on your target
The QNX target has to be started and the application EcMasterDemo and the configuration XML file (for example the file el9800.xml) has to be copied onto the target.

Step 5: Copy the Link Layer files (shared object files) to your target

After copying the application you need to download a Link Layer library (e.g. emlll8255x.so) which contains hardware support for your NIC. The Link Layer library should be located in /lib/dll or any other location that is specified by the _CS_LIBPATH configuration-string.

Step 6: Run the example application

The EcMasterDemo.exe and the libraries have to be executable. The file EcMasterDemo.exe has to be executed. The full path and file name of the configuration file has to be given as a command line parameter as well as the appropriate Link Layer.
Example:

```
./EcMasterDemo –f /tmp/test.xml –i8255x 1 1 –t 60000
```

2.4.1.8 Renesas R-IN32M3

2.4.1.8.1 Prerequisites, basic settings:

- Hardware:
  - R-IN32M3-EC Evaluation Board,
  - adviceLUNA Emulator
- Software:
  - microVIEW-PLUS debugger,
  - GNU compiler (Sourcery G++ Lite for ARM EABI)
- Verify TCP/IP evaluation sample from Renesas works fine. Please download from official Renesas site following files: r-in32m3_tcpip_evaluation.zip and r-in32m3_samplesoft.zip.

2.4.1.8.2 How to create the demo applications

- Create ENI file for EtherCAT configuration.
- Convert eni file to the C file with array. You can use xxd.exe with parameters: xxd.exe -i eni.xml Master.c or some other tool which converts binary file to the C array. Replace MasterENI.c file with generated one. File should be manually modified to look like:

```
unsigned char MasterENI_xml_data[] = {
...
};
unsigned int MasterENI_xml_data_size = ???;
```

- Import project Workspace\RIN32M3\EcMasterDemo into Eclipse IDE.
- Hardcoded parameters for the demo can be changed using DEMO_PARAMETERS definition. See “ECMasterDemo Command line options” for details.

2.4.1.8.3 How to run the EC-Master demo application

- Upload Workspace\RIN32M3\EcMasterDemo\Release\EcMasterDemo.bin with debugger and run

2.4.1.9 RTAI

Step 1: Load required RTAI Kernel modules

Example:

```
> insmod /usr/realtime/modules/rtai_hal.ko
> insmod /usr/realtime/modules/rtai_sched.ko
> insmod /usr/realtime/modules/rtai_shm.ko
```

Step 2: Make sure that the native Linux driver for your EtherCAT network card is <u>not</u> loaded

Note: SockRaw cannot be used with RTAI.

See 2.4.1.3 “Linux” for how to unload drivers that may use network adapter.

Step 3: Copy the ENI file on your target

The location of the ENI file is `/opt/EC-Master-RTAI/Examples/EcMasterDemoDc/RTAI/MasterENI.xml` .

Step 4: Create the EcMasterDemoDc RTAI Kernel module

```
> cd /opt/EC-Master-RTAI/Examples/EcMasterDemoDc/RTAI
> make -f Kbuild
```

Step 5: Load the EcMasterDemoDc RTAI Kernel module

```
> insmod KcCMasterDemoDc.ko
```

Step 6: Verify that the EcMasterDemoDc RTAI Kernel module is loaded succesfully

Give the EcMasterDemoDc RTAI module some seconds to start.
Using `dmesg` you should see that the EC-Master has successfully started:

```
> dmesg
...
[ 3593.654951] Master state changed from <SAFEOP> to <OP>
```

2.4.1.10 RTEMS

Step 1: Operating system configuration
Build the RTEMS operating system with least following configuration parameters:

```
--target=i386-rtems4.11 --enable-rtemsbsp=pc386 --enable-cxx
```

Step 2: Compile EcMasterDemo
To load ENI files from disk and support logging you may suit the function `rtemsMountFilesystems()` to your needs.

Step 3: Starting the demo
Boot the file EcMasterDemo.exe.The full path and file name of the configuration file has to be given as a command line parameter as well as the appropriate Link Layer.

Example

```
EcMasterDemo.exe -f /mnt/hda1/eni.xml -t 60000 -i8254x 1 1
```

2.4.1.11 TI AM335x SYSBIOS Industrial SDK (Beaglebone)

2.4.1.11.1 Prerequisites, basic settings:

- Install Starterware
- Install AM335x SYSBIOS Industrial SDK
- New environment variable IA_SDK_HOME has to be set to the root directory of Industrial SDK. For example: IA_SDK_HOME=C:\ti\am335x_sysbios_ind_sdk_1.1.0.3\sdk
- Enable CPSW for ICE V2 board (Jumpers J18 and J19 - see ISDK User guide)

2.4.1.11.2 How to create the demo applications

- Create ENI file for EtherCAT configuration.
- Convert eni file to the C file with array. You can use xxd.exe with parameters: xxd.exe -i eni.xml Master.c or some other tool which converts binary file to the C array. Replace MasterENI.c file with generated one.
- Import project Workspace\SYSBIOS\EcMasterDemo into CCS Studio Workspace Output files:
  - Workspace\SYSBIOS\EcMasterDemo\Release\EcMasterDemo.bin
    --> EC-Master demo application (downloadable version)
  - Workspace\SYSBIOS\EcMasterDemo\Release\EcMasterDemo_SD.bin
    --> EC-Master demo application (bootable binary version)
- Hardcoded parameters for the demo can be changed using DEMO_PARAMETERS definition. See “ECMasterDemo Command line options” for details.

2.4.1.11.3 How to run the EC-Master demo application

- create a bootable SD card using the appropriate TI tool
- copy Workspace\SYSBIOS\EcMasterDemo\Release\EcMasterDemo_SD.bin onto the SD card and rename it to “app”
- connect a USB cable with the board and set up a terminal connection with the PC
- turn on the board, demo will be started automaticall

2.4.1.12 TI Starterware (Beaglebone)

2.4.1.12.1 Prerequisites, basic settings:

- Install Starterware
- New environment variable STARTERWARE_ROOT has to be set to the root directory of Starterware. For example: C:\ti\AM335X_StarterWare_02_00_01_01

2.4.1.12.2 How to create the demo applications

- Rebuild following projects with VFPv3 support and NEON:
  ```
  ${STARTERWARE_ROOT}\build\armv7a\cgt_ccs\am335x\beaglebone\platform
  ${STARTERWARE_ROOT}\build\armv7a\cgt_ccs\am335x\drivers
  ${STARTERWARE_ROOT}\build\armv7a\cgt_ccs\am335x\system_config
  ${STARTERWARE_ROOT}\build\armv7a\cgt_ccs\mmcsdlib
  ${STARTERWARE_ROOT}\build\armv7a\cgt_ccs\utils
  ```
- Import the following 2 projects into CCS Studio Workspace:
  ```
  Examples\EcMasterDemo\Starterware
  Examples\EcMasterDemoMotion\Starterware
  ```

Output files:

Examples\EcMasterDemo\Starterware\Release\APP
--> EC-Master demo application (bootable binary version)
Examples\EcMasterDemo\Starterware\Debug\EcMasterDemo.out
--> EC-Master demo application (downloadable debug version)
Examples\EcMasterDemoMotion\Starterware\Release\APP
--> EC-Master demo application (bootable binary version)
Examples\EcMasterDemoMotion\Starterware\Debug\EcMasterDemoMotion.out
--> EC-Master demo application (downloadable debug version)

2.4.1.12.3 How to run the EC-Master demo application

- create a bootable SD card using the appropriate TI tool
- copy Examples\EcMasterDemo\Starterware\Release\APP onto the SD card

- create an appropriate ENI file (8.3 file format) and copy it into the root directory of the SD card
- boot the card
- connect a USB cable with the board and set up a terminal connection with the PC
- the command line supports some basic functions like 'ls', 'cat' etc. based on the Starterware examples
- a new command 'ec' was added.
    - first parameter: ENI file name
    - second parameter: cycle time in usec (fastest cycle time = 100 usec)
- if you want to run the demo with the ENI file named 'myeni.xml' and 500 usec cycle time:
    - ec myeni.xml 500
- the demo can be stopped using the <ESC> or <CTRL-C> key
- two log files are generated: ma0_0.log (all messages) and er0_0.log (error messages only)

2.4.1.12.4 How to run the EC-Master motion demo application

- create a bootable SD card using the appropriate TI tool
- copy Examples\EcMasterDemoMotion\Starterware\Release\APP onto the SD card
- create an appropriate ENI file (8.3 file format) and copy it into the root directory of the SD card
    - the DC configuration has to be done appropriately, please see the EtherCAT general documenation and EC-Master manuals for details
- create an appropriate motion demo configuration file and copy it into the root directory of the SD card
    - see example in Examples\EcMasterDemoMotion\Config\DemoConfig.xml
    - see additional info in: Examples\EcMasterDemoMotion\readme.txt
- boot the card
- connect a USB cable with the board and set up a terminal connection with the PC
- the command line supports some basic functions like 'ls', 'cat' etc. based on the Starterware examples
- a new command 'md' was added.
    - parameter: motion config file name (default filename: md.xml)
- the demo can be stopped using the <ESC> or <CTRL-C> key
- two log files are generated: ma0_0.log (all messages) and er0_0.log (error messages only)
- two csv files are generated: dc0_0.log (DC information) and mot0_0.log (motion information)

2.4.1.13 Windriver VxWorks

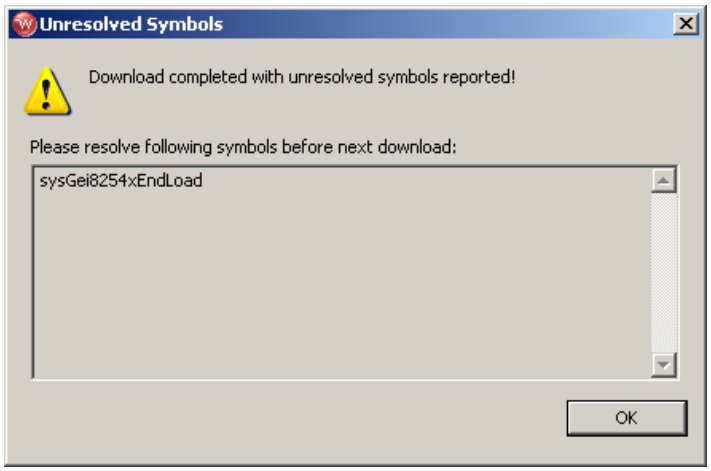
Step 1: VxWorks OS configuration
See section VxWorks, SNARF Link Layer or KUKA VxWin, SNARF Link Layer for how to prepare the operating system.

Step 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.

Step 3: Connection of the EtherCAT slaves
The slaves have to be connected with the VxWorks system using an Ethernet switch or a patch cable.
You should never connect your local IT infrastructure together with EtherCAT modules at the same switch as the EtherCAT 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.

Step 4: 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.

There may be unresolved symbols in case of a missing Link Layer. For example if the Intel Pro/1000 network driver is not included into the image the following dialog will be shown:



If you are using the PRO/100 Link Layer you can ignore this message.

Step 5: Download a Link Layer module
After downloading your application you need to download a Link Layer module (e.g. emlll8255x.out) which in case contains hardware support for your NIC. By default the Link Layers emllSnarfGpp / emllSnarfPid are contained with the binary delivery.

Step 6: Set up a FTP server connection on your host
The demo application needs to load a XML file (MasterENI.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.
You may check the FTP connection using the “ls” command. The file MasterENI.xml will have to be accessed using the default directory.

Step 7: Check for exclusive hardware access
Be sure that the network adapter instance dedicated to EtherCAT is not controlled by a VxWorks driver, this can be verified using:
> muxShow

If it is needed, first unload the driver using:
(e.g. first instance of the Intel Pro/100):
> muxDevUnload “fei”, 1

(e.g. second instance of the Intel Pro/1000):
> muxDevUnload “gei”, 2

(e.g. first instance of the Realtek 8139):
> muxDevUnload “rtl”, 1

(e.g. first instance of the Realtek 8169):
> muxDevUnload “rtg”, 1

(e.g. first instance of the FEC on Freescale iMX platform):
> muxDevUnload “motfec”, 1

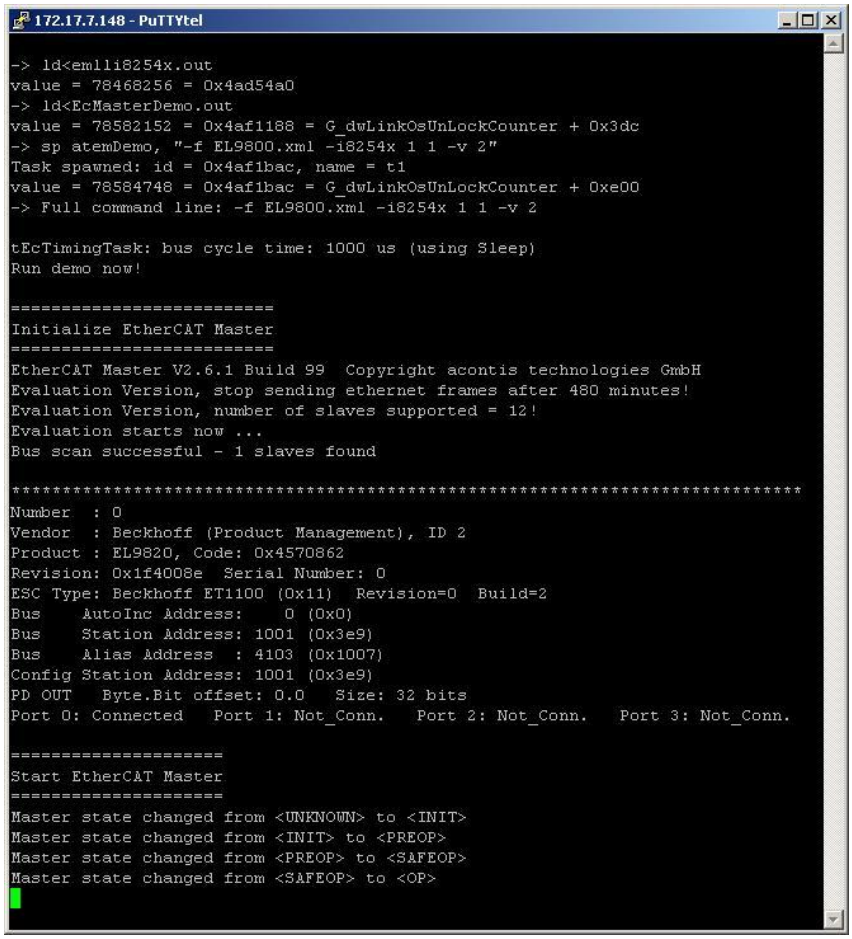
(e.g. first instance of the ETSEC on Freescale PPC platform):
> muxDevUnload “motetsec”, 1

Step 8: Run the example application
The downloadable module EcMasterDemo.out has to be executed. The configuration file EL9800.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<EcMasterDemo.out
sp atemDemo,"-f EL9800.xml -i8254x 1 1 -v 2"
```

Example:



2.4.1.14 Xenomai

Step 1: Prepare system

Prepare the system to run EcMasterDemo on Linux as described in chapter 2.4.1.3, “Linux”.

Step 2: Compile EcMasterDemo

As a starting point there is the Eclipse project for EcMasterDemo for Xenomai located at Workspace/Xenomai/EcMasterDemo.
Ensure ARCH is set accordingly (export ARCH=x86) when compiling using Eclipse!

Step 3: Run using GDB

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

```
export LD_LIBRARY_PATH=../../Bin/Xenomai/x86:/usr/xenomai/lib:.
gdb --args ./EcMasterDemo -I8254x 2 1 -f eni.xml -v 3
[…]
(gdb) handle SIGXCPU nostop noprint nopass
(gdb) run
```

2.4.2 Command line parameters

EcMasterDemo <Link Layer> [-f configFileName] [-t time] [-b time] [-v level] [-a affinity] [-perf] [-auxclk period] [-sp [port]]

e.g. EcMasterDemo -winpcap 192.168.157.2 1 -f MasterENI.xml -t 0 -v 3

The parameters are as follows:

- ***-f <configFileName>***
  Path to ENI file. **Note**: On Windows CE absolute file paths are needed.

- ***-t <time>***
  <time>: Time in msec, 0 = forever (default = 120000)
  Running duration in msec. When the time expires the demo application exits completely.

- ***-b <cycle time>***
  <cycle time>: Bus cycle time in µsec
  Specifies the bus cycle time. Defaults to 1000µs (1ms).

- ***-v <level>***
  <level>: Verbosity level: 0=off (default), 1..n=more messages
  The verbosity level specifies how much console output messages will be generated by the demo application. A high verbosity level leads to more messages.

- ***-a <affinity>***
  <affinity>: 0 = first CPU, 1 = second, ...
  The CPU affinity specifies which CPU the demo application ought to use.

- ***-perf***
  Enable max. and average time measurement in µs for all EtherCAT jobs (e.g. ProcessAllRxFrames).

- ***-auxclk <period>***
  <period>: Clock period in µs (if supported by Operating System).

- ***-sp [port]***
  If platform has support for IP Sockets, this commandline option enables the Remote API Server to be started with the EcMasterDemo. 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.
  This option is included for attaching the EC-Lyser Application to the running master.

- ***-log prefix***
  Use given file name prefix for log files.

2.4.2.1 Link Layer

Using one of the following Link Layer options, the demo application will dynamically load the network driver for the specified network adapter card (e.g. –i8254x 1 1 for the Intel Pro/1000 network card). The EtherCAT master then will use the appropirate network driver to access the Ethernet adapter.

- ***-i8254x <instance> <mode>***
  <*instance*>: Device instance 1=first, 2=second
  <*mode*>: Mode 0 = Interrupt mode, 1= Polling mode
  Hardware: Intel Pro/1000 network adapter card

- ***-i8255x <instance> <mode>***
  <*instance*>: Device instance 1=first, 2=second
  <*mode*>: Mode 0 = Interrupt mode, 1= Polling mode
  Hardware: Intel Pro/100 network adapter card

- ***-rtl8139 <instance> <mode>***
  <*instance*>: Device instance 1=first, 2=second
  <*mode*>: Mode 0 = Interrupt mode, 1= Polling mode
  Hardware: Realtek RTL8139

- ***-rtl8169 <instance> <mode>***
  <*instance*>: Device instance 1=first, 2=second
  <*mode*>: Mode 0 = Interrupt mode, 1= Polling mode
  Hardware: Realtek RTL8168 / RTL8169 / RTL8111

- ***-winpcap <ipAddress> <mode>***
  <ipAddress>: IP address of network adapter card, e.g. 192.168.157.2
  <*mode*>: Mode 0 = Interrupt mode, 1= Polling mode
  Hardware: Hardware independent.

- ***-snarf <adapterName>***
  <adapterName>: Adapter name, e.g. fei0
  Hardware: Hardware independent
  This parameter is only available for VxWorks

- ***-etherlib <adapterName>***
  <adapterName>: Adapter name, e.g. fei0
  Hardware: Hardware independent
  This parameter is only available for VxWorks (5.4-5.5GPP)

- ***-hnx <instance>***
  <instance>: Device instance (0=first), e.g. 0
  Hardware: Hilscher NetX XPEC Builtin PHY raw driver

- ***-sockraw <device>***
  <device>: Network device, e.g. eth1
  Hardware: Hardware independent
  This parameter is only available for Linux.

- ***-hnx***
  Hardware: Hilscher NetX XPEC Builtin PHY raw driver

- ***-ndisuio*** (Windows CE only)
  Using this option the example application will dynamically load the NDISUIO driver for the specified network adapter card (e.g. –ndisuio PCI\RTL81391). The EtherCAT master then will use NDISUIO to access the Ethernet adapter.

- ***-fslfec <instance> <mode>***
  Hardware: Freescale FEC/ENET
  <Instance>: Device instance 1=first, 2=second
  <Mode>: Mode 0 = Interrupt mode, 1= Polling mode
  <RefBoard> custom or mars or sabrelite or imx28evk or topaz
  if custom
  < FecType> imx25 or imx28 or imx53 or imx6
  <PhyInterface> fixed or mii or rmii or gmii or sgmii or rgmii

- ***-ccat <instance> <mode>***
  <*instance*>: Device instance 1=first, 2=second
  <*mode*>: Mode 0 = Interrupt mode, 1= Polling mode
  Hardware: Beckhoff CCAT

- ***-gem <instance> <mode>***
  Hardware: Xilinx Zynq-7000 (GEM)
  <*instance*>: Device instance 1=first, 2=second
  <*mode*>: Mode 0 = Interrupt mode, 1= Polling mode
  <*PhyAddr*>: Optional parameter: PHY address (0..31)
  <*PhyConMode*>:Optional parameter: PHY connection mode MIO (0) or EMIO (1)

- ***-rin32m3***
  Hardware: Renesas R-IN32M3-EC

- ***-dw3504 <instance> <mode> <PhyAddress>***
  Hardware: Synopsys DesignWare 3504-0 Universal 10/100/1000 Ethernet MAC (DW3504)
  <*instance*>: Device instance 1=first, 2=second
  <*mode*>: Mode 0 = Interrupt mode, 1= Polling mode
  <*phyaddress*>: 0 .. 31

- ***-r6040 <instance> <mode>***
  Hardware: RDC R6040
  <*instance*>: Device instance 1=first, 2=second
  <*mode*>: Mode 0 = Interrupt mode (currently not supprted), 1= Polling mode

2.5 Compiling the EcMasterDemo

2.5.1 EtherCAT Master Software Development Kit (SDK)

The EtherCAT master development kit is needed to write applications based on the master stack.
The master stack is shipped as a library which is linked together with the application.

The following components are supplied together with an SDK:

- Documentation (….\Doc)
- EtherCAT Software Development Kit (…\SDK) containing libraries and header files to build C/C++-applications.
  - ….\Bin: Executables containing the master stack
  - ….\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 (e.g. Windows CE registry files)
  - ….\Sources\Common: Shared .cpp-files
- One or multiple example applications (…\Examples) using a predefined EtherCAT-configuration. It is easily adaptable to different configurations using an appropriate EtherCAT configuration XML file.

2.5.2 General

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

2.5.2.1 Include search path

The header files are located in the following two directories:

a) `<InstallPath>\SDK\INC\<OS>\<ARCH>` (where <OS> is a placeholder for the operating system and <ARCH> for the architecture if different archtitectures are supported)
b) `<InstallPath>\SDK\INC`
c) `<InstallPath>\Sources\Common`

2.5.2.2 Preprocessor macro

The demo applicatons are the same for all operating systems. The appropriate pre-processor macro has to be set for the operating system (for example `VXWORKS`, `LINUX`, `__INTIME__`, `__TKERNEL`,…).
For big endian systems the macro `EC_BIG_ENDIAN` has to be defined.

2.5.2.3 Libraries

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

2.5.3 OS Compiler settings

2.5.3.1 DOS GO32-V2

The following settings are necessary to build the example application for DOS GO32-V2.

- Preprocessor definitions:
  ```
  NO_OS
  ```

- Extra include paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/SDK/INC/GO32-V2
  <InstallPath>/SDK/INC
  <InstallPath>/Sources/Common
  <InstallPath>/ Workspace/GO32-V2/GO32-V2_common/include
  ```

- Extra source paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/Sources/Common/EcTimer.cpp
  ```

- Extra library paths to the main EtherCAT components:
  ```
  <InstallPath>/SDK/LIB/GO32-V2
  ```

  Extra libraries
  ```
  libEcMaster.a libemllR6040.a
  ```

2.5.3.1.1 How to create the demo applications

- The DJGPP toolchain has to be installed and configured.
- Start the RHIDE IDE and open the project EcMasterDemo.gpr
- Build the project: menu “Compile” -> “Build all”
- Alternatively the project can be build from the command line like the following:

  ```
  make -f EcMasterDemo.mak
  ```

Important! To run the demo application it is nessesery to quit the RHIDE IDE and start the application as written above from command line.

2.5.3.2 Greenhills INTEGRITY

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

- Extra include paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/SDK/INC/INTEGRITY
  <InstallPath>/SDK/INC
  <InstallPath>/Sources/Common
  ```

- Extra source paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/Sources/Common/EcTimer.cpp
  ```

- Extra library paths to the main EtherCAT components:
  ```
  <InstallPath>/SDK/LIB/INTEGRITY/x86
  ```

- Extra libraries (in this order)
  ```
  AtemRasSrv EcMaster
  ```

2.5.3.3 Linux

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

- Possible ARCHs:

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

armv4t-eabi and armv6-vfp-eabihf are incompatible with each other. A potentially armv6-vfp-eabihf-compatible system returns success on “readelf -A /proc/self/exe | grep Tag_ABI_VFP_args”. If “readelf”

isn't available on the target, the matching ARM version can be figured out by trying to run EcMasterDemo.

- Extra include paths:
  `<InstallPath>/Examples/EcMasterDemo`
  `<InstallPath>/SDK/INC/Linux`
  `<InstallPath>/SDK/INC`
  `<InstallPath>/Sources/Common`

- Extra source paths:
  `<InstallPath>/Examples/EcMasterDemo`
  `<InstallPath>/Sources/Common/EcTimer.cpp`

- Extra library paths to the main EtherCAT components (replace "x86" according to ARCH):
  `<InstallPath>/SDK/LIB/Linux/x86`

- Extra libraries (in this order)
  `AtemRasSrv EcMaster pthread dl rt`

Xilinx Zynq-7000 needs extra CFLAGS "-mcpu=cortex-a9 -march=armv7-a".

2.5.3.4 Microsoft Windows CE

The following settings are necessary to build the example application (e.g. *Win32 Smart Device Project Console Application*) for Windows CE:

- Preprocessor definitions:
  `WIN32, UNDER_CE` (implicitely defined by VC++ 4.0) , `_CRT_SECURE_NO_DEPRECATE`

- Include path:
  `<InstallPath>/SDK/INC/WinCE;<InstallPath>/SDK/INC;<InstallPath>/Sources/Common`

- Don't "Treat wchar_t as Built-in Type"

- Library path of the main EtherCAT components:
  `<InstallPath>/SDK/LIB`

- Libraries:
  `"coredll.lib", "corelibc.lib", "EcMaster.lib" and "AtemRasSrv.lib"`

- Entry Point:
  `mainWCRTStartup`

2.5.3.5 Microsoft Windows

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

- Preprocessor definitions:
  `WIN32` (implicitely defined by VC 6.0)

- Library path of the main EtherCAT components:
  `<InstallPath>/SDK/LIB/Windows/x86`

- Include path:
  `<InstallPath>/SDK/INC/Windows`
  `<InstallPath>/SDK/INC`
  `<InstallPath>/Sources/Common`

2.5.3.6 QNX Neutrino

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

- Extra include paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/SDK/INC/QNX6
  <InstallPath>/SDK/INC
  <InstallPath>/Sources/Common
  ```

- Extra source paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/Sources/Common/EcTimer.cpp
  ```

- Extra library paths to the main EtherCAT components:
  ```
  <InstallPath>/SDK/LIB/QNX6/x86
  ```

  Extra libraries
  ```
  AtemRasSrv EcMaster socket
  ```

2.5.3.7 RTEMS

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

- Possible ARCHs: x86 (pc386)

- Extra include paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/SDK/INC/RTEMS
  <InstallPath>/SDK/INC
  <InstallPath>/Sources/Common
  ```

- Extra source paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/Sources/Common/EcTimer.cpp
  ```

- Extra library paths to the main EtherCAT components (replace “x86” according to ARCH):
  ```
  <InstallPath>/SDK/LIB/RTEMS/x86
  ```

- Extra libraries (in this order)
  ```
  libEcMaster.a libemllI8254x.a libemllRTL8169.a libemllCCAT.a
  ```

2.5.3.8 Windriver VxWorks

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

- Preprocessor definitions:
  ```
  -DVXWORKS
  ```
- Library path of the main EtherCAT components (general purpose platform):
  ```
  <InstallPath>/SDK/LIB/VxWorks70/PENTIUM4_SMPgnu
  ```
- Include paths:
  ```
  -I<InstallPath>/SDK/INC/VxWorks
  -I<InstallPath>/SDK/INC
  -I<InstallPath>/Sources/Common
  ```

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

2.5.3.8.1 VxWorks settings Power PC (PPC)

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

- Preprocessor definitions:
  ```
  -DVXWORKS
  -DEC_BIG_ENDIAN
  ```
- Library path of the main EtherCAT components (general purpose platform):
  ```
  <InstallPath>/SDK/LIB/VxWorks70/PPCE500V2_SMPgnu
  ```
- Include paths:
  ```
  -I<InstallPath>/SDK/INC/VxWorks
  -I<InstallPath>/SDK/INC
  -I<InstallPath>/Sources/Common
  ```

GNU/PowerPC: you may have to set the –mlongcall compiler option to avoid relocation offset errors when downloading *.out files.

2.5.3.9 Xenomai

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

- Extra include paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/SDK/INC/Xenomai
  <InstallPath>/SDK/INC
  <InstallPath>/Sources/Common
  ```
- Extra source paths:
  ```
  <InstallPath>/Examples/EcMasterDemo
  <InstallPath>/Sources/Common/EcTimer.cpp
  <InstallPath>/Sources/Sources/OsLayer/Xenomai/EcOs.cpp
  ```
- Extra library paths to the main EtherCAT components:
  ```
  <InstallPath>/SDK/LIB/Xenomai/x86
  ```
- Extra libraries (in this order)
  ```
  AtemRasSrv EcMaster pthread dl rt native xenomai
  ```

2.5.4 Link Layer selection and initialization

The different Link Layer modules are selected and parametered by a common structure (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 the EtherCAT master stack with the call of ecatInitMaster.

2.5.4.1 ecatInitMaster

Initializes the EtherCAT master stack.

```
EC_T_DWORD ecatInitMaster(
    EC_T_INIT_MASTER_PARMS*      pParms
);
```

Parameters

pParms

[in] Pointer to parameter definitions

Return

EC_E_NOERROR or error code

Comment

The EtherCAT master stack will be initialized by calling this function. This function has to be called prior to calling any other functions of the EtherCAT master.

```
typedef struct _EC_T_INIT_MASTER_PARMS{
    :                   :       :       :
    :                   :       :       :
    :                   :       :       :
    :                   :       :       :
    EC_T_LINK_PARMS*    pLinkParms;
    :                   :       :       :
    :                   :       :       :
} EC_T_INIT_MASTER_PARMS;
```

Description

pLinkParms

[in] Pointer to the common Link Layer parameters. This parameter set will be used by the master to load the appropriate Link Layer dynamically if present and contains the Link Layer specific parameter set (see below).

EC_T_LINK_PARMS

EtherCAT Link Layer common parameters.

```
typedef struct _ EC_T_LINK_PARMS
{
    EC_T_DWORD          dwSignature
    EC_T_DWORD          dwSize;
    EC_T_CHAR           szDriverIdent[MAX_DRIVER_IDENT_LEN];
    EC_T_DWORD          dwInstance
    EC_T_LINKMODE       eLinkMode;
    EC_T_DWORD          dwIstPriority
} EC_T_LINK_PARMS;
```

Description

dwSignature

[in] Signature of the adapter specific structure containing the EC_T_LINK_PARMS structure

dwSize

[in] Size of the adapter specific structure containing the EC_T_LINK_PARMS structure

szDriverIdent

[in] Name of Link Layer module (driver identification) this name is used for Link Layer Selection

dwInstance

[in] Instance of the adapter

eLinkMode

[in] Mode of operation: EcLinkMode_POLLING or EcLinkMode_INTERRUPT

dwIstPriority

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

2.5.4.2 Link Layer specific parameter set

The Link Layer specific parameter set extends the common Link Layer parameters, which is described here.

Multi-Platform Link Layers may be available for several different operating systems. The initialization is widely generic.

2.5.4.2.1 CPSW EtherCAT driver

The parameters to the CPSW Link Layer are setup-specific.
The function “CreateLinkParmsFromCmdLineCPSW” in selectLinkLayer.cpp demonstrates how to initialize the Link Layer instance.
Because the link status cannot be read quickly from a register of the adapter, it will not be automatically refreshed like by the other Link Layers.
To refresh the link status, EC_LINKIOCTL_UPDATE_LINKSTATUS should be called from another task:

```
for (;;)
{
    ecatIoControl(EC_IOCTL_LINKLAYER_MAIN + EC_LINKIOCTL_UPDATE_LINKSTATUS, EC_NULL);
    OsSleep(1000);
}
```

2.5.4.2.2 DW3504 EtherCAT driver

The parameters to the Synopsys DesignWare 3504-0 Universal 10/100/1000 Ethernet MAC (DW3504) Link Layer are setup-specific.
The function “CreateLinkParmsFromCmdLineDW3504” in selectLinkLayer.cpp demonstrates how to initialize the Link Layer instance.

```
typedef struct _EC_T_LINK_PARMS_DW3504
{
    EC_T_LINK_PARMS     linkParms;
    EC_T_DWORD          dwPhyAddr;
    EC_T_DWORD          dwRegisterBasePhys
} EC_T_LINK_PARMS_DW3504;
```

Description

linkParms

Common parameters

dwPhyAddr

[in] PHY address (0 .. 31) on MII bus

dwRegisterBasePhys

[in] Physical base address of register block (8k)

2.5.4.2.3 FreeScale FEC EtherCAT driver

```
/* Link parameters */
typedef struct _EC_T_LINK_PARMS_FSLFEC
{
    EC_T_DWORD              dwType;
    EC_T_DWORD              dwUnit;
    EC_T_DWORD              dwIstPriority;
    PF_DOINT_HDL            pfDoIntHandling;
    EC_T_DWORD              dwRxBuffers;
    EC_T_DWORD              dwTxBuffers;
    PF_emllISRCallback      pfIsrCallback;
    EC_T_PVOID              pvIsrCBContext;

    EC_T_FEC_TYPE           eFecType;
    EC_T_PHY_INTERFACE      ePhyInterface;
} EC_T_LINK_PARMS_FSLFEC;
```

Description

dwType
: Validation pattern. The value EC_LINK_TYPE_FSLFEC has to be assigned to this parameter.

dwUnit
: Network adapter instance, first is identified with 1. The second with 2 etc.

dwIstPriority
: Obsolete.

eLinkMode
: Obsolete.

pfDoIntHandling
: Obsolete.

dwRxBuffers
: Amount of allocated receive buffers in queue.

dwTxBuffers
: Amount of allocated transmit buffers in queue (each of which 1536 bytes of size).

eFecType
: eFEC_IMX25 or
: eFEC_IMX28 or
: eFEC_IMX53 or
: eFEC_IMX6

ePhyInterface
: ePHY_FIXED_LINK or
: ePHY_MII or
: ePHY_RMII or
: ePHY_GMII or
: ePHY_SGMII or
: ePHY_RGMII

Comment

The demo applications support out of the box some reference boards where the combinations eFecType/ePhyInterface are already defined.

2.5.4.2.4 FreeScale TSEC/eTSEC EtherCAT driver

The parameters to the ETSEC Link Layer are much more setup-specific than e.g. the Intel Pro/1000. The function “CreateLinkParmsFromCmdLineETSEC” in selectLinkLayer.cpp demonstrates how to initialize the Link Layer instance.

```
typedef struct _EC_T_LINK_PARMS_ETSEC
{
   EC_T_LINK_PARMS linkParms;
   EC_T_DWORD      dwRegisterBase;
   EC_T_DWORD      dwLocalMdioBase;
   EC_T_DWORD      dwPhyMdioBase;
```

```
    EC_T_DWORD          dwPhyAddr;
    EC_T_DWORD          dwTbiPhyAddr;
    EC_T_DWORD          dwFixedLinkVal;
    EC_T_BYTE           abyStationAddress[6];
    EC_T_BYTE           byReserved[2];
    EC_T_VOID*          oMiiBusMtx;
    EC_T_DWORD          dwRxInterrupt;
} EC_T_LINK_PARMS_ETSEC;
```

Description

linkParms
    Common parameters

dwRegisterBase
    [in] Physical base address of register block (4k)

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

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

dwPhyAddr
    [in] PHY address on MII bus. ETSEC_FIXED_LINK (0xFFFFFFFF) if fixed link configuration.

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

dwFixedLinkVal
    [in] Only evaluated if dwPhyAddr == ETSEC_FIXED_LINK. Set to ETSEC_LINKFLAG_100baseT_Full or ETSEC_LINKFLAG_LINKOK.

abyStationAddress
    [in] MAC station address

oMiiBusMtx
    [in] This mutex protect the access to the (shared) MII bus. Set to 0 if mutex shouldn't be used. The MII bus is shared between eTSEC instances. So this mutex should be created once and assigned here for all Link Layer instances.

dwRxInterrupt
    [in] Receive interrupt number (IRQ)

2.5.4.2.5 Intel Pro/100 EtherCAT driver emIII8255x

```
typedef struct _EC_T_LINK_PARMS_I8255X
{
    EC_T_LINK_PARMS linkParms;
} EC_T_LINK_PARMS_I8255X;
```

Description

linkParms
Common parameters

Comment

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

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

For some operating systems it is possible to alternatively address the emIII8255x using its PCI address. The format of dwInstanceId using PCI bus address is "`0x01bbddff`" with "`bb`" Bus Number, "`dd`" Device Number, "`ff`" Function Number. E.g. "`0000:00:19.0`“ is dwInstanceId `0x01001900`. See chapter 2.5.4.2.6, “Intel Pro/1000 EtherCAT driver emIII8254x” for an example of how to show the PCI address on Linux.

2.5.4.2.6 Intel Pro/1000 EtherCAT driver emIII8254x

```
typedef struct _EC_T_LINK_PARMS_I8254x
{
    EC_T_LINK_PARMS    linkParms;
    EC_T_WORD          dwRxBufferCnt;
    EC_T_WORD          dwRxBufferSize;
    EC_T_WORD          dwTxBufferCnt;
    EC_T_WORD          dwTxBufferSize;
    EC_T_BOOL          bDisableLocks
} EC_T_LINK_PARMS_I8254x;
```

Description

linkParms
Common parameters
dwRxBufferCnt
Amount of allocated receive buffers in queue.
dwRxBufferSize
Size of allocated receive buffers in queue.
dwTxBufferCnt
Amount of allocated transmit buffers in queue.
dwTxBufferSize
Size of allocated transmit buffers in queue.
bDisableLocks
False: Thread-safe, True: Increase performance, but not thread-safe.

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

```
dwRes = ecatInitMaster(&oInitParms);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    LogError("ERROR EtherCAT-Master! %s (0x%x)", ecatGetText(dwRes), dwRes);
    goto Exit;
}
{
    EC_T_IOCTLPARMS oIoCtlParms = {0};
    oIoCtlParms.pbyInBuf        = (EC_T_BYTE*)EC_NULL + 0x20103;
    oIoCtlParms.dwInBufSize     = sizeof(EC_T_DWORD);
    ecatIoControl(EC_IOCTL_LINKLAYER_MAIN + EC_LINKIOCTL_FORCELINKMODE, &oIoCtlParms);
    OsSleep(1000);
}
```

For some operating systems it is possible to alternatively address the emlll8254x using its PCI address by adding EC_LINKUNIT_PCILOCATION (0x01000000) and the PCI location as dwInstance.

E.g. on Linux the PCI address of the emlll8254x can be shown using "`lspci | grep Ethernet`", e.g.:

```
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 dwInstanceId using PCI bus address is "`0x01bbddff`" with "`bb`" Bus Number, "`dd`" Device Number, "`ff`" Function Number. E.g. "`0000:00:19.0`" is dwInstanceId `0x01001900`.

2.5.4.2.7 RDC R6040 EtherCAT driver emllR6040

typedef struct _EC_T_LINK_PARMS_R6040
{
EC_T_LINK_PARMS linkParms;
} EC_T_LINK_PARMS_R6040;

Description
linkParms
Common parameters

Comment

```
#include "EcLink.h"
EC_T_LINK_PARMS_R6040 oLinkParmsAdapter;

OsMemset(&oLinkParmsAdapter, 0, sizeof(EC_T_LINK_PARMS_R6040));
oLinkParmsAdapter.linkParms.dwSignature   = EC_LINK_PARMS_SIGNATURE_R6040;
oLinkParmsAdapter.linkParms.dwSize        = sizeof(EC_T_LINK_PARMS_R6040);
OsStrncpy(oLinkParmsAdapter.linkParms.szDriverIdent,
          EC_LINK_PARMS_IDENT_R6040, MAX_DRIVER_IDENT_LEN - 1);
oLinkParmsAdapter.linkParms.dwInstance    = 1;
oLinkParmsAdapter.linkParms.eLinkMode     = EcLinkMode_POLLING;
oLinkParmsAdapter.linkParms.dwIstPriority = 0; /* not used */
```

2.5.4.2.8 VxWorks SNARF Link Layer

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

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

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

The network adapter driver has to be loaded prior to initialize the EtherCAT master stack.

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

2.5.4.2.9 Windows CE NDISUIO Link Layer

A NDISUIO like Link Layer is shipped with the EtherCAT master stack. This Link Layer is based on a network filter driver that enables the software on top to send and receive raw Ethernet frames. Using this Link Layer any Windows CE standard network drivers can be used.
A PCI network adapter in Windows CE usually is identified using its registry instance key. The format for the instance key is PCI\CCCCn where CCCC is the name for the device and n is the instance number. The device name for a Realtek 8139 network adapter is RTL8139 and thus the first instance of this adapter gets the instance key PCI\RTL81391.

The ECAT Link Layer driver AtNdisUio.dll has to be installed first. This driver may either be statically loaded using appropriate registry keys or dynamically on behalf of the application.
The following registry keys are required to install this driver:

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\ECAT]
    "Prefix"="ECT"
    "Dll"= "AtNdisuio.dll"
    "Index"=dword:1
    "Order"=dword:3
```

The following steps show how to load and check for the driver:

```
#define NDISUIO_DEVNAME   TEXT("ECT1:")
#define NDISUIO_DRIVERKEY TEXT("Drivers\\BuiltIn\\ECAT")
        HANDLE hNdisUioDevice    = EC_NULL;
        HANDLE hNdisUioDriver    = EC_NULL;
        /* check if NDISUIO driver started */
        hNdisUioDevice = CreateFile( NDISUIO_DEVNAME,
                                     GENERIC_READ | GENERIC_WRITE, 0, 0,
                                     OPEN_EXISTING, 0, 0);
        /* try to load driver if not already loaded */
        if (hNdisUioDevice == EC_NULL)
        {
            hNdisUioDriver = ActivateDeviceEx(NDISUIO_DRIVERKEY, 0, 0, 0);
        }
        /* check if driver is available */
        if ((hNdisUioDevice == EC_NULL) && (hNdisUioDriver == EC_NULL))
        {
            LogError("No NDISUIO ECAT driver available!!!\n" );
        }
        else if (hNdisUioDevice != EC_NULL)
        {
            /* close handle, it was just for the check */
            CloseHandle(hNdisUioDevice);
            hNdisUioDevice = EC_NULL;
        }
```

The parameters for the NDISUIO EtherCAT Link Layer are set using a specific descriptor. The pvLinkParams value is a pointer to this descriptor:

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

OsMemset(&oLinkParmsAdapter, 0, sizeof(EC_T_LINK_PARMS_NDISUIO));
oLinkParmsAdapter.linkParms.dwSignature = EC_LINK_PARMS_SIGNATURE_NDISUIO;
oLinkParmsAdapter.linkParms.dwSize      = sizeof(EC_T_LINK_PARMS_NDISUIO);
OsStrncpy(oLinkParmsAdapter.linkParms.szDriverIdent,
          EC_LINK_PARMS_IDENT_NDISUIO, MAX_DRIVER_IDENT_LEN - 1);
OsStrncpy(oLinkParmsAdapter.szAdapterName, TEXT(""PCI\\RTL81391"),
          MAX_LEN_NDISUIO_ADAPTER_NAME - 1);
```

typedef struct _EC_T_LINK_PARMS_NDISUIO
{
EC_T_LINK_PARMS linkParms;
EC_T_WCHAR szNetworkAdapterName[MAX_LEN_NDISUIO_ADAPTER_NAME];
} EC_T_LINK_PARMS_NDISUIO;

Description
linkParms
Common parameters
szNetworkAdapterName
Registry instance key for the network adapter.

Using the NDISUIO Link Layer has some major disadvantages. As the Windows CE network layering is involved in this architecture, the drivers are usually not optimized for realtime behavior and the needed CPU time is often too high to reach reasonable cycle times. Additionally there is an major impact concerning deterministic behaviour if in parallel to EtherCAT traffic the Windows CE applications need to use a second network card for transferring TCP/IP data.
Using a dedicated EtherCAT driver these disadvantages can be overcome.

2.5.4.2.10 Windows WinPcap based Link Layer

The parameter for the WinPcap EtherCAT Link Layer is just a pointer to the IP address string of the network adapter to use.

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

OsMemset(&oLinkParmsAdapter, 0, sizeof(EC_T_LINK_PARMS_WINPCAP));
oLinkParmsAdapter.linkParms.dwSignature = EC_LINK_PARMS_SIGNATURE_WINPCAP;
oLinkParmsAdapter.linkParms.dwSize      = sizeof(EC_T_LINK_PARMS_WINPCAP);
OsStrncpy(oLinkParmsAdapter.linkParms.szDriverIdent,
          EC_LINK_PARMS_IDENT_WINPCAP, MAX_DRIVER_IDENT_LEN - 1);
oLinkParmsAdapter.abyIpAddress[0] = 192;
oLinkParmsAdapter.abyIpAddress[0] = 168;
oLinkParmsAdapter.abyIpAddress[0] = 149;
oLinkParmsAdapter.abyIpAddress[0] = 150;
```

At least WinPcap version 4.1.2 must be used.

2.5.4.2.11 Xilinx Zynq-7000 (GEM) EtherCAT driver

The parameters to the GEM Link Layer are setup-specific.
The function “CreateLinkParmsFromCmdLineGEM” in selectLinkLayer.cpp demonstrates how to initialize the Link Layer instance.

```
typedef struct _EC_T_LINK_PARMS_GEM
{
    EC_T_LINK_PARMS     linkParms;
    EC_T_GEM_RXSOURCE eRxSource
    EC_T_DWORD          dwPhyAddr;
    EC_T_DWORD          dwRxInterrupt;
    EC_T_BOOL           bUseDmaBuffers;
    EC_T_BOOL           bNoPhyAccess;
} EC_T_LINK_PARMS_GEM;
```

Description

linkParms
    Common parameters
eRxSource
    [in] Source of Rx clock, control and data signals: eGemRxSource_MIO or eGemRxSource_EMIO
dwPhyAddr
    [in] PHY address (0 .. 31) on MII bus
dwRxInterrupt
    [in] Receive interrupt number (IRQ) (not used in polling mode)
bUseDmaBuffers
    [in] Use buffers from DMA (EC_TRUE) or from heap for receive. AllocSend is not supported, when EC_FALSE.
bNoPhyAccess
    [in] When EC_FALSE, Link layer should initialize PHY and read link status (connected/disconnected).
        When EC_TRUE, than client is responsible of PHY initialization and clock initialization

3 Software Integration

3.1 Application framework and example application

3.1.1 Overview

The example application EcMasterDemo will handle the following tasks:

- Showing basic EtherCAT communication
- Master stack initialization
- Start (set all slaves into OPERATIONAL state)
- Working with the EL9800 EtherCAT evaluation kit, EL2004, EL1004 and EL4132
- ``Out of the box`` solution for different operating systems:
  - Windows
  - Windows CE
  - RTOS-32
  - VxWorks
  - Linux
  - QNX
  - RTX
  - INtime
  - T-Kernel
  - …
- Example implementation for polled mode operation
- Thread with periodic tasks and application thread already implemented
- The output messages of the demo application will be printed on the console as well as in some files. The following log files will be created:
  - error0.log application error messages (logged via LogError function)
  - ecmaster0.log all messages

  Note: for VxWorks these log files are splitted to avoid memory resource problems when using the netdrv driver (FTP filesystem). After a specified number of messages the old file will be closed and thus written to the hard disk and a new file will be created. The file name convention is file.x.log, e.g. app.0.log for the first application log file and app.1.log for the second.

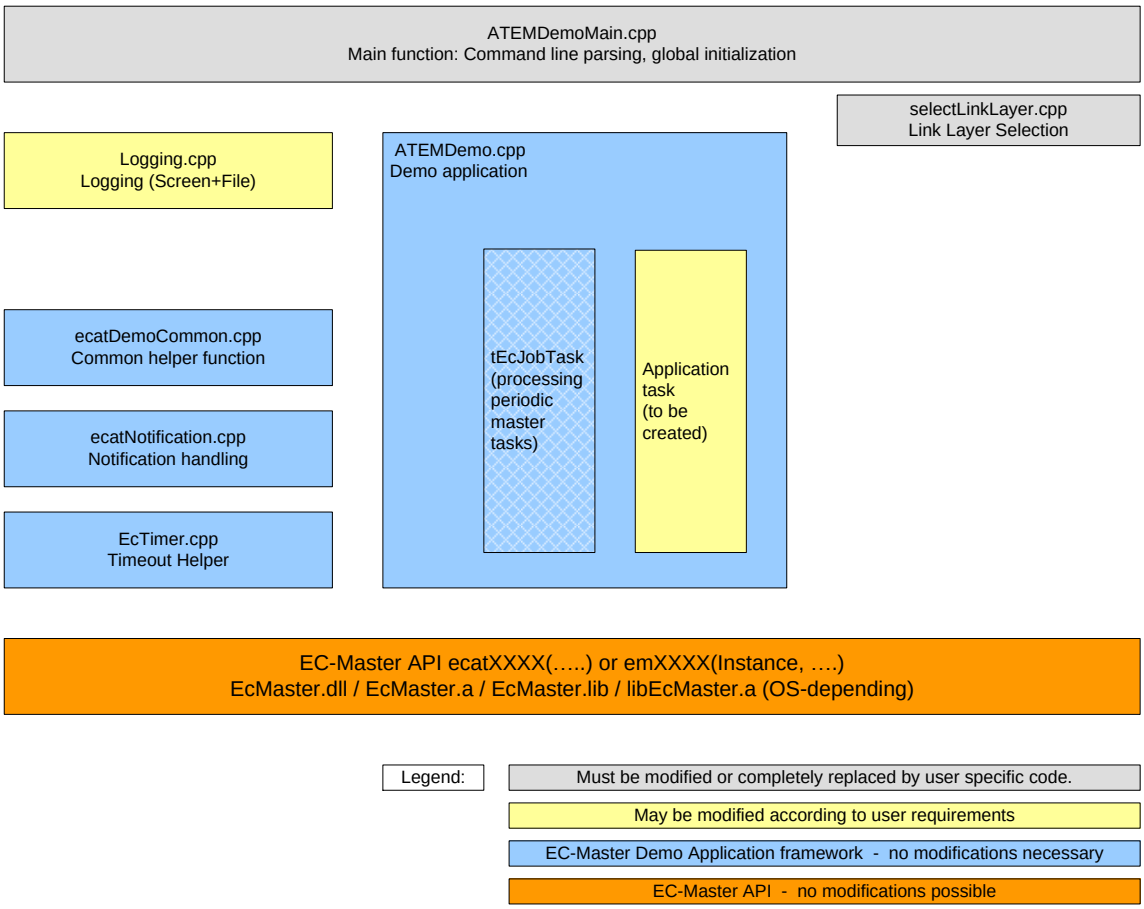
3.1.2 File reference

The EC-Master Demo application consists of the following files:

ATEMDemoMain.cpp	Entrypoint for the different operating systems and parsing of command line parameters
ATEMDemo.cpp	Initialize, start and terminate the EtherCAT master (function ATEMDemo())
ATEMDemoConfig.h	Contains basic static configuration parameters (task priorities, timer settings, EtherCAT master parameter)
selectLinkLayer.cpp	Common Functions which abstract the command line parsing into Link Layer parameters
ecatNotification.cpp	Slave monitoring and error detection (function ecatNotify())
ecatDemoCommon.cpp	Class with generic helper functions; e.g. trigger bus scan, getting information of slaves connected to the EtherCAT bus, CoE object dictionary example.
SlaveInfo.h	Slave information services (bus scan, slave properties)
Logging.cpp	Message logging functions

EcTimer.cpp	Start and monitor timeouts

The following picture gives an overview of delivered files and their responsibility.



Picture 1: Files overview

3.1.3 Master lifecycle

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

Basically the operation of the EtherCAT master is wrapped between the functions

- ecatInitMaster()
- ecatSetMasterState(<StateChangeTimeout>,eEcatState_OP)

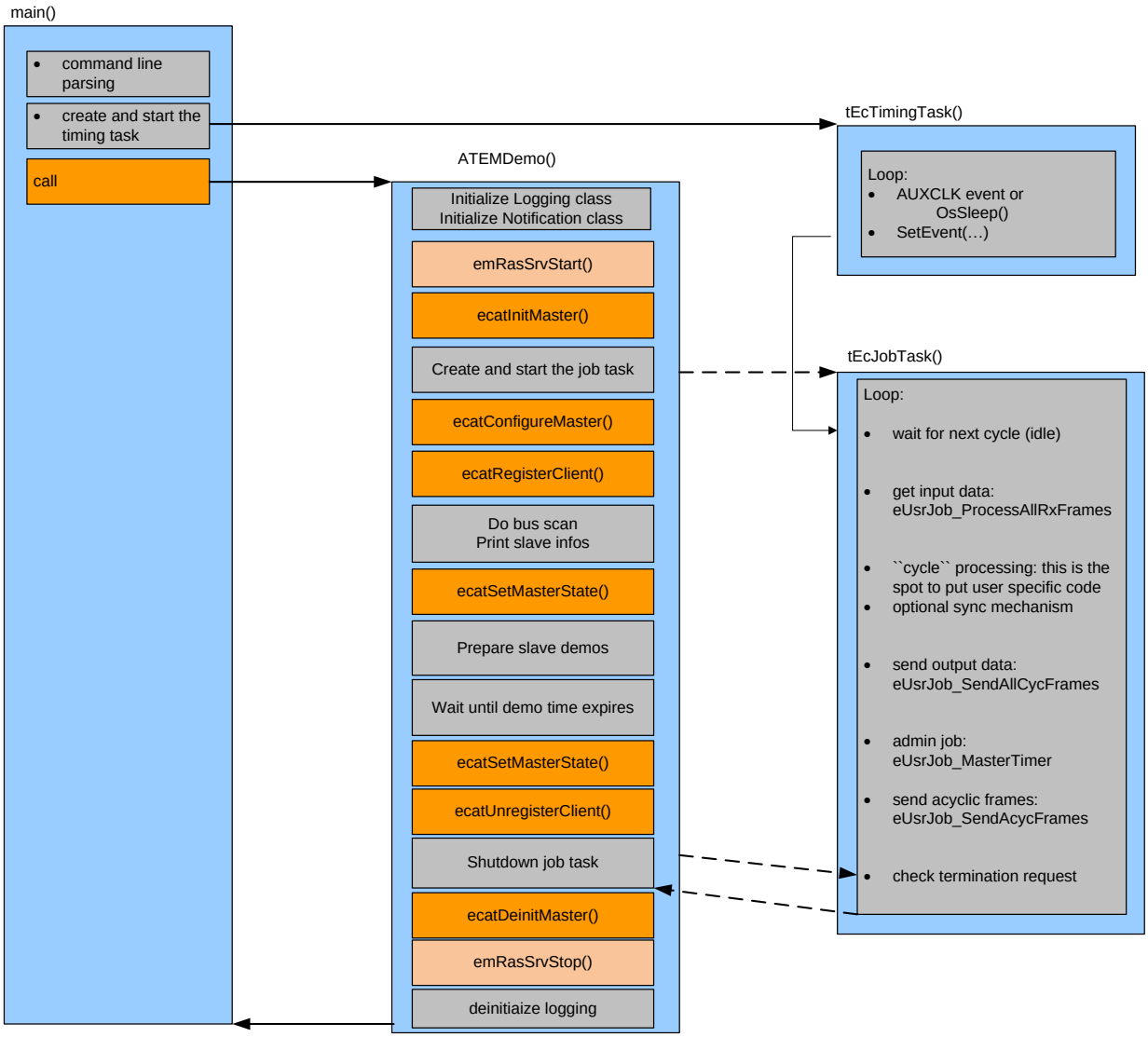
and

- ecatStop()
- ecatDeinitMaster()

With the former two functions the master is prepared for operation and started. Also during this preparation there is a thread set up and started, which does all the cyclic duties of the master.

With the later two functions, the master is stopped and memory is cleaned up.

An overview of the complete lifecycle is given in Picture.



Picture 2: Demo application lifecycle

Here a somewhat closer description of the functions:

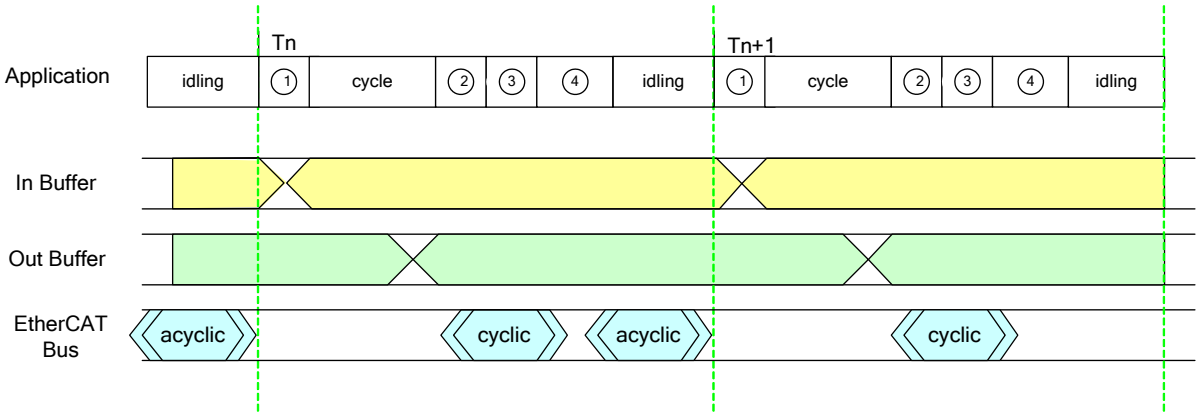
main()	Simply the wrapper to start the demo; probably dependant on the operating system. Implement command line parsing for individual parameter setting here.
ATEMDemo()	Demo application. The function takes care for starting and stopping the master and all related tasks. In between there is a spot where the function does nothing (idling). During this idling all relevant work is done by the ``tEcJobTask()``, described below.

atemDemoInit()	Prepare the master for operation and create the application threads. In case of the demo this is the ``tEcJobTask()``. More threads can be created if required by the user application. Not shown is the creation of an instance of the ``CEmNotification`` class, which will be explained later.
atemDemoDeInit()	Stop all application threads.
tEcJobTask()	Thread which does the necessary, periodic work. Very important here is the spot between the calls to ``eUsrJob_ProcessAllRxFrames`` and to ``eUsrJob_SendAllCycFrames``. In the chapter 3.1.4 this spot is referenced as ``cycle``. Application specific manipulation of the process image, which must be synchronous with the bus cycle, can be put in myAppWorkPd.
tEcTimingTask()	Timing Thread. This thread sets the timing event that triggers the tEcJobTask for the next cycle.
ecatSetMasterState()	EC-Master API function: Startup the EtherCAT master and switch the bus to the different states from INIT to OPERATIONAL.
ecatStop()	EC-Master API function: Stop master operation. The master quits sending packets.
ecatInitMaster()	EC-Master API function: Prepare the master for operation and set operational parameters, e.g. used Link Layer, buffer sizes, maximum number of slaves, … .
ecatDeinitMaster()	EC-Master API function: Clean up.
ecatConfigureMaster()	EC-Master API function: Tell master about it’s XML file configuration.

Remark: During cyclic operation, which is the desired OPERATIONAL state of an EtherCAT system, the main work is done in the `` tEcJobTask()``. The originating process, in case of the demo this is ``ATEMDemo()``, is doing nothing (idling).

3.1.4 Synchronisation

This chapter puts the tasks or functions, which run in the `` tEcJobTask()``, into relation with timing and communication on the EtherCAT bus. See Picture.



Picture 3: Synchronisation

Application	Shown are the tasks/jobs (1) through (4) which must be done by the application every single cycle. The details of the individual tasks are described below. When the application is done with the jobs, it waits for the next cycle. (period between (4) and (1)).
In buffer	Shown are the contents of the input section of the process image. The contents are not valid while the EtherCAT master updates the data (1).
Out buffer	Shown are the contents of the output section of the process image. The contents are not valid while the application updates the data (1).
EtherCAT bus	Shown are the timing positions, when the EtherCAT master does cyclic and acyclic communication on the EtherCAT bus. Besides the timing position of the start for the cyclic frames, the shown positions may vary, depending on the number of frames.

In the ``ATEMDemo()`` application the tasks/jobs (1) through (4) shown in the picture are managed and scheduled by the `` tEcJobTask()``. Here a more detailed description:

Job 1	The job ``eUsrJob_ProcessAllRxFrames`` works on the frames and data received with previous bus activity. This includes cyclic as well as acyclic frames. The received frames are analysed for new input data and the local process image is updated. During this process the input data section of the process image is invalid.
cycle	In the current ATEMDemo(): Call ELxxxx() slave functions. This is the spot where a user defined application can manipulate the process image. The application has updated input information (from Job 1 above), can do calculations and manipulation, and write new data to the output section of the process image.
Job 2	This function triggers the transmission of all cyclic frames on the EtherCAT bus.
Job 3	The job ``eUsrJob_MasterTimer`` has administrative charater and are basically necessary to run the timeout timers.

There is no interaction with the process image during these calls nor does this call trigger any bus traffic.
It is not necessary to run this function with every bus cycle, especially on systems with short cycle times < 1 msec. But it is recommended to run this function with a 1 msec period.

Job 4 — With the job ``eUsrJob_SendAcycFrames`` call, the acyclic frames are scheduled for transmission.

idling — Currently implemented as waiting for the next cycle (triggered by the timing event).

3.1.5 Event notification

The EtherCAT master provides event notification for a great number of events. These events are for example:

- bus state change
- link state change
- working counter errors
- …
- many more

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

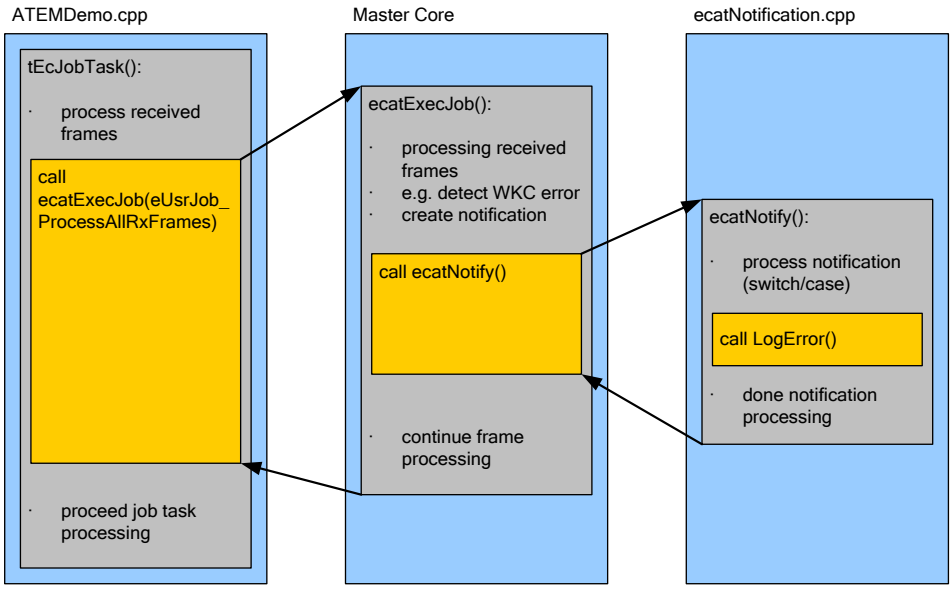
ecatRegisterClient()

In case of the EcMasterDemo the class ``CEmNotification`` is provided. It implements the complete framework to catch and handle the EtherCAT stack's notifications. The class is instantiated once and attached to the EtherCAT master with the `` ecatRegisterCllient()`` call shown above. The class implements the method ``ecatNotify()`` as major entry point (or callback function) for events.

There exist three different ways how events can be handled. The method to handle an event is mainly determined by the time required to handle the event and the processing context in which the event shall be handled. The methods are described below.

3.1.5.1 Direct

Minor events can be handled directly within the context where they are detected. One possible example for such an event is the detection of a wrong working counter (WKC). See Picture for the example.

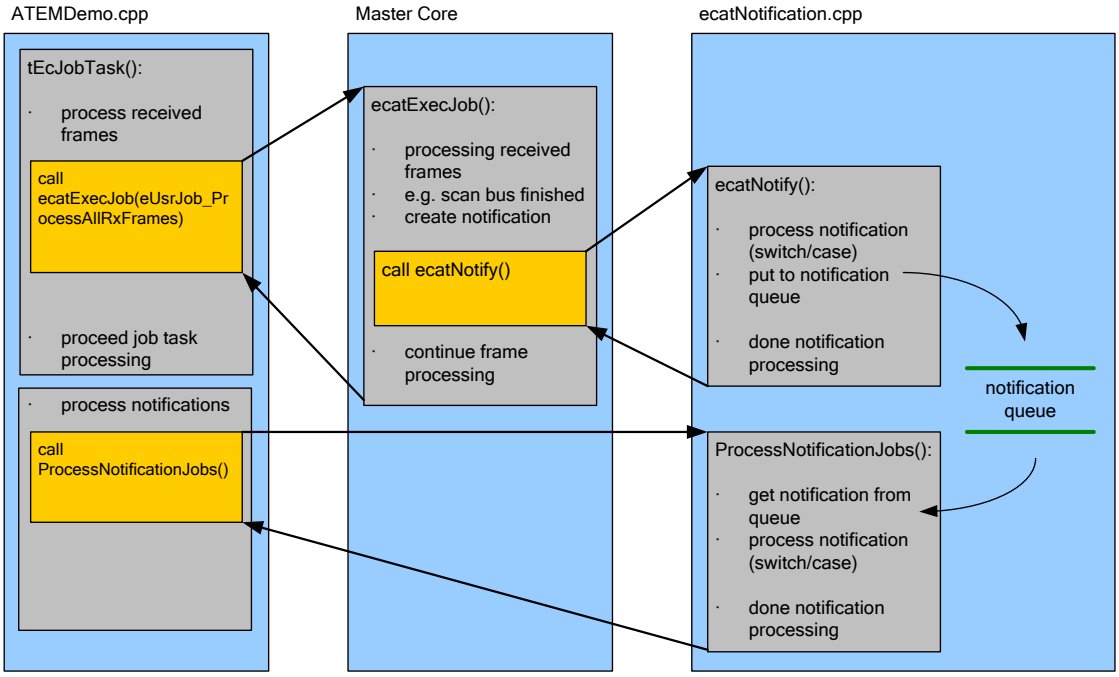


Picture 4: Direct event handling

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

3.1.5.2 Queue

Events which require more time consuming processing can not be handled directly in the context where they are detected. Handling or processing of the event must be postponed. This is achieved by a queue, which is also readily implemented with the ``CEmNotification`` class. Picture shows an example for this behavior.



Picture 5: Postponed (or queued) event handling

The job task periodically checks the event queue for events which must be handled. This is done via the simple call to ``CEmNotification::ProcessNotificationJobs()``. When there is an event in the queue, it is handled during this call, which is within the context of the job task.
Important: The call to CEmNotification::ProcessNotificationJobs() shall NOT be executed in the tEcJobTask(). As the CPU time consumption may be high, this would have a high impact to the real-time behavior of the cyclic operation.

3.1.6 File logging

There are several functions which log information into files. In case the VxWorks netDrv driver is used (e.g. when accessing the PC hard disk via FTP) the content of these files is stored in memory. These files will be closed only when the application terminates or when the specified number of messages to be stored in a single file is exceeded. There are no means to limit the amount of memory needed for those files. If you extend the duration of the demo and the number of messages to be stored in the log file is too high it may occur that there is no free memory available and thus the system to crash.
Using the InitLogging() function the application can determine the rollover parameter when a new log file shall be created.
The file logging for VxWorks is disabled by default. Setting the global variable bLogFileEnb to a value of 1 the file logging is enabled.

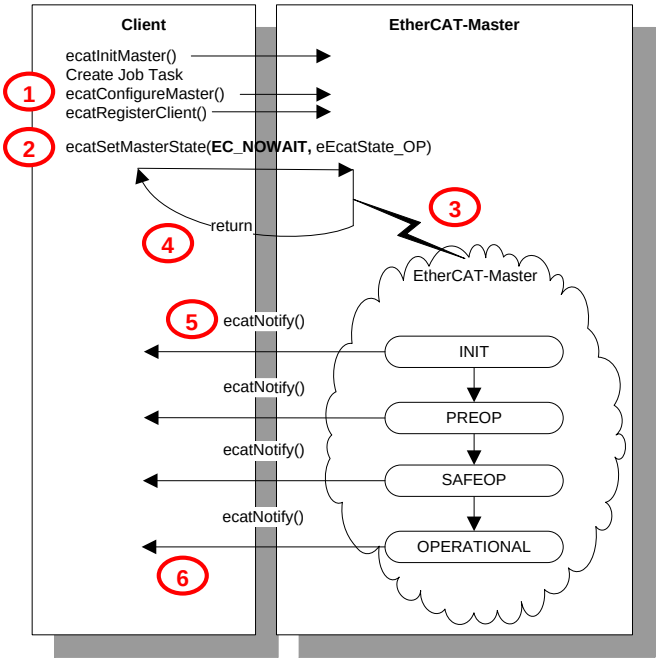
3.1.7 Master stack debug messages

Every time the EtherCAT master stack detects an error it calls the OS-Layer function OsDbgMsg(). The demo application registers a hook OsDbgMsgHook() to be called by OsDbgMsg(). The hook will store those messages into a log file (ecat.log).

3.2 Master startup

The master stack has to be initialized once when the application is starting. After this one-time initialization one or more clients may register with the master. Finally, after all clients are registered the master can be started.
Starting the master means that all slaves will be set into the operational state. Every time the state of the master has changed the clients are notified about this state-change.

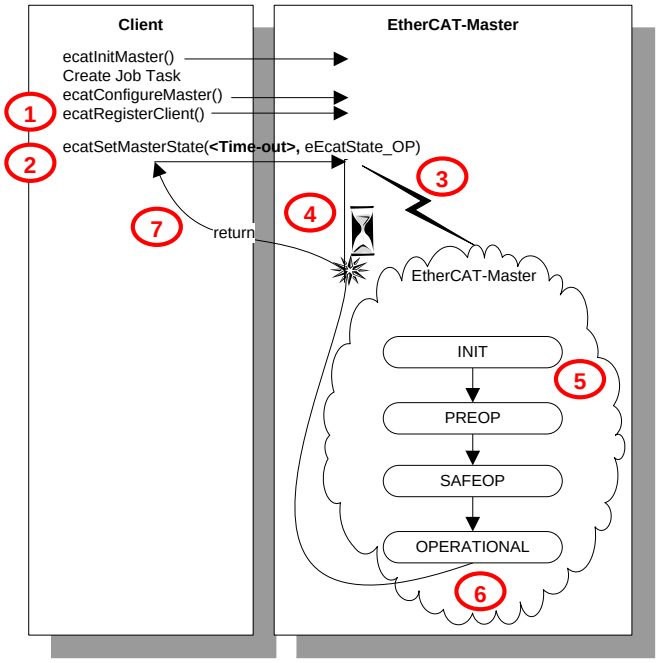
Asynchronous (deferred) startup



- Client calls `ecatInitMaster(…)`

- Client creates Job Task. See 3.1.3, “Master lifecycle”.
- Client calls **ecatConfigureMaster(…)**
- Client calls **ecatRegisterClient(…) (See “1”)**
- Client calls **ecatSetMasterState(…)** with a timeout parameter EC_NOWAIT **(See “2”)**
- Inside **ecatSetMasterState(…)** the master startup procedure will be initiated **(See “3”)**
- Function **ecatSetMasterState(…)** returns immediately (EC_NOWAIT) **(See “4”)**
- The master initializes all slaves until all slaves reach OPERATIONAL state
- After every state change the client will be notified **(See “5”)**
- After reaching the **OPERATIONAL** state the system is ready **(See “6”)**

Synchronous startup



- Client calls **`ecatInitMaster(…)`**
- Client creates Job Task. See 3.1.3, “Master lifecycle”.
- Client calls **`ecatConfigureMaster(…)`**
- Client calls **`ecatRegisterClient(…)`** **`(See “1” )`**
- Client calls **`ecatSetMasterState(…)`** with an appropriate timeout value **`(See “2” )`**
- Inside **`ecatSetMasterState(…)`** the master startup procedure will be initiated **`(See “3” )`**
- The client is blocked until the whole startup has finished **`(See “4” )`**
- The master initializes all slaves until all slaves reach OPERATIONAL state **`(See “5” )`**
- After reaching the **`OPERATIONAL`** state the system is ready **`(See “6” )`**
- **`ecatSetMasterState(…)`** returns **`(See “7” )`**

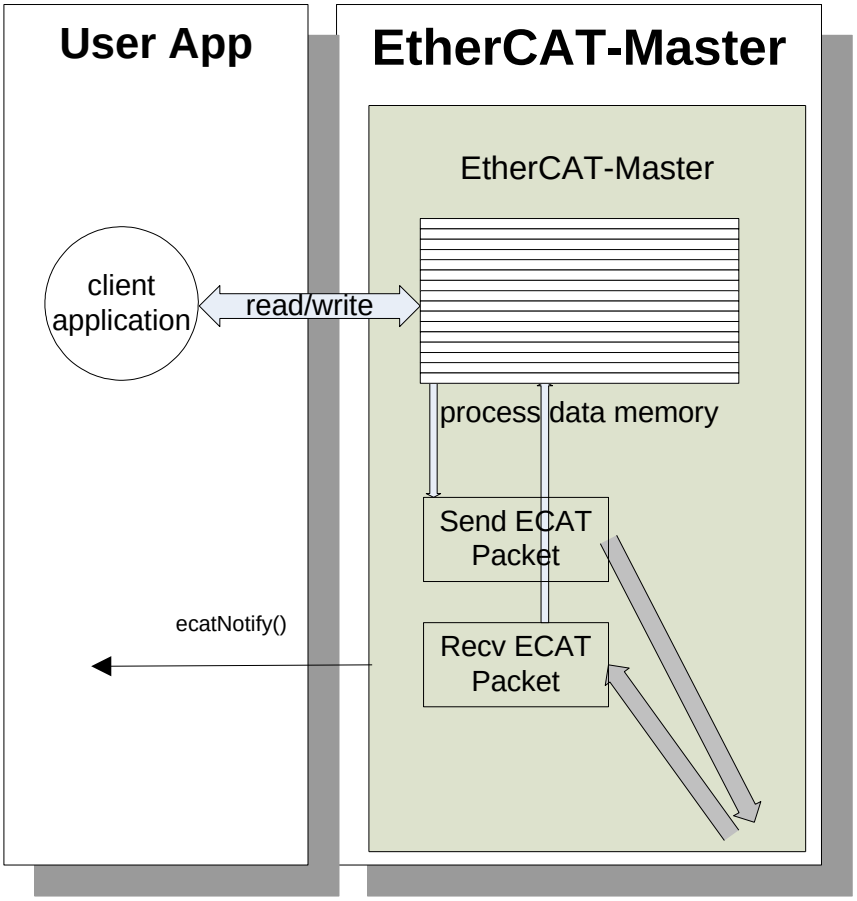
3.3 Process data update and synchronization

The EtherCAT master's main task is to exchange process data objects between the client and the EtherCAT slaves. All mapped process data objects of the slaves will be copied by the master into a process data memory area.
New input values received from the slaves will be written into the input process data memory.
New output values which shall be sent to the slaves will be read from the output process data memory.
When the client registers with the master the client gets a pointer to those process data memory areas.
The EtherCAT master has two different options how process data memory is allocated (provided).

A) The master itself is the memory provider (default case)
B) The application serves as the memory provider

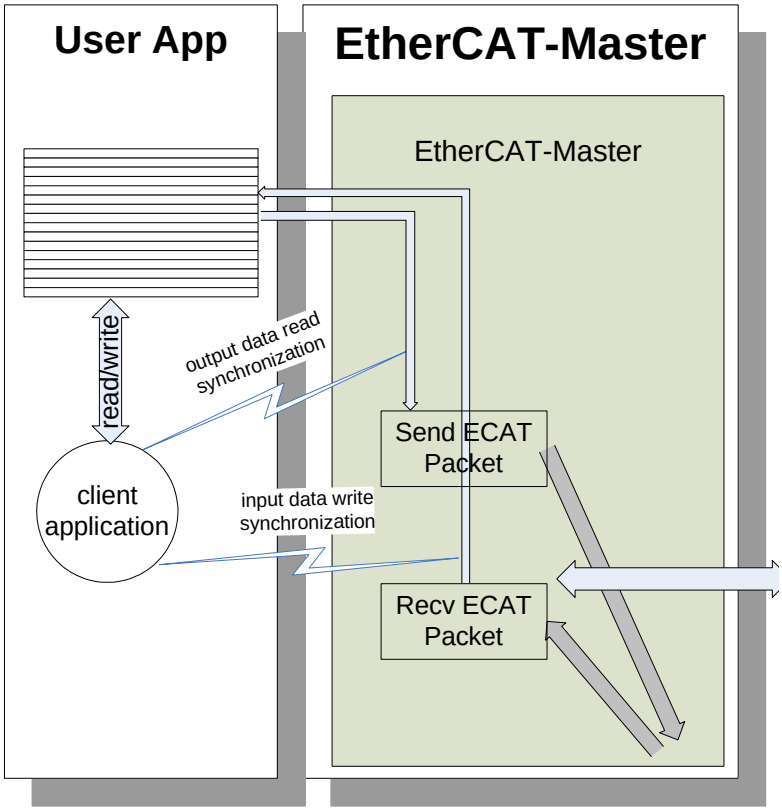
3.3.1 EtherCAT master as process data memory provider

If the application does not register a memory provider then the master will internally allocate the necessary memory needed to store input and output process data values.



3.3.2 User application as process data memory provider

The application may register a memory provider in case the master shall use externally allocated memory to store input and output process data values. See 4.3.30 “ecatIoControl – EC_IOCTL_REGISTER_PDMEMORYPROVIDER”.



The memory provider may optionally supply callback functions to synchronize memory access between the clients and the master.

In this case the master’s sequence of receiving new input process data values is as follows:

- new process data is received, data is internally located in the Ethernet frame buffer
- master requests write access to the input process data memory (callback function pfPDInDataWriteRequest)
- master copies input data into process data memory
- master releases write access (callback function pfPDInDataWriteRelease)

The master’s sequence of sending new output process data values is as follows:

- Application: new process data is stored in the output process data memory buffer
- Master requests read access to the output process data memory (callback function pfPDOutDataReadRequest)
- Master copies output data into the Ethernet frame
- Master releases read access (callback function pfPDOutDataReadRelease)
- new output process data will be sent to the slaves

3.3.3 Process data memory provider: fixed and dynamic buffers

The EtherCAT master uses two separate buffers where process data input values and process data output values are stored. The buffers used may either be always the same (fixed buffers) or be changed on every process data transfer cycle (dynamic buffers).

Case 1: EtherCAT master provides process data memory (fixed buffers)
If the application does not call ecatIoControl - EC_IOCTL_REGISTER_PDMEMORYPROVIDER the master is used as process data memory provider. The memory will then be allocated once when the master is configured (ecatConfigureMaster). The master will always use the same buffers to read/write process data.

Case 2: User application registers an external memory provider with fixed buffers
When the application calls ecatIoControl - EC_IOCTL_REGISTER_PDMEMORYPROVIDER it can determine the address where the process data buffers are located. These fixed buffers will be used by the EtherCAT master to store process data.

Case 3: User application registers an external memory provider without fixed buffers
The application calls ecatIoControl - EC_IOCTL_REGISTER_PDMEMORYPROVIDER with setting the fixed buffer address values to EC_NULL. In this case the EtherCAT master will request the buffer addresses cyclically when process data are read or written (callback functions). This mode may be used to implement dynamic buffering mechanisms between the application and the EtherCAT master (double buffering, triple buffering).

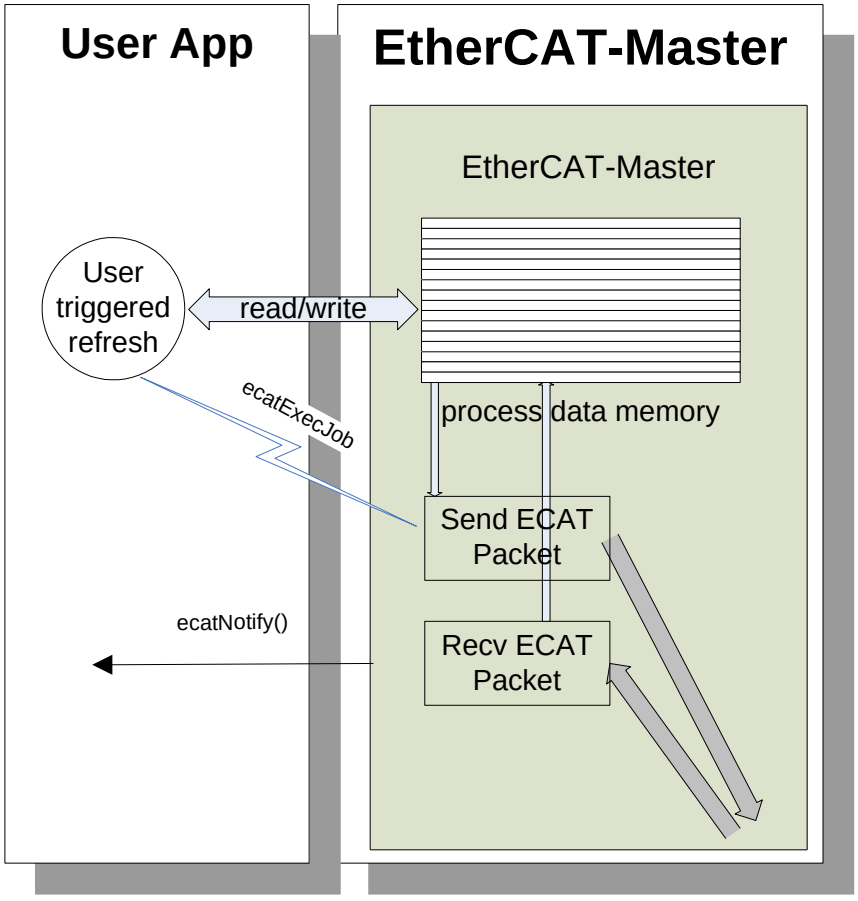
3.3.4 Process data synchronization

The master operation is fully controlled by the user's application. Thus the user application is responsible for synchronization of the process data between the EtherCAT master stack and the application itself.
To update the process data in the slaves, one registered client task initiates the update by calling the function ecatExecJob (see 4.3.14) with the *eUsrJob_SendAllCycFrames* parameter.
Using the same function with appropriate parameters a very flexible operation of the whole system can be achieved (see below).

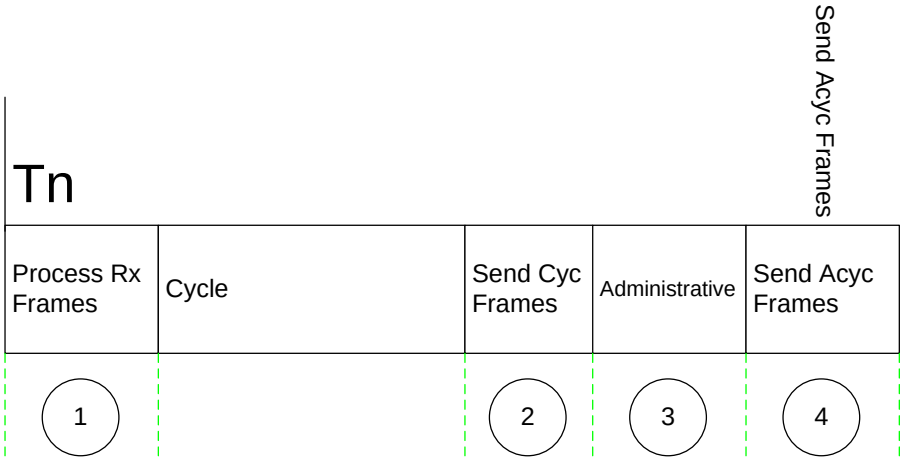
When a process data update is initiated by calling ecatExecJob(*eUsrJob_SendAllCycFrames*) new output data are read from the process data output area and stored in Ethernet/EtherCAT frames prior to sending them to the Link Layer. When this call returns all output process data values are stored in Ethernet/EtherCAT frames which are then processed by the network controller. If only one single thread is both writing into the process data output area and calling ecatExecJob(*eUsrJob_SendAllCycFrames*) no further output process data synchronization is necessary.

The application is responisible to (cyclically) calling the function ecatExecJob with the appropriate parameters.

Note: By using the callback functions when registering an external memory provider it is possible to synchronize process data memory access between the application and the master. This may be necessary for example in a multi-threading environment.



The master operates in a single mode called User Controlled Mode. This mode can be used in all applications, it is optimized for systems which require a tightly coupled timing, e.g. PLC with MCFB (Motion Control Function Blocks) or position control realized by the control application instead of a drive control (intelligent drive). The (user) application has to initiate each step necessary to update process data as well as for the master's internal management.
The timing of the necessary jobs within one cycle is described in the following diagram.



Steps the application (PLC) has to call:

1 – eUsrJob_ProcessAllRxFrames
Link layer in polling mode: process all received frames (e.g. read new input process data).

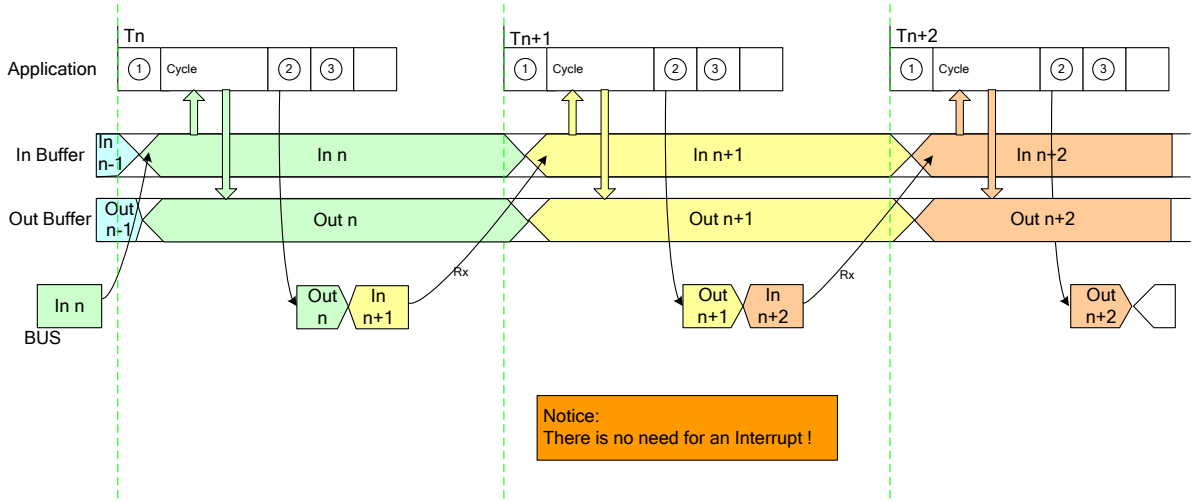
2 – eUsrJob_SendAllCycFrames
Send cyclic frames to update process output data.

3 – eUsrJob_MasterTimer
Trigger the lower level master and slave state machines.
This job has to be called cyclically. The master cycle time is determined by the period between calling ecatExecJob(eUsrJob_MasterTimer).

4 – eUsrJob_SendAcycFrames
Transmit pending acyclic frame(s).

Note: When the Link Layer is running in interrupt mode processing of received frames is done immediately after the frame is received.

3.3.4.1 Cyclic frames – Link layer in polling mode

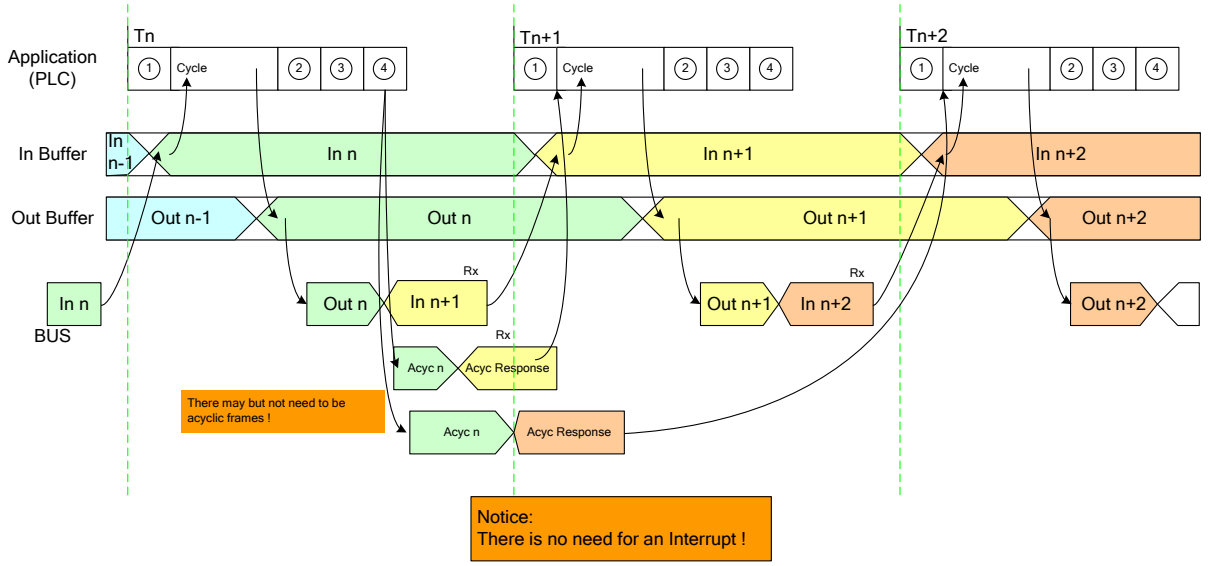


Application has to perform:

```
/* Job 1: incoming process data is stored to Process data image.
*/
ecatExecJob( eUsrJob_ProcessAllRxFrames, &bPrevCycProcessed );
…
…
/* do your process data cycle */
…
…
/* Job 2: send out actualized process data. Outputs are actualized in slaves, and input
data is collected to be present for next cycle. Storage to process data image is done
while eUsrJob_ProcessAllRxFrames
*/
ecatExecJob( eUsrJob_SendAllCycFrames, EC_NULL );

/* Job 3: trigger master state machines, which are necessary to perform any status change
or master internal administration tasks
*/
ecatExecJob( eUsrJob_MasterTimer, EC_NULL );
```

3.3.4.2 Cyclic and acyclic frames – Link layer in polling mode



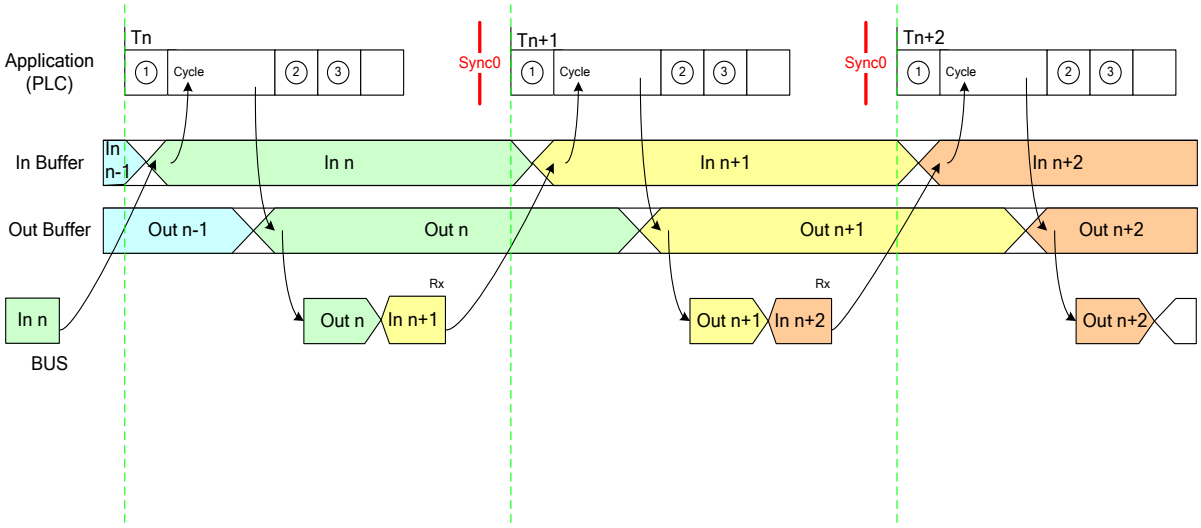
Application has to perform:

```
/* Job 1: incoming process data is stored to Process data image. */
ecatExecJob( eUsrJob_ProcessAllRxFrames, &bPrevCycProcessed );
…
…
/* do your process data cycle */
…
…
/* Job 2: send out actualized process data. Outputs are actualized in slaves, and input
data is collected to be present for next cycle. Storage to process data image is done
while eUsrJob_ProcessAllRxFrames
*/
ecatExecJob( eUsrJob_SendAllCycFrames, EC_NULL );

/* Job 3: trigger master state machines, which are necessary to perform any status change
or master internal administration tasks
*/
ecatExecJob( eUsrJob_MasterTimer, EC_NULL );

/* Job 4: transmit queued acyclic commands, which in case can be enqueued through user
application or master administrative tasks. Reponses are retrieved by issuing
eUsrJob_ProcessAllRxFrames.
*/
ecatExecJob( eUsrJob_SendAcycFrames, EC_NULL );
```

3.3.4.3 Cyclic frames with DC – Link layer in polling mode

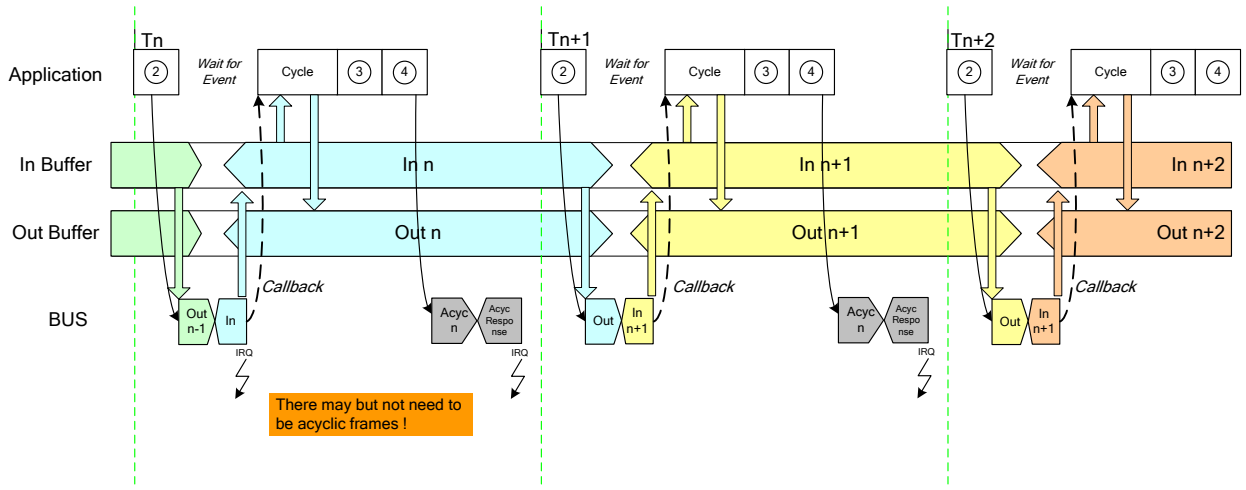


Application has to perform:

```
/* Job 1: incoming process data is stored to Process data image. */
ecatExecJob( eUsrJob_ProcessAllRxFrames, &bPrevCycProcessed );
…
/* do your process data cycle */
…
/* Job 2: send out actualized process data. Outputs are actualized in slaves, and input
data is collected to be present for next cycle. Storage to process data image is done
while eUsrJob_ProcessAllRxFrames
*/
ecatExecJob( eUsrJob_SendAllCycFrames, EC_NULL );

/* Job 3: trigger master state machines, which are necessary to perform any status change
or master internal administration tasks
*/
ecatExecJob( eUsrJob_MasterTimer, EC_NULL );
```

3.3.4.4 Cyclic and acyclic frames – Link layer in interrupt mode



Application has to perform during startup:

```
dwRes = ecatInitMaster(&oInitParms);

/* setup callback function which is called after RX */
dwRes = ecatIoControl(EC_IOCTL_REGISTER_CYCFRAME_RX_CB, &oIoCtlParms);

/* create cyclic process data Thread */
S_pvtJobThread = OsCreateThread( "tEcJobTask", tEcJobTask, JOBS_PRIORITY,
PROCDATA_THREAD_STACKSIZE, (EC_T_VOID*)&S_DemoThreadParam);

dwRes = ecatConfigureMaster(eCnfType_Filename, szCfgFile, OsStrlen(szCfgFile));
```

Application has to perform inside job task:

```
/* Job 2: send out process data. Outputs are actualized in slaves, and input data is
collected to be present for current cycle. Storage to process data image is done after
reception of response frame within the interrupt service thread */
ecatExecJob( eUsrJob_SendAllCycFrames, EC_NULL );

/* wait until cyclic frame is received */
OsWaitForEvent(S_pvCycFrameRxEvent, CYCLE_TIME)

/* do your process data cycle */
/*-----------------------------*/

/* Job 3: trigger master state machines, which are necessary to perform any status change
or master internal administration tasks
*/
ecatExecJob( eUsrJob_MasterTimer, EC_NULL );

/* Job 4: transmit queued acyclic commands, which in case can be enqueued through user
application or master administrative tasks.*/
ecatExecJob( eUsrJob_SendAcycFrames, EC_NULL );
```

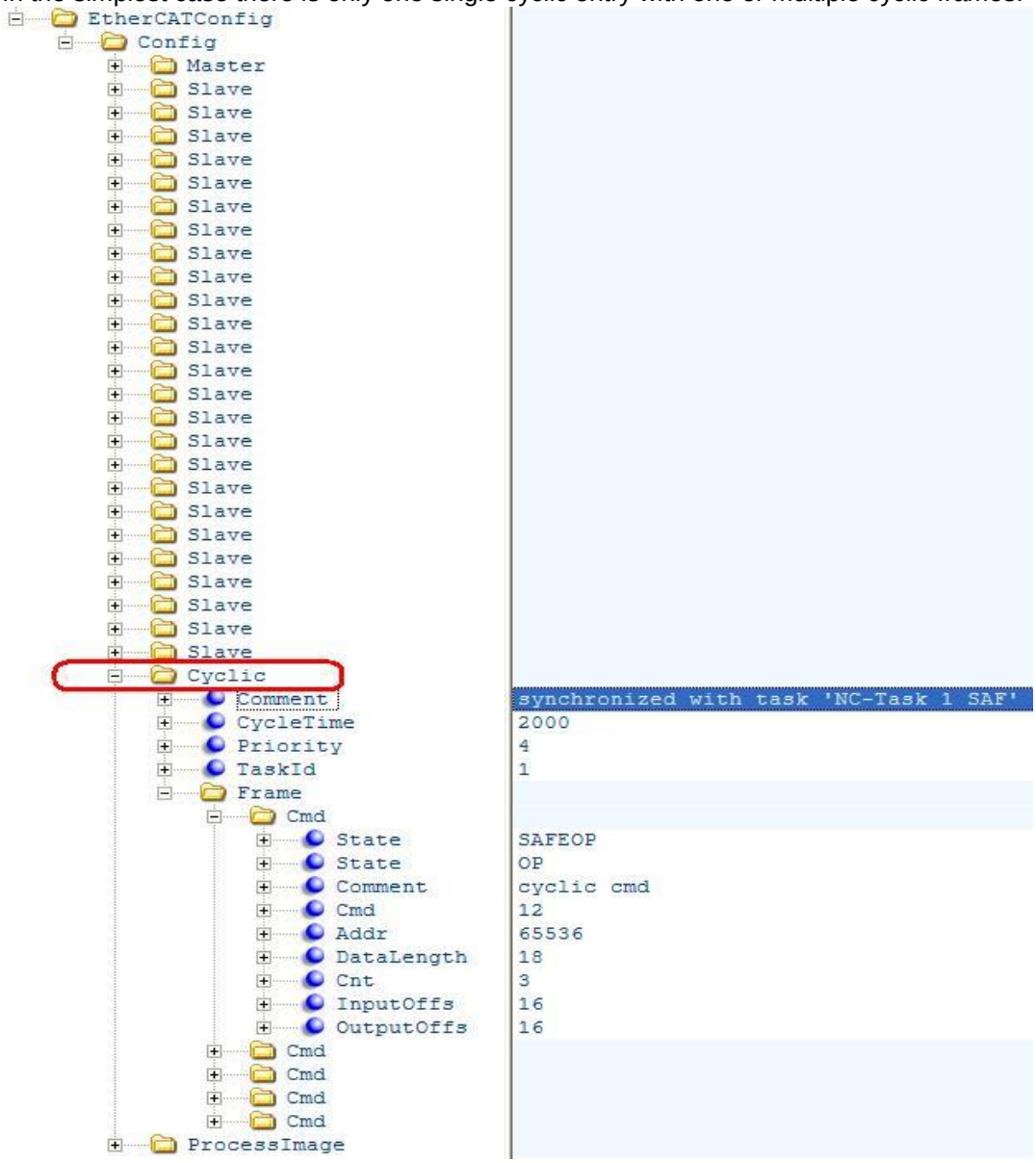
For closer details find an example project <EcMasterDemoSyncSm> in the folder *Examples*

3.3.5 Single or multiple cyclic entries in ENI file

For reading new input data values and writing new output data values (process data update) the EtherCAT configuration file contains one or multiple “Cyclic” entries. These entries contain one or multiple frames (so-called cyclic frames) to be sent cyclically by the master.
Inside the cyclic frames there are one or multiple EtherCAT datagrams containing logical read/write commands for reading and writing process data values.

3.3.5.1 Configuration variant 1: single cyclic entry

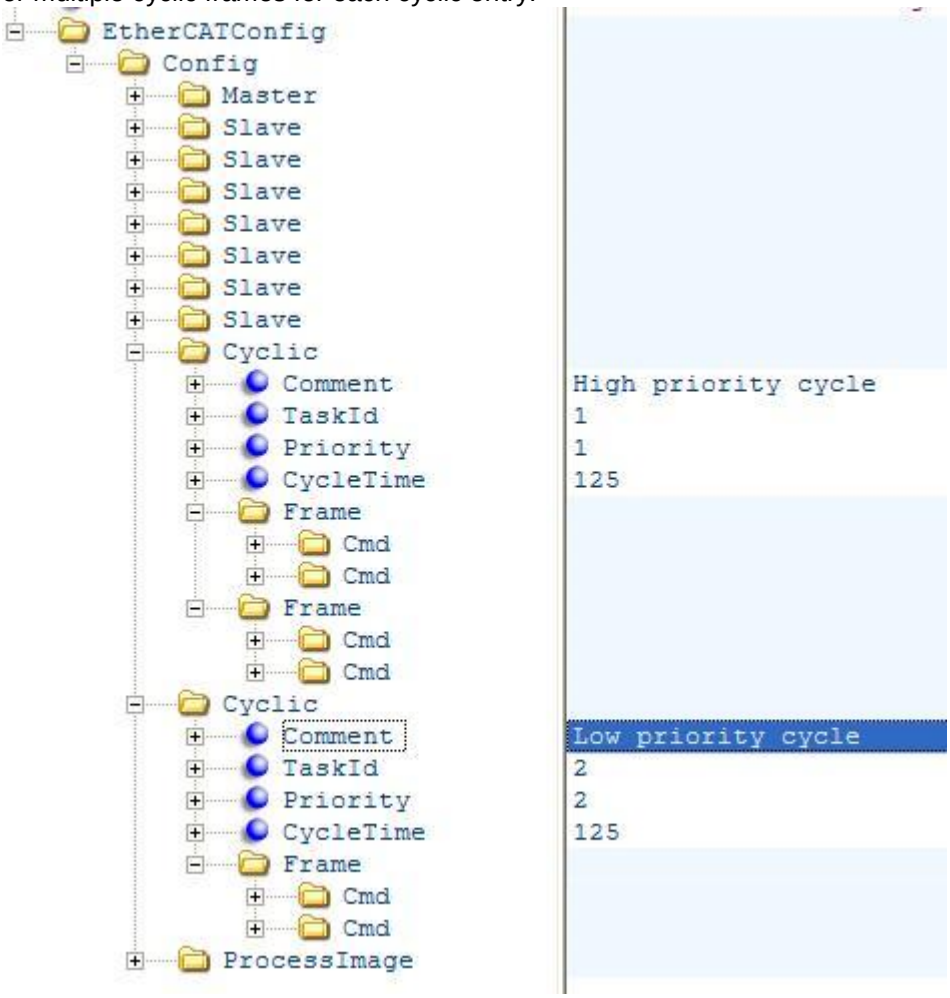
In the simplest case there is only one single cyclic entry with one or multiple cyclic frames.



All process data synchronization modes support this configuration variant. See 3.3.4 “Process data synchronization”.

3.3.5.2 Configuration variant 2: multiple cyclic entries

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



The application has to use the eUsrJob_SendCycFramesByTaskId job call to the master to send the appropriate cyclic frame. See 4.3.14 “ecatExecJob”.

3.3.6 Copy Information for Slave-to-Slave communication

It is possible to configure the system to copy Input Variables to Output Variables within the Master Stack instead of Slave-to-Slave communication via LRW.

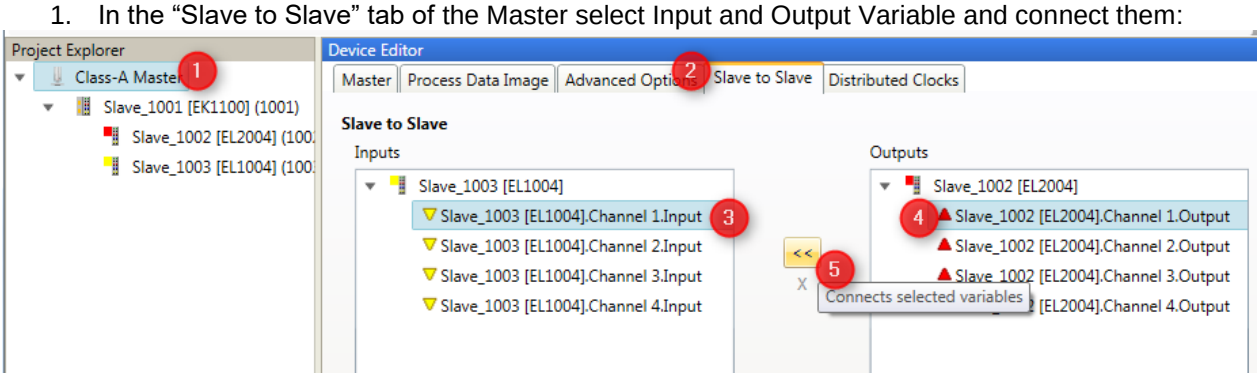
The copy info declarations of the corresponding received cyclic frame are processed on cyclic frame processing in ecatExecJob(eUsrJob_ProcessAllRxFrames, …).

It takes two communication cycles to exchange the process data, but it is needed e.g. if applying cable redundancy or if WKC of INPUT must be checked before changing OUTPUT, see 3.3.8 “Cyclic cmd WKC validation”.

The copy info declarations are located at /EtherCATConfig/Config/Cyclic/Frame/Cmd/CopyInfos in the ENI file, see 9.15 “CopyInfosType” in ETG.2100.

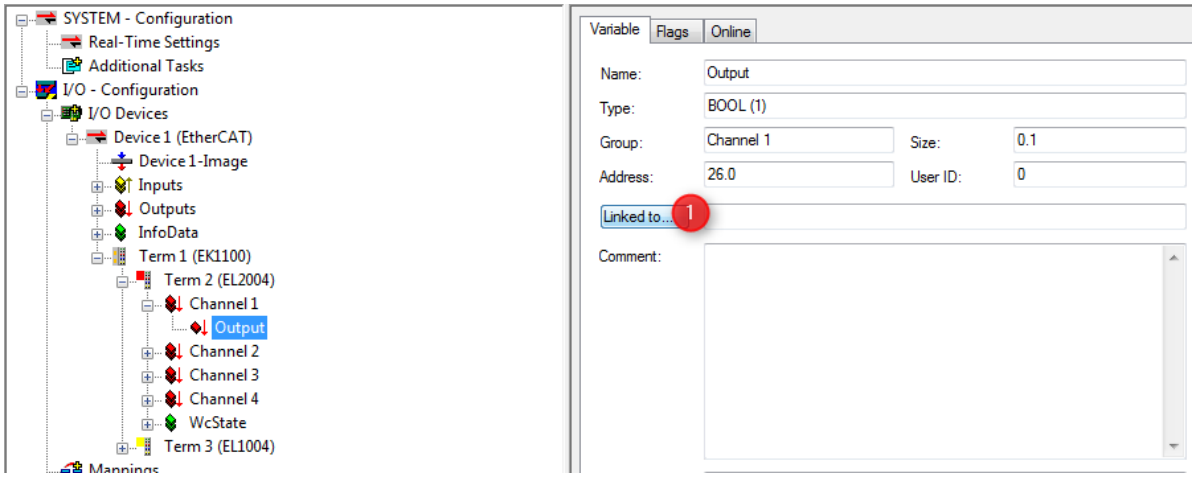
Configuration with EC-Engineer

1. In the “Slave to Slave” tab of the Master select Input and Output Variable and connect them:

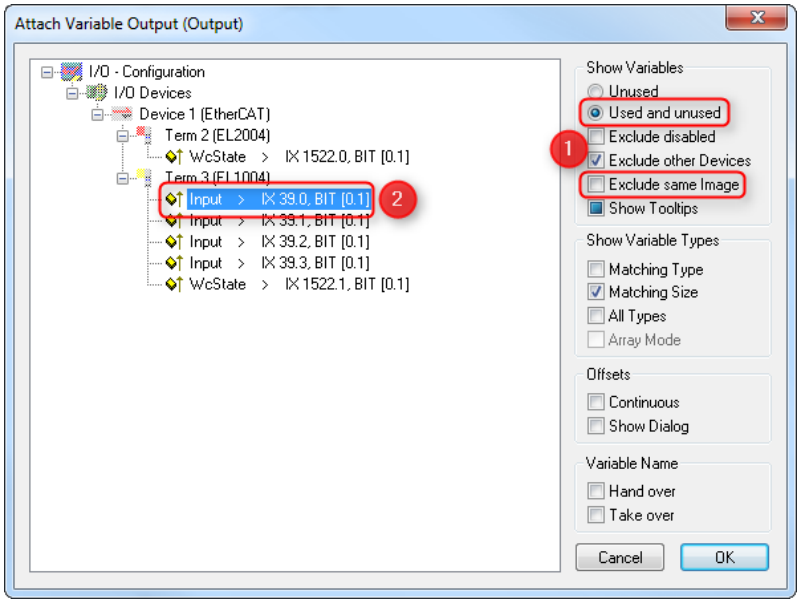


Configuration with ET9000

1. Select “Linked to…” from the Output Variable:



2. Select Input Variable to be attached to the Output Variable:

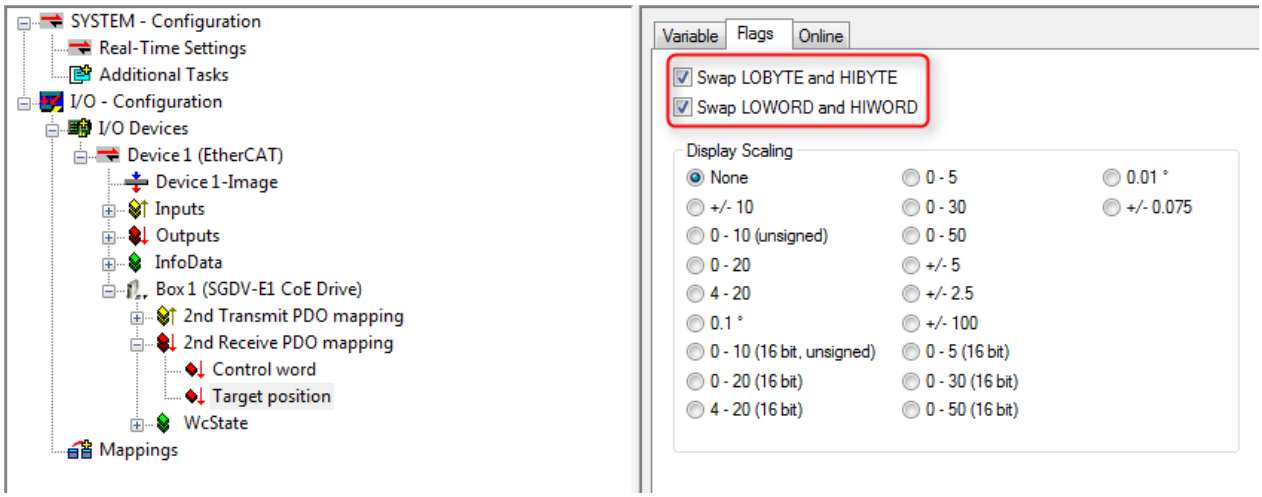


Hint

Copy info declaration processing is independent of WKC values, but updating the INPUT source depends on successful *Cyclic cmd WKC validation*, see 3.3.8 “Cyclic cmd WKC validation”.

3.3.7 Swap variables' bytes according to ENI

The following screenshot (ET9000) shows how to configure variables to be swapped by the EC-Master:



Hint: The EC-Master does not distinguish between WORD or BYTE swapping. Setting any PDO swap flag instructs the EC-Master to swap the PDO variable.

The swap declarations are located at DataType's attribute SwapData of RxPdo or TxPdo, e.g. /EtherCATConfig/Config/Slave/ProcessData/RxPdo/Entry/DataType in the ENI file.

3.3.8 Cyclic cmd WKC validation

New input values received from the slaves will be written into the input process data memory **only** if the WKC of the corresponding datagram is not 0 and not greater than the expected WKC value.
See also *EC_NOTIFY_CYCCMD_WKC_ERROR*.

3.4 Accessing process data in the application

The process data, exchanged between the EtherCAT master and the slaves in every cycle, are stored in the process data image. There are two separate memory areas, one for the input data and another one for the output data. The base addresses of these areas are provided by calling the functions ecatGetProcessImageInputPtr() and ecatGetProcessImageOutputPtr(). 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”. See 4.3.19 “ecatGetProcessImageInputPtr()” and 4.3.20 “ecatGetProcessImageOutputPtr()”.

3.4.1 Process Data Access Functions selection

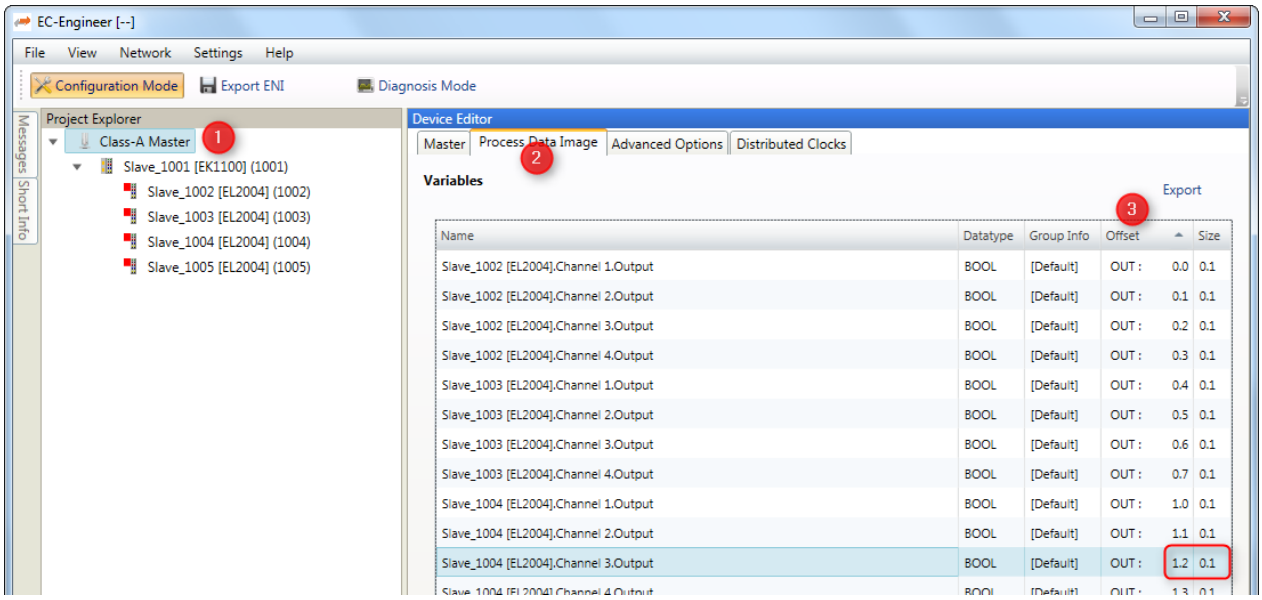
Process data variables that are packed as array of bits are bit aligned and not byte aligned in process data. See 4.4.1 “EC_COPYBITS” for how to copy data areas with bit offsets that are not byte aligned. Getting and setting bits that are bit aligned and not byte aligned should be done using EC_SETBITS and EC_GETBITS. Accessing complete EC_T_BYTE, EC_T_WORD, EC_T_DWORD, EC_T_QWORD can be accessed more efficiently using the appropriate macros according to the following table.

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

Variable type	Bit size	Macro	Chapter	Hint
EC_T_BYTE	8	N/A		Bytes are byte-aligend and can be directly addressed at pbyBuffer[BitOffset/8]
EC_T_WORD	16	EC_SET_FRM_WORD, EC_GET_FRM_WORD	4.4.2	Contains swap for big endian systems.
EC_T_DWORD	32	EC_SET_FRM_DWORD, EC_GET_FRM_DWORD	4.4.3	Contains swap for big endian systems.
EC_T_QWORD	64	EC_SET_FRM_QWORD, EC_GET_FRM_QWORD	4.4.4	Contains swap for big endian systems.
Bit	1	EC_SETBITS / EC_GETBITS	4.4.8	

3.4.2 Process variables’ offset and size

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



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

There are possible ways to get this offset. All offsets are given as **bit offsets**!

1. Working with a fixed offset:
   The offset value is figured out from the EtherCAT configuration tool. It's not recommended to use fixed values because the offsets changes in case of adding/removing slaves to/from the configuration.

   As listed in the screenshot above "Slave_1004 [EL2004].Channel 3.Output" in the example is at offset **1.2** with size **0.1**. The numbering is Byte.Bit so the offset in the example is Byte 1, Bit 2 means bit offset 8*__1__+__2__ = 10 and size is __0__*8+__1__ = 1.

   Sample code:

```
EC_T_BYTE byNewValue = 0x01;

/* set variable in process data */
EC_SETBITS(ecatGetProcessImageOutputPtr(), &byNewValue, /*offset*/10, /*size*/1);
```

2. Figure out the slave offset dynamically by calling the function ecatGetCfgSlaveInfo():
   The offsets are stored in EC_T_CFG_SLAVE_INFO.dwPdOffsIn and EC_T_CFG_SLAVE_INFO.dwPdOffsOut.
   E.g. setting "Slave_1004 [EL2004].Channel 3.Output" according to the screenshot above is like:

```
EC_T_BYTE byNewValue = 0x01;

EC_T_CFG_SLAVE_INFO SlaveInfo;
dwRes = ecatGetCfgSlaveInfo(EC_TRUE, 1004, &SlaveInfo);

/* set variable in process data */
EC_SETBITS(ecatGetProcessImageOutputPtr(), &byNewValue, SlaveInfo.dwPdOffsOut +
/*variable off*/2, /*variable size*/1);
```

   Figure out the variable offset by calling the function ecatFindInpVarByName() or ecatFindOutpVarByName():
   The offset is stored in EC_T_PROCESS_VAR_INFO.nBitOffs. Each input or output has a unique variable name. All variables names are stored in the ENI file under "EtherCATConfig/Config/ProcessImage/Inputs/Variable".
   E.g. setting "Slave_1004 [EL2004].Channel 3.Output" according to the screenshot above is like:

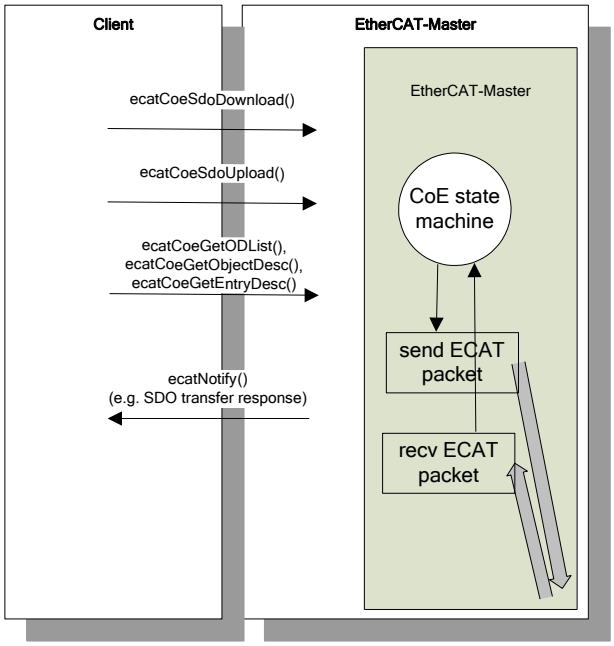
```
EC_T_BYTE byNewValue = 0x01;

EC_T_PROCESS_VAR_INFO VarInfo;
dwRes = ecatFindOutpVarByName("Slave_1004 [EL2004].Channel 3.Output", &VarInfo)

/* set variable in process data */
EC_SETBITS(ecatGetProcessImageOutputPtr(), &byNewValue, VarInfo.nBitOffs,
VarInfo.nBitSize);
```

3.5 CanOpen over EtherCAT transfers (CoE)

The EtherCAT client may use these services for example to access the object dictionary of a CoE slave.



The following services are supported:

- SDO download: SDO data transfer from the controller to a slave
- SDO upload: SDO data transfer from a slave to the controller
- SDO information service: read SDO object properties (object dictionary) from a slave
- Emergency Request

The CoE mailbox transfer is controlled by the master timer (application needs to call ecatExecJob(eUsrJob_MasterTimer) cyclically).
The client will be notified about a complete mailbox transfer using the generic ecatNotify() callback function.

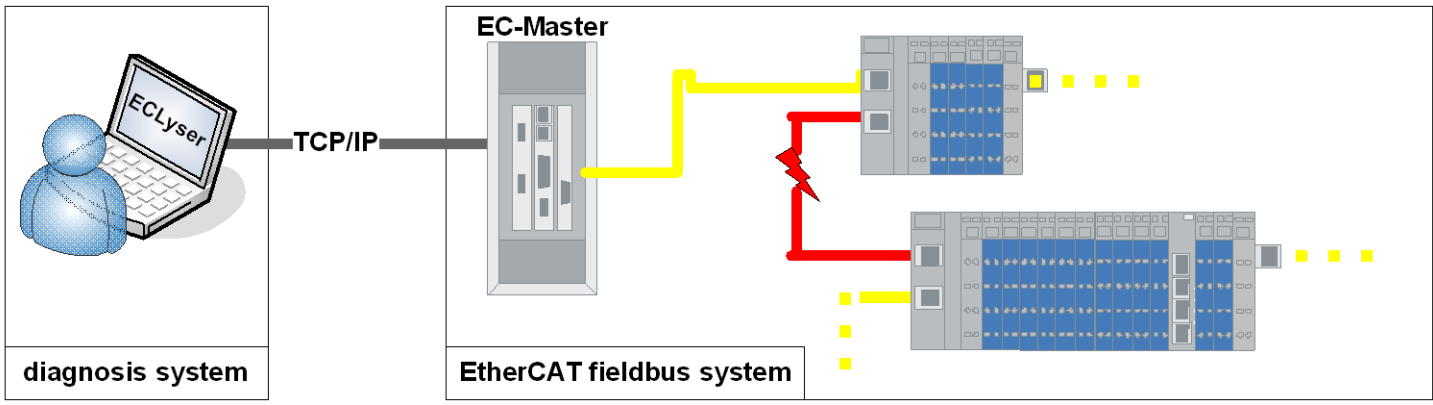
3.6 Error detection and diagnosis

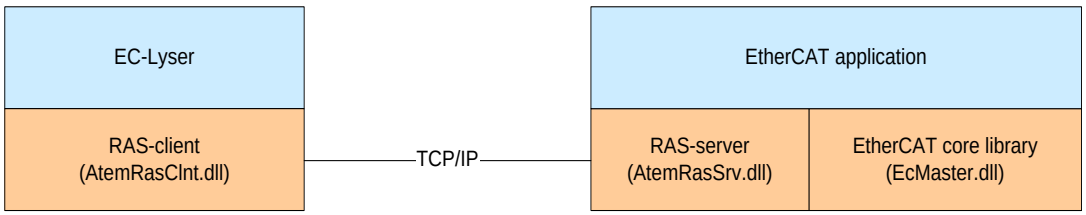
One of the parameters the client has to set when registering with the EtherCAT master is a generic notification callback function (ecatNotify). In case an error is detected the master will call this function.

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

3.7.1 Integration Requirements

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





An example on how to integrate the Remote API Server within your application is given with the example application EcMasterDemo, which in case is preconfigured to listen for EC-Lyser on TCP Port 6000 when commandline parameter “-sp” is given (→Further command line parameters).

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. Of course you need to link against the Remote API Server library “AtemRasSrv.lib” if using a DLL based OS or against “AtemRasSrv.a” if using an archive based toolchain (VxWorks, QNX or similar).

3.7.2 Pseudo Code

```
#include <AtEmRasSrv.h>

/* custom Remote API Notification handler, example in EcMasterDemo (ecatNotification.cpp) */
EC_T_DWORD emRasNotify(
    EC_T_DWORD          dwCode,     /**< [in]   Notification code identifier */
    EC_T_NOTIFYPARMS*   pParms      /**< [in]   Notification data portion */
                      )
{
    /* custom notification handler */
    …
}

/* your initialization function, creating master instance */
void InitFunction(void)
{
    ATEMRAS_T_SRVPARMS  oRemoteApiConfig = {0};
    EC_T_VOID           hRemApiHandle;
/* INADDR_ANY */
    oRemoteApiConfig.oAddr.dwAddr        = 0;
/*< default is port 6000 > */
    oRemoteApiConfig.wPort               = wServerPort;
/*< default is 20 msec */
    oRemoteApiConfig.dwCycleTime         = REMOTE_CYCLE_TIME;
/*< default is 10000 msec */
    oRemoteApiConfig.dwWDTOLimit         = (REMOTE_WD_TO_LIMIT/REMOTE_CYCLE_TIME);
/* Reconnect Timeout after 600*100msec = 60secs + 10secs */
    oRemoteApiConfig.dwReConTOLimit      = 60000;
    oRemoteApiConfig.dwMasterPrio        = ECAT_MAIN_THREAD_PRIO;
    oRemoteApiConfig.dwClientPrio        = ECAT_MAIN_THREAD_PRIO;
/* Notification context, if required*/
    oRemoteApiConfig.pvNotifCtxt         = pNotification;
/* Notification function for emras Layer */
    oRemoteApiConfig.pfNotification      = emRasNotify;
/* memory for concurrent notifications */
    oRemoteApiConfig.dwConcNotifyAmount  = 100;
/* memory for concurrent mailbox notifications */
    oRemoteApiConfig.dwMbxNotifyAmount   = 50;
/* 2K user space for Mailbox Notifications */
    oRemoteApiConfig.dwMbxUsrNotifySize  = 2000;
/* span between to consecutive cyclic notifications of same type */
    oRemoteApiConfig.dwCycErrInterval    = 500;

/* start remote API server */
    emRasSrvStart(oRemoteApiConfig, &hRemApiHandle);

/* initmaster, configure master, your hole application */
    emInitMaster(…);

…
…
…
/* end of custom application */
    emStop(…);

/* stop remote API server */
    emRasSrvStop(hRemApiHandle, 2000)

    emDeinitMaster(…);
}
```

3.7.3 Required API Calls

3.7.3.1 emRasSrvStart

Initializes and start remote API Server Instance.

```
EC_T_DWORD emRasSrvStart (
    ATEMRAS_T_SRVPARMS      oParms,
    EC_T_PVOID*             ppHandle
);
```

Parameters

oParms
[in] Parameter definitions

ppHandle
[out] Server Instance handle

Return

EC_E_NOERROR or error code

Comment

The Remote API Server will be initialized and started by calling this function.

ATEMRAS_T_SRVPARMS
Remote API Server initialization parameters.

```
typedef struct _ATEMRAS_T_SRVPARMS{
        ATEMRAS_T_IPADDR        oAddr;
        EC_T_WORD               wPort;
        EC_T_DWORD              dwCycleTime;
        EC_T_DWORD              dwWDTOLimit;
        EC_T_DWORD              dwReConTOLimit;
        EC_T_DWORD              dwMasterPrio;
        EC_T_DWORD              dwClientPrio;
        EC_T_DWORD              dwConcNotifyAmount;
        EC_T_DWORD              dwMbxNotifyAmount;
        EC_T_DWORD              dwMbxUsrNotifySize;
        EC_T_PVOID              pvNotifCtxt;
        EC_PF_NOTIFY            pfNotification;
        EC_T_DWORD              dwCycErrInterval;
} ATEMRAS_T_SRVPARMS;
```

Description

oAddr
- [in] IP Address to bind Remote API Server to (DWORD or 4Byte Array).

wPort
- [in] TCP Port to bind Remote API Server to.

dwCycleTime
- [in] Time in milliseconds which determines the timeout value of a poll cycle which either accepts a new connection or, if connection established, reads commands from the socket Interface. This is the maximum timeout data is processed asynchronuous when ready for read.

dwWDTOLimit
- [in] Amount of cycles (determined by dwCycleTime) before connection enters wdexpired state.

dwReConLimit
- [in] Amount of cycles (determined by dwCycleTime) after connection is in wdexpired state before connection is completely disconnected and resources allocated from master stack are freed.

dwMasterPrio
- [in] Priority of connection acceptor thread.

dwClientPrio
- [in] Priority of command receiver thread in an established connection state.

dwConcNotifyAmount
- [in] Amount of concurrently queueable Notifications (not Mailboxes).

dwMbxNotifyAmount
- [in] Amount of pre-allocated notification memory buffers used for mailbox notifications. The application can handle up to this amount of mailboxes simultaneously. For security reasons the actual used amount of mailboxes shall be slightly lower than dwMbxNotifyAmount.

dwMbxUsrNotifySize
- [in] User definable amount of bytes each mailbox notification buffer is enlarged off. This value should be at least the size of the largest transferred / used mailbox object.

pvNotifCtxt
- [in] Buffer to user defined data, which is passed to each call of Remote API Server Notification function.

pfNotification
- [in] Pointer to function which is called to notify change of state or errors.

dwCycErrInterval
- [in] Shortest amount of time in msec in between two cyclic error messages of the same kind are transferred to a remote client.

3.7.3.2 emRasSrvStop

Stop and de-initialize remote API Server Instance.

EC_T_DWORD emRasSrvStop(
EC_T_PVOID pvHandle,
EC_T_DWORD dwTimeout
);

Parameters
pvHandle
[in] Handle retrieved from emRasSrvStart
dwTimeout
[in] Timeout used to shut down all spawned threads. This timeout value is in msecs and multiplied internally by the amount of threads spawned.

Return
EC_E_NOERROR or error code

Comment
-

3.7.3.3 emrasNotify - xxx

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

```
EC_T_DWORD emrasNotify(
    EC_T_DWORD          dwCode,
    EC_T_NOTIFYPARMS*   pParms
);
```

Parameters

dwCode
[in] Notification code
pParms
[in] Notification code depending data

Return

EC_E_NOERROR or error code

Comment

EC_T_NOTIFYPARMS
Data structure filled with detailed information about the according notification.

```
typedef struct _EC_T_NOTIFYPARMS{
    EC_T_VOID*          pCallerData;
    EC_T_BYTE*          pbyInBuf;
    EC_T_DWORD          dwInBufSize;
    EC_T_BYTE*          pbyOutBuf;
    EC_T_DWORD          dwOutBufSize;
    EC_T_DWORD*         pdwNumOutData;
} EC_T_NOTIFYPARMS;
```

Description
pCallerData
[in] Client depending caller data parameter. This pointer is one of the parameters when the client registers with the master.
pbyInBuf
[in] Notification input parameters
dwInBufSize
[in] Size of the input buffer in bytes
pbyOutBuf
[out] Notification output (result)
dwOutBufSize
[in] Size of the output buffer in bytes
pdwNumOutData
[out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

3.7.3.4 emrasNotify – ATEMRAS_NOTIFY_CONNECTION

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

Parameters

pbyInBuf
[in] Pointer to data of type ATEMRAS_T_CONNOTIFYDESC
dwInBufSize
[in] Size of the input buffer in bytes
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

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

```
typedef struct _ATEMRAS_T_CONNOTIFYDESC{
    EC_T_DWORD          dwCause;
    EC_T_DWORD          dwCookie;
} ATEMRAS_T_CONNOTIFYDESC;
```

Description
dwCause
[in] Cause of state connection state change which is one of:

EC_E_NOERROR :	new logon
EMRAS_E_LOGONCANCELLED:	error during logon
EMRAS_EVT_RECONNECT:	resume of former connection
EMRAS_EVT_RECONEXPIRED:	re-connect failed due to long term timeout
EMRAS_EVT_SERVERSTOPPED:	RAS Server shutdown, re-connects impossible
EC_E_INVALIDSTATE:	if accepted socket object is invalid

dwCookie
[in] Unique identification cookie of connection instance.

3.7.3.5 emrasNotify – ATEMRAS_NOTIFY_REGISTER

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

Parameters

pbyInBuf
[in] Pointer to data of type ATEMRAS_T_REGNOTIFYDESC
dwInBufSize
[in] Size of the input buffer in bytes
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

ATEMRAS_T_REGNOTIFYDESC

```
typedef struct _ATEMRAS_T_REGNOTIFYDESC{
    EC_T_DWORD          dwCookie;
    EC_T_DWORD          dwResult;
    EC_T_DWORD          dwInstanceId;
    EC_T_DWORD          dwClientId;
} ATEMRAS_T_REGNOTIFYDESC;
```

Description
dwCookie
[in] Unique identification cookie of connection instance
dwResult
[in] Result of registration request
dwInstanceId
[in] Master Instance client registered to
dwClientId
[in] Client ID of registered client

3.7.3.6 emrasNotify – ATEMRAS_NOTIFY_UNREGISTER

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

Parameters

pbyInBuf
[in] Pointer to data of type ATEMRAS_T_REGNOTIFYDESC
dwInBufSize
[in] Size of the input buffer in bytes
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

ATEMRAS_T_REGNOTIFYDESC

```
typedef struct _ATEMRAS_T_REGNOTIFYDESC{
    EC_T_DWORD          dwCookie;
    EC_T_DWORD          dwResult;
    EC_T_DWORD          dwInstanceId;
    EC_T_DWORD          dwClientId;
} ATEMRAS_T_REGNOTIFYDESC;
```

Description
dwCookie
[in] Unique identification cookie of connection instance.
dwResult
[in] Result of un - registration request.
dwInstanceId
[in] Master Instance client un - registered from.
dwClientId
[in] Client ID of un - registered client.

3.7.3.7 emrasNotify – ATEMRAS_NOTIFY_MARSHALERROR

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

Parameters

pbyInBuf
[in] Pointer to data of type ATEMRAS_T_MARSHALERRORDESC
dwInBufSize
[in] Size of the input buffer in bytes
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

ATEMRAS_T_MARSHALERRORDESC

```
typedef struct _ATEMRAS_T_MARSHALERRORDESC{
    EC_T_DWORD          dwCookie;
    EC_T_DWORD          dwCause;
    EC_T_DWORD          dwLenStatCmd;
    EC_T_DWORD          dwCommandCode;
} ATEMRAS_T_MARSHALERRORDESC;
```

Description

dwCookie
[in] Unique identification cookie of connection instance
dwCause
[in] Error code
dwLenStaCmd
[in] Length of faulty command
dwCommandCode
[in] Command code of faulty command

3.7.3.8 emrasNotify – ATEMRAS_NOTIFY_ACKERROR

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

Parameters

pbyInBuf
[in] Pointer to EC_T_DWORD containing error code
dwInBufSize
[in] Size of the input buffer in bytes
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

–

3.7.3.9 emrasNotify – ATEMRAS_NOTIFY_NONOTIFYMEMORY

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

Parameters

pbyInBuf
[in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance
dwInBufSize
[in] Size of the input buffer in bytes
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

–

3.7.3.10 emrasNotify – ATEMRAS_NOTIFY_STDNOTIFYMEMORYSMALL

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

Parameters

pbyInBuf
[in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance
dwInBufSize
[in] Size of the input buffer in bytes
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

–

3.7.3.11 emrasNotify – ATEMRAS_NOTIFY_MBXNOTIFYMEMORYSMALL

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

Parameters

pbyInBuf
[in] Pointer to EC_T_DWORD containing unique identification cookie of connection instance
dwInBufSize
[in] Size of the input buffer in bytes
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

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.

3.8 EtherCAT Master Stack Source Code

In a source code delivery the master stack 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)
- Master stack files (configuration, core and interface layer)
- OS layer files (only valid for the master stack)

The master stack can be ported to several different operating systems and CPU architectures with different compilers and development environments.
There is no supported build environment shipped with the source code.

To build the master stack the appropriate build environment for the target operating system has to be used. If no integrated development environment is available makefiles and dependency rules may have to be created which contain the necessary master stack source and header files.
If an integrated development environment exists (like Wind River Workbench for VxWorks or Visual Studio for Windows CE) all necessary files have to be included into a project which for example may generate an EtherCAT master stack library (e.g. a DLL for Windows CE or a static library for VxWorks).

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

a) Link Layer Binary (e.g. a downloadable object moduel in VxWorks or a DLL in Windows). The Link Layer binary will be dynamically bound to the application at runtime. (currently not for On Time RTOS-32 and T-Kernel these use static libraries)
b) Master Stack Binary (a static library)
c) Remote API Server Binary (a static library)

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. emlll8255x.dll for the I8255x Link Layer).
- VxWorks: a downloadable kernel module (*.out) has to be created. The name of the module has to be emllXxxx.out where Xxxx shall be replaced by the Link Layer type (e.g. emlll8255x.out for the I8255x Link Layer). sysLoSalAdd.c should be included in BSP if needed and should not be compiled within the Link Layer binary
- Linux/QNX: a shared object library (*.so) has to be created.
- RTX a RTX dynamic link library (*.rtdll) has to be created. The name of the DLL has to be emllXxxx.dll where Xxxx shall be replaced by the Link Layer type (e.g. emlll8255x.dll for the I8255x Link Layer).
- INtime: a shared library (*.rsl) has to be created. The name of the RSL has to be emllXxxx.rsl where Xxxx shall be replaced by the Link Layer type (e.g. emlll8255x.rsl for the I8255x Link Layer).

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

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

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

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

The following pre-processor settings are required for the compiler:

- If a debug build shall be made the pre-processor macro DEBUG has to be defined (e.g. –DDEBUG for the gnu compiler)
- For **Windows** support WIN32 has to be defined
- For **Windows CE** support WIN32 and UNDER_CE have to be defined
- For **VxWorks** support VXWORKS has to be defined. If **VxWorks SMP** shall be used then additionally _WRS_VX_SMP has to be defined. If **VxWorks** with **Power PC** shall be used then additionally EC_BIG_ENDIAN has to be defined.
  GNU/PowerPC: you may have to set the –mlongcall compiler option to avoid relocation offset errors when downloading *.out files.
- For **On Time RTOS-32** support RTOS_32 has to be defined
- For **Linux** support LINUX has to be defined
- For **QNX Neutrino** support QNX6 has to be defined
- For **RTX** support UNDER_RTSS has to be defined
- For **INtime** support __INTIME__ has to be defined
- For **T-Kernel** support __TKERNEL has to be defined

Link Layer Binaries: the following include search paths have to be set

- Path to the SDK header files
- Path to the private header files
- Path to the Link Layer header files
- Path to the Link Layer sources

Master Stack Binaries: the following include search paths have to be set

- Path to the SDK header files
- Path to the private header files
- Path to the OS layer header files
- Path to the master stack sources

4 Application programming interface, reference

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

4.1 Generic API return status values

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

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

The EC_E_BUSY return status value indicates that a previously requested operation is still in progress. For example if the master is requested to enter the OPERATIONAL state the next request from the API will return this status value unless the OPERATIONAL state is entered.

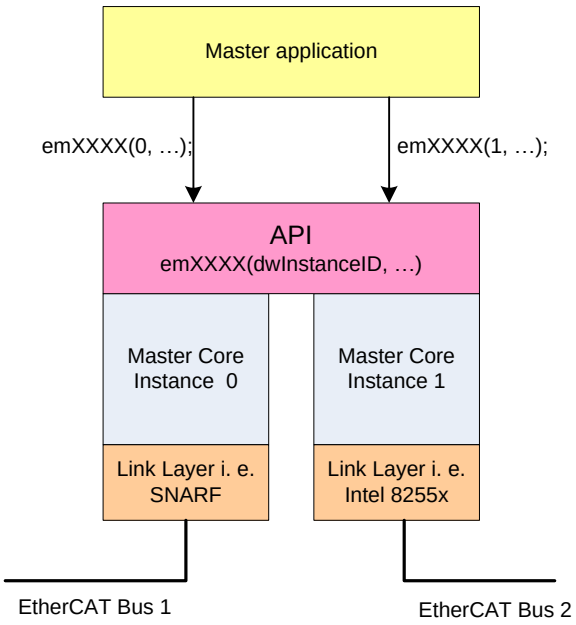
4.2 Multiple EtherCAT Bus Support

4.2.1 Licensing

Multiple EtherCAT Bus support is included within the Class B and Class A master stack.
For each bus a separate runtime license is required.
A single runtime allows the usage of the multi instance functions only with an instance identifier of 0.

4.2.2 Overview

The acontis EtherCAT master allows controlling more than one EtherCAT bus within one application process. For this use case the master core is instantiated several times by using the multi instance API functions inside the application.
Each API function is available as a single instance version (prefix ecat, e.g. ecatInitMaster) and a multi instance version (prefix em, e.g. emInitMaster).
The first parameter of all multi instance functions emXxx is the instance identifier.
The single instance functions ecatXxx will use the first master core instance with the identifier 0.
The maximum number of supported instances is 10.



4.2.3 Example application

The application EcMasterDemoMulti demonstrates a client application which handles two master instances with the following configuration (el9800.xml):

- Master instance 0: One Beckhoff EtherCAT Evaluation Board EL9800
- Master instance 1: One Beckhoff EtherCAT Evaluation Board EL9800

Parameters for this application:

```
-winpcap 192.168.1.32 1 -f el9800.xml @ -winpcap 192.168.2.32 1 -f el9800.xml
```

4.3 General functions

4.3.1 ecatInitMaster

Initializes the EtherCAT master stack.

```
EC_T_DWORD ecatInitMaster(
    EC_T_INIT_MASTER_PARMS*      pParms
);
```

Parameters

pParms

[in] Pointer to parameter definitions

Return

EC_E_NOERROR or error code, e.g.:
EC_E_NOTFOUND if Link Layer configured at **pLinkParms** can't be found or is not usable.

Comment

The EtherCAT master stack will be initialized by calling this function. This function has to be called prior to calling any other functions of the EtherCAT master.
This function may not be called from within the JobTask's context.

EC_T_INIT_MASTER_PARMS

```
typedef struct _EC_T_INIT_MASTER_PARMS
{
    EC_T_DWORD      dwSignature;
    EC_T_DWORD      dwSize;

    EC_T_OS_INIT_DESC* pOsParms;
    EC_T_LINK_PARMS*   pLinkParms;
    EC_T_LINK_PARMS*   pLinkParmsRed;
    EC_T_DWORD      dwBusCycleTimeUsec;

    /* memory */
    EC_T_DWORD      dwMaxBusSlaves;
    EC_T_DWORD      dwMaxQueuedEthFrames;
    EC_T_DWORD      dwAdditionalEoEEndpoints;

    /* bus load */
    EC_T_DWORD      dwMaxSlaveCmdPerFrame;
    EC_T_DWORD      dwMaxSentQueuedFramesPerCycle;

    /* CPU load */
    EC_T_DWORD      dwMaxSlavesProcessedPerCycle;

    /* retry and timeouts */
    EC_T_DWORD      dwEcatCmdMaxRetries;
    EC_T_DWORD      dwEcatCmdTimeout;
    EC_T_DWORD      dwEoETimeout;
    EC_T_DWORD      dwFoEBusyTimeout;

    /* VLAN */
    EC_T_BOOL       bVLANEnable;
    EC_T_WORD       wVLANId;
    EC_T_BYTE       byVLANPrio;

    /* logging */
    EC_T_DWORD      dwLogLevel;
    EC_T_PFLOGMSG_CB    pfLogMsgCallBack;
    EC_T_BOOL       bLogToLinkLayer;

    EC_T_DWORD      dwReserved[4];
} EC_T_INIT_MASTER_PARMS;
```

dwSignature
[in] Signature of the structure. The value ATECAT_SIGNATURE (defined in the header file AtEthercat.h) has to be assigned to this parameter.
dwSize
[in] Size of the structure. The value sizeof(EC_T_INIT_MASTER_PARMS) has to be assigned to this parameter.
pOsParms
[in] Pointer to the OS layer parameter structure EC_T_OS_INIT_DESC. This parameter will be given to the OS layer initialization function *OsInit*.
pvLinkParms
[in] Pointer to the Link Layer parameters (e.g. to determine which network adapter to use). The parameters are highly depending on the network adapter used by the Link Layer. More information can be found in chapter Link Layer selection and initialization.
pvLinkParmsRed
[in] Pointer to the Link Layer parameters for second network interface used for redundancy support. Set to EC_NULL if redundancy should not be used
dwBusCycleTimeUsec
[in] Bus cycle time in microseconds
dwMaxBusSlaves
[in] The maximum number of master internal pre-allocated bus slave objects. If there are more slaves connected to the bus then pre-allocated the error code of the bus scan notification (see EC_NOTIFY_SB_STATUS) is set to EC_E_MAX_BUS_SLAVES_EXCEEDED.
dwMaxQueuedEthFrames
[in] Maximum number of queued Ethernet frames. If the application (e.g. when calling mailbox services) or the master (e.g. when polling mailbox slaves) wants to send acyclic EtherCAT commands these commands are internally queued first. This value is internally limited to 127.
dwAdditionalEoEEndpoints
[in] Extend the virtual switch to this number of additional EoE endpoints. The default number of ports is equal to the number of EoE slaves and one EoE endpoint. Setting this value to 1 permit to register 2 EoE endpoints.
dwMaxSlaveCmdPerFrame
[in] Maximum number of EtherCAT commands stored in a single Ethernet frame. It is possible to store multiple EtherCAT commands in a single Ethernet frame to optimize the throughput and performance. For debugging purposes it may be useful to set this value to 1. A value of 0 will instruct the master to fill the Ethernet frame with as many EtherCAT commands as possible.
dwMaxSentQueuedFramesPerCycle
[in] This parameter determines (if set to a value greater than 0) the maximum number of queued (acyclic) frames that are sent with eUsrJob_SendAcycFrames within one cycle. Using this parameter the application is able to avoid that too much acyclic frames are sent between sending cyclic frames.
If the master is operational the number of frames sent by a call to ecatExecJob(eUsrJob_SendAcycFrames) thus will be limited to the value configured by dwMaxSentQueuedFramesPerCyc.
dwMaxSlavesProcessedPerCycle
[in] Maximum slave-related state machine calls per cycle
dwEcatCmdMaxRetries
[in] Maximum number of retries in case of timeouts when waiting for pending queued EtherCAT commands.
dwEcatCmdTimeout
[in] Queued (acyclic) EtherCAT command receive timeout (milliseconds).
If not set (equal=0), the default value is calculated by dwBusCycleTimeUsec/1000 * 20. In a well balanced system all queued (acyclic) frames can be sent and will be received and processed within one master cycle. The number of acyclic frames that are sent within one master cycle can be limited by the parameter dwMaxSentQueuedFramesPerCyc (see below). In such cases it may actually take multiple master cycles until the acyclic frame will be sent. The master verifies if the time between queuing a frame and receiving the frame does not exceed this timeout

value set in dwEcatCmdTimeout. The timeout value must therefore be higher than the time needed for one master cycle. The master cycle time is determined by the period between calling ecatExecJob(eUsrJob_MasterTimer)

dwEoeTimeout
[in] Timeout waiting for a EoE response.

dwFoeBusyTimeout
[in] obsolete.

bVLANEnable
[in] Set to EC_TRUE to activate VLAN support. VLAN support is not included within the standard EtherCAT master license agreement, thus the VLAN support has to be separately licensed. To activate VLAN support, the EC-Master Stack has to be compiled with the option VLAN_FRAME_SUPPORT set within the File EcType.h.

wVLANId
[in] 12Bit VLANId

byVLANPrio
[in] 3Bit VLAN priority

dwLogLevel
[in] Verbosity level. See `EC_LOG_LEVEL_…` below.

pfLogMsgCallBack
[in] Optional call-back function invoked by the Master to log messages.

bLogToLinkLayer
[in] Enable logging to EtherCAT network adapter.

```
typedef struct _EC_T_OS_INIT_DESC{
   EC_T_DWORD      dwTimerPriority;
   EC_PF_SYSTIME   pfSystemTimeGet;
   EC_T_PVOID      pvPlatformSpecific;
} EC_T_OS_INIT_DESC;
```

dwTimerPriority
[in] Priority of Timer Thread, spawned by OS Layer (when used).

pfSystemTimeGet
[in] Function which returns Host time in nanoseconds since 1st January 2000, which is used for DC Initialization, as time base.

pvPlatformSpecific
[in] Platform specific OS Layer initialization parameters.

EC_LOG_LEVEL_...
The following log levels are defined:

```
EC_LOG_LEVEL_CRITICAL   1
EC_LOG_LEVEL_ERROR      2
EC_LOG_LEVEL_WARNING    3
EC_LOG_LEVEL_INFO       4
EC_LOG_LEVEL_VERBOSE    5
```

4.3.2 ecatDeinitMaster

Terminates the EtherCAT master and releases all resources.

```
EC_T_DWORD ecatDeinitMaster(
    EC_T_VOID
);
```

Parameters

–

Return

EC_E_NOERROR or error code

Comment

Stops the EtherCAT master task and releases all resources.
This function may not be called from within the JobTask's context.

4.3.3 ecatGetMasterParms

Gets current Master Init Parameters provided by ecatInitMaster or ecatSetMasterParms.

EC_T_DWORD ecatGetMasterParms(
EC_T_INIT_MASTER_PARMS* pParms,
EC_T_DWORD dwParmsBufSize
);

Parameters
pParms
[out] Pointer to parameter definitions
dwParmsBufSize
[in] Size of buffer *pParms* points to.

Return
EC_E_NOERROR or error code, e.g.:
EC_E_INVALIDPARM if *pParms* is not usable.

Comment
Also fills OS parms, Main Link parms, Red Link parms if buffer at *pParms* is big enough according to *dwParmsBufSize*.

4.3.4 ecatSetMasterParms

Change Init Parameters provided by ecatInitMaster.
.
EC_T_DWORD ecatSetMasterParms(
EC_T_INIT_MASTER_PARMS* pParms
);

Parameters
pParms
[in] Pointer to parameter definitions

Return
EC_E_NOERROR or error code, e.g.:
EC_E_INVALIDPARM if *pParms* is not usable or a un-changeable parameter was modified, see below.

Comment
Currently OS parms, Main Link parms, Red Link parms, dwMaxBusSlaves, dwMaxQueuedEthFrames, dwAdditionalEoEEndpoints, bVLANEnable, wVLANId, byVLANPrio cannot be changed.

4.3.5 ecatScanBus

Scans all connected slaves
EC_T_DWORD ecatScanBus (
EC_T_DWORD dwTimeout
);

Parameters
dwTimeout
[in] Order timeout [ms]

Return
EC_E_NOERROR or error code

Comment
This function should not be called from within the JobTask's context.

4.3.6 ecatConfigureMaster

Configure the Master.

```
EC_T_DWORD ecatConfigureMaster(
    EC_T_CNF_TYPE           eCnfType,
    EC_T_PBYTE              pbyCnfData,
    EC_T_DWORD              dwCnfDataLen
);
```

Parameters

eCnfType
[in] Enum type of configuration data provided

pbyCnfData
[in] Configuration data, or EC_NULL if eCnfType is eCnfType_GenPreopENI

dwCnfDataLen
[in] Length of configuration data in byte, or zero if eCnfType is eCnfType_GenPreopENI

Return

EC_E_NOERROR or error code

Comment

This function has to be called after *ecatInitMaster* and prior to calling *ecatStart*. Among others the EtherCAT topology defined in the given XML configuration file will be stored internally.
Analyzing the network including Mailbox communication like CoE can be done without given ENI file using eCnfType_GenPreopENI. This is mainly used for configuration tools to get information about the slaves in order to create the ENI file used for ecatConfigureMaster(…).
Remark: A client must not be registered (ecatRegisterClient()) prior to calling this function. In such a case the client registration will be lost. A client may be registered directly after calling ecatInitMaster but has to be unregistered before calling ecatConfigureMaster.

```
typedef enum _EC_T_CNF_TYPE{
    eCnfType_Filename       /* data given marks a ENI file to read */
    eCnfType_Data     /* data ptr contains ENI data */
    eCnfType_Datadiag       /* data ptr contains ENI data for diagnosis */
    eCnfType_GenPreopENI    /* generate ENI based on bus-scan result to get into preop state*/
} EC_T_CNF_TYPE;
```

Depending on this enum the data field is interpreted differently.
This function may not be called from within the JobTask's context.

4.3.7 ecatRegisterClient

Register a client with the EtherCAT Master. It must not be called prior to ecatConfigureMaster otherwise the registration handle is lost. This function replaces the ecatIoControl – EC_IOCTL_REGISTERCLIENT.

```
EC_T_DWORD ecatRegisterClient(
    EC_PF_NOTIFY                pfnNotify,
    EC_T_VOID*                  pCallerData,
    EC_T_REGISTERRESULTS*       pRegResults
);
```

Parameters

pfnNotify

[in] Client's notification callback function. This function will be called by the master every time a state change occurs, an error occurs or a mailbox transfer terminates. See section 4.5.1 for more information.

pCallerData

[in] Pointer to a caller data area which will be returned back by the master to the client on every notification callback. This value will not be interpreted by the master.

pRegResults

[out] Registration results, a pointer to a structure of type EC_T_REGISTERRESULTS.

Return

EC_E_NOERROR or error code

Comment

EC_T_REGISTERRESULTS
Contain the registration results.

```
typedef struct _EC_T_REGISTERRESULTS{
    EC_T_DWORD          dwClntId;
    EC_T_BYTE *         pbyPDIn;
    EC_T_DWORD          dwPDInSize;
    EC_T_BYTE *         pbyPDOut;
    EC_T_DWORD          dwPDOutSize;
} EC_T_REGISTERRESULTS;
```

Description

dwClntId
[out] Client ID

pbyPDIn
[out] Pointer to process data input memory

dwPDInSize
[out] Size of process data input memory (in bytes)

pbyPDOut
[out] Pointer to process data output memory

dwPDOutSize
[out] Size of process data output memory (in bytes)

In the XML configuration file the data offsets into the process data memory will be determined (Config/Cyclic/Frame/Cmd/InputOffs respectively OutputOffs). Usually I/O data starts at an offset of 0x10. This function may not be called from within the JobTask's context.

4.3.8 ecatUnregisterClient

Unregister a client from the EtherCAT master. This function replaces the ecatIoControl – EC_IOCTL_UNREGISTERCLIENT.

```
EC_T_DWORD ecatUnregisterClient(
    EC_T_DWORD        dwClientId
);
```

Parameters

dwClientId

[in] Client ID determined when registering with the master.

Return

EC_E_NOERROR or error code

Comment

This function may not be called from within the JobTask's context.

4.3.9 ecatGetSrcMacAddress

Gets the source MAC address.

```
EC_T_DWORD ecatGetSrcMacAddress(
    ETHERNET_ADDRESS* pSrcMacAddress
);
```

Parameters

pSrcMacAddress

[out] 6-byte buffer to write source MAC address to.

Return

EC_E_NOERROR or error code

Comment

Refers to adapter from EC_T_INIT_MASTER_PARMS.pLinkParms at 4.3.1 "ecatInitMaster".

4.3.10 ecatSetMasterState

The EtherCAT master (and all slaves) will be set into the requested state.

```
EC_T_DWORD ecatSetMasterState(
    EC_T_DWORD      dwTimeout,
    EC_T_STATE      eReqState
);
```

Parameters

dwTimeout

[in] Timeout in msec. This function will block until the requested state is reached or the timeout elapsed. If the timeout value is set to EC_NOWAIT the function will return immediately.

eReqState

[in] Requested State.

eEcatState_INIT	Master state Init
eEcatState_PREOP	Master state pre-operational
eEcatState_SAFEOP	Master state safe operational
eEcatState_OP	Master state operational (if called first, same as ecatStart)

Return

EC_E_NOERROR or error code. If the last requested state was not reached, EC_E_BUSY will be returned and the request has to be repeated at a later time.

Comment

If the function is called with EC_NOWAIT, the client may wait for reaching the requested state using the notification callback (EC_NOTIFY_STATECHANGED)
This function may not be called from within the JobTask's context.

4.3.11 ecatGetMasterState

The current master state is returned.

```
EC_T_STATE ecatGetMasterState(EC_T_VOID);
```

Parameters

-

Return

*Current master state (*eEcatState_INIT, eEcatState_PREOP, eEcatState_SAFEOP or eEcatState_OP).

Comment

-

4.3.12 ecatStart

The EtherCAT master (and all slaves) will be set into the OPERATIONAL state.

```
EC_T_DWORD ecatStart(
    EC_T_DWORD        dwTimeout
);
```

Parameters

dwTimeout

[in] Timeout in msec. This function will block until the OPERATIONAL state is reached or the timeout elapsed. If the timeout value is set to EC_NOWAIT the function will return immediately.

Return

EC_E_NOERROR or error code

Comment

If the function is called with EC_NOWAIT, the client may wait for reaching the OPERATIONAL state using the notification callback (EC_NOTIFY_STATECHANGED).
This function may not be called from within the JobTask's context.

4.3.13 ecatStop

The EtherCAT master and all slaves will be set back into the INIT state.

```
EC_T_DWORD ecatStop(
    EC_T_DWORD        dwTimeout
);
```

Parameters

dwTimeout

[in] Timeout in msec. This function will block until the INIT state is reached or the timeout elapsed. If the timeout value is set to EC_NOWAIT the function will return immediately.

Return

EC_E_NOERROR or error code. If the last requested state was not reached, EC_E_BUSY will be returned and the request has to be repeated at a later time.

Comment

If the function is called with EC_NOWAIT, the client may wait for reaching the INIT state using the notification callback (**ECAT_NOTIFY_STATECHANGE**).
Caveat: in multithreading systems ecatStop() must be executed only after after all other EtherCAT operations (e.g. mailbox transfers) on all slaves are completely terminated.
This function may not be called from within the JobTask's context.

4.3.14 ecatExecJob

Execute or initiate the requested master job.

```
EC_T_DWORD ecatExecJob(
    EC_T_USER_JOB    EUserJob,
    EC_T_VOID*       pvParam
);
```

Parameters

EUserJob

[in] Requested job.

pvParam

[in] Additional job specific parameter. This parameter is reserved if not otherwise specified below.

Return

EC_E_NOERROR or error code, depending on the requested job (see below).
EC_BUSY will be returned in case one of the requests eUsrJob_ProcessAllRxFrames, eUsrJob_SendAllCycFrames, eUsrJob_SendCycFramesByTaskId or eUsrJob_SendAcycFrames would be executed concurrently.
EC_BUSY will be returned in case one of the requests eUsrJob_MasterTimer would be executed concurrently.

Important note:

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 ecatExecJob(), the calls have to be synchronized externally.

Comment

The following EUserJob requests are currently defined:

a) 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.
pvParam: Pointer to a flag of EC_T_BOOL type (not used if set to EC_NULL).
If this flag is set to a value of EC_TRUE it indicates that all previously initiated cyclic frames (eUsrJob_SendAllCycFrames) are received and processed within this call.
Return: EC_E_NOERROR if successful, error code in case of failures.

b) eUsrJob_SendAllCycFrames
Send all cyclic frames. New values will be written to the EtherCAT slave's outputs and new input values will be received. If the Link Layer operates in interrupt mode, the process data input values will be updated immediately after receiving the frames. If the Link Layer operates in polling mode, the next call to ecatExecJob with the eUsrJob_ProcessAllRxFrames job will check for received frames and update the process data input values.
Return: EC_E_NOERROR if successful, error code in case of failures. In case not all previously initiated cyclic frames are processed when calling this function an error notification will be generated (EC_NOTIFY_FRAME_RESPONSE_ERROR).

c) eUsrJob_SendAcycFrames
Acyclic EtherCAT datagrams stored in the acyclic frame buffer FIFO will be sent when executing this call.
pvParam: Pointer to EC_T_DWORD type (not used if set to EC_NULL).
Indicates number of frames send within this call.
Return: EC_E_NOERROR if successful, error code in case of failures.

d) eUsrJob_RunMcSm
This job is obsolete.

e) eUsrJob_MasterTimer
To trigger the master and slave state machines as well as the mailbox handling this call has to be executed cyclically. The master cycle time is determined by the period between calling ecatExecJob(eUsrJob_MasterTimer). The state-machines are handling the EtherCAT state change transfers.
Return: EC_E_NOERROR if successful, error code in case of failures.

f) eUsrJob_SendCycFramesByTaskId
Send cyclic frames related to a specific task id. If more than one cyclic entries are configured (see Configuration variant 2: multiple cyclic entries) this user job can be used to send the appropriate cyclic frames. All frames stored in cyclic entries with the given task id will be sent.
pvParam: Pointer to a EC_T_DWORD type value with the task id.
Return: EC_E_NOERROR if successful, error code in case of failures. If not all previously initiated cyclic frames for the same task are already processed when calling this function an error will be generated (EC_NOTIFY_FRAME_RESPONSE_ERROR).

g) eUsrJob_StampSendAllCycFrames
same as b) but the transmitted frame is timestamped for Distributed Clocks, if a callback is registered and an ARMW is contained.

h) eUsrJob_StampSendCycFramesByTaskId
same as f) but the transmitted frame is timestamped for Distributed Clocks, if a callback is registered and an ARMW is contained.

4.3.15 ecatGetProcessData

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).
This function must not be called from within the job task (dead lock!).

```
EC_T_DWORD ecatGetProcessData(
    EC_T_BOOL       bOutputData,
    EC_T_DWORD      dwOffset,
    EC_T_BYTE*      pbyData,
    EC_T_DWORD      dwLength,
    EC_T_DWORD      dwTimeout);
```

Parameters

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 to hold read data.
dwLength
[in] Size of pbyData Buffer and length of data to read.
dwTimeout
[in] Timeout in msec.

Return

EC_E_NOERROR or error code

Comment

This function may not be called from within the JobTask's context.

4.3.16 ecatSetProcessData

Write Process data synchronized. . If process data shall be set outside the cyclic master job task (which is calling ecatExecJob), direct access to the process data is not recommended as data consistency cannot be guaranteed. A call to this function will send a data write request to the master stack and then check every millisecond whether new data is written. The master stack will copy the data after calling ecatExecJob(*eUsrJob_* MasterTimer) within the job task. This function is usually only called remotely (using the Remote API).
This function must not be called from within the job task (dead lock!).

```
EC_T_DWORD ecatSetProcessData(
    EC_T_BOOL       bOutputData,
    EC_T_DWORD      dwOffset,
    EC_T_BYTE*      pbyData,
    EC_T_DWORD      dwLength,
    EC_T_DWORD      dwTimeout);
```

Parameters

bOutPutData
[in] EC_TRUE: write output data, EC_FALSE: write input data.
dwOffset
[in] Byte offset in Process data to write to.
pbyData
[in] Buffer to hold write data.
dwLength
[in] Size of pbyData Buffer and length of data to write.

dwTimeout
[in] Timeout in msec.

Return
EC_E_NOERROR or error code

Comment
This function may not be called from within the JobTask's context.

4.3.17 ecatSetProcessDataBits

Writes a specific number of bits from a given buffer to the process image with a bit offset (synchronized). This function is handled in the same way as function ecatSetProcessData and must not be called from within the job task (dead lock).

```
EC_T_DWORD ecatSetProcessDataBits(
    EC_T_BOOL       bOutputData,
    EC_T_DWORD      dwBitOffsetPd,
    EC_T_BYTE*      pbyDataSrc,
    EC_T_DWORD      dwBitLengthSrc,
    EC_T_DWORD      dwTimeout);
```

Parameters
bOutputData
[in] EC_TRUE: write output data, EC_FALSE: write input data.
dwBitOffsetPd
[in] Bit offset in Process data image.
pbyDataSrc
[in] Data that shall be written to the process image
dwBitLengthSrc
[in] Number of bits that shall be written to the process image
dwTimeout
[in] Timeout in msec. The timeout value must not be set to EC_NOWAIT.

Return
EC_E_NOERROR or error code

Comment
This function may not be called from within the JobTask's context.

4.3.18 ecatGetProcessDataBits

Reads a specific number of bits from the process image to the given buffer with a bit offset (synchronized). This function is handled in the same way as function ecatGetProcessData and must not be called from the job task (dead lock).

```
EC_T_DWORD ecatGetProcessDataBits(
    EC_T_BOOL       bOutputData,
    EC_T_DWORD      dwBitOffsetPd,
    EC_T_BYTE*      pbyDataDst,
    EC_T_DWORD      dwBitLengthDst,
    EC_T_DWORD      dwTimeout);
```

Parameters
bOutputData
[in] EC_TRUE: write output data, EC_FALSE: write input data.
dwBitOffsetPd
[in] Bit offset in Process data image.
pbyDataDst
[out] Data buffer for the read data
dwBitLengthDst

[in] Number of bits that shall be read from the process image

dwTimeout

[in] Timeout in msec. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

Comment

This function may not be called from within the JobTask's context.

4.3.19 ecatGetProcessImageInputPtr

Gets the process data input image pointer

EC_T_BYTE* ecatGetProcessDataInputPtr(EC_T_VOID);

Return

The process data input image pointer or EC_NULL

4.3.20 ecatGetProcessImageOutputPtr

Gets the process data output image pointer

EC_T_BYTE* ecatGetProcessDataOutputPtr(EC_T_VOID);

Return

The process data output image pointer or EC_NULL

4.3.21 ecatGetDiagnosisImagePtr

Gets the diagnosis image pointer

EC_T_BYTE* ecatGetDiagnosisImagePtr(EC_T_VOID);

Return

The diagnosis image pointer or EC_NULL

4.3.22 ecatForceProcessDataBits

Forces a specific number of bits from a given buffer to the process image with a bit offset.
All output data set by this API are overwriting the values set by the application. All input data set by this API are overwriting the values read from the slaves.
Forcing will be terminiated by calling the corresponding functions ecatReleaseProcessDataBits() or ecatReleaseAllProcessDataBits().
This function is handled in the same way as function ecatSetProcessData and must not be called from within the job task (dead lock).

```
EC_T_DWORD ecatForceProcessDataBits (
    EC_T_DWORD      dwClientId,
    EC_T_BOOL       bOutputData,
    EC_T_DWORD      dwBitOffsetPd,
    EC_T_WORD       wBitLength,
    EC_T_BYTE*      pbyData,
    EC_T_DWORD      dwTimeout);
```

Parameters

dwClientId

[in] Client ID determined when registering with the master.

bOutputData

[in] EC_TRUE: write output data, EC_FALSE: write input data.

dwBitOffsetPd

[in] Bit offset in Process data image.
wBitLength
[in] Number of bits that shall be written to the process image.
pbyData
[in] Data that shall be written to the process image.
dwTimeout
[in] Timeout in msec. The timeout value must not be set to EC_NOWAIT.

Return
EC_E_NOERROR or error code

Comment
This function may not be called from within the JobTask's context.

4.3.23 ecatReleaseProcessDataBits

Release previously forced process data.
For a forced output: Value set by application become valid again. Because ecatForceProcessDataBits() writes directly into the process output image, the application has to update the process image with the required value, otherwise the forced value is still valid.
For a forced input: Value read from the slaves become valid again.
This function is handled in the same way as function ecatSetProcessData and must not be called from within the job task (dead lock).

```
EC_T_DWORD ecatReleaseProcessDataBits (
    EC_T_DWORD      dwClientId,
    EC_T_BOOL       bOutputData,
    EC_T_DWORD      dwBitOffsetPd,
    EC_T_WORD       wBitLength,
    EC_T_DWORD      dwTimeout);
```

Parameters
dwClientId
[in] Client ID determined when registering with the master.
bOutputData
[in] EC_TRUE: write output data, EC_FALSE: write input data.
dwBitOffsetPd
[in] Bit offset in Process data image.
wBitLength
[in] Number of bits that shall be written to the process image.
dwTimeout
[in] Timeout in msec. The timeout value must not be set to EC_NOWAIT.

Return
EC_E_NOERROR or error code

Comment
This function may not be called from within the JobTask's context.

4.3.24 ecatReleaseAllProcessDataBits

Release all previously forced process data for a dedicated client.
For a forced output: Value set by application become valid again. Because ecatForceProcessDataBits() writes directly into the process output image, the application has to update the process image with the required value, otherwise the forced value is still valid.
For a forced input: Value read from the slaves become valid again.
This function is handled in the same way as function ecatSetProcessData and must not be called from within the job task (dead lock).

```
EC_T_DWORD ecatReleaseAllProcessDataBits (
    EC_T_DWORD      dwClientId,
```

EC_T_DWORD dwTimeout);

Parameters
dwClientId
[in] Client ID determined when registering with the master.
dwTimeout
[in] Timeout in msec. The timeout value must not be set to EC_NOWAIT.

Return
EC_E_NOERROR or error code

Comment
This function may not be called from within the JobTask's context.

4.3.25 ecatGetVersion

Gets the EC-Master's version as EC_T_DWORD.

EC_T_DWORD ecatGetVersion(EC_T_DWORD* pdwVersion);

Parameters
pdwVersion
[out] Pointer to EC_T_DWORD to carry out EC-Master's version.

Return
EC_E_NOERROR or error code

4.3.26 ecatSetLicenseKey

Sets the license key for the protected version.

EC_T_DWORD ecatSetLicenseKey(EC_T_CHAR* szKey);

Parameters
szKey
[in] License key as zero terminated string with 26 characters.

Comment
Must be called after ecatInitMaster() and before ecatConfigureMaster().

Example:
```
dwRes = ecatSetLicenseKey("DA1099F2-15C249E9-54327FBC");
```

4.3.27 ecatSetOemKey

Provide OEM Key needed for OEM Masters to parse ENI files and provide access via RAS.

EC_T_DWORD ecatSetOemKey(EC_T_UINT64 qwOemKey);

Parameters
qwOemKey
[in] 64 bit OEM key.

Comment
Must be called after ecatInitMaster() and before ecatConfigureMaster().

Example:

```
dwRes = ecatSetOemKey(0x1234567812345678);
```

4.3.28 ecatIoControl

With ecatIoControl() a generic control interface exists between the application and the EC-Master and its Link Layers.

```
EC_T_DWORD ecatIoControl(
    EC_T_DWORD          dwCode,
    EC_T_IOCTLPARMS*    pParms
);
```

Parameters

dwCode
[in] Control code

pParms
[in] Control code depending parameters

Return

EC_E_NOERROR or error code

Comment

EC_T_IOCTLPARMS
Control parameters, the content is depending on the control code.

```
typedef struct _EC_T_IOCTLPARMS{
    EC_T_BYTE*          pbyInBuf;
    EC_T_DWORD          dwInBufSize;
    EC_T_BYTE*          pbyOutBuf;
    EC_T_DWORD          dwOutBufSize;
    EC_T_DWORD *        pdwNumOutData;
} EC_T_IOCTLPARMS;
```

Description

pbyInBuf
[in] Pointer to control input parameter.

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

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

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.

4.3.29 ecatIoControl – EC_IOCTL_GET_PDMEMORYSIZE

Queries the master for the necessary size the process data image has got. This information may be used to provide process data image storage from outside the master core. This IOCTL is to be called after ecatConfigureMaster and before ecatStart.

Parameters

pbyInBuf
[] Should be set to EC_NULL.
dwInBufSize
[] 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.

Comment

–

EC_T_MEMREQ_DESC
Structure containing the necessary memory sizes in bytes.

```
typedef struct _EC_T_MEMREQ_DESC{
    EC_T_DWORD          dwPDOutSize,
    EC_T_DWORD          dwPDInSize
} EC_T_MEMREQ_DESC;
```

4.3.30 ecatIoControl – EC_IOCTL_REGISTER_PDMEMORYPROVIDER

This function call registers an external memory provider to the EtherCAT master, this memory will be used to store process data. If no memory provider is registered the master will internally allocate the necessary amount of memory.
The function ecatIoControl - EC_IOCTL_GET_PDMEMORYSIZE should be executed to determine the amount of memory the master needs to store process data values.
The external memory provider may additionally supply some hooks to give the master a possibility to synchronize memory access with the application.
The memory provider has to be registered after calling ecatConfigureMaster() but prior to registering any client and prior to calling ecatStart(). Every client that registers with the master (ecatRegisterClient()) will get back the memory pointers to PDOut/PDIn data registered within this call.

Parameters

pbyInBuf
[in] Memory provider (EC_T_MEMPROV_DESC)
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

Return

EC_E_NOERROR or error code.
EC_E_INVALIDSIZE: length of data buffer too small.

EC_T_MEMPROV_DESC

Descriptor of the memory provider.

```
typedef struct _EC_T_MEMPROV_DESC{
    EC_T_PVOID          pvContext;
    EC_T_BOOL           bUseMasterShadowInUserSyncNowaitMode; (obsolete)

    EC_T_PBYTE          pbyPDOutData;
    EC_T_DWORD          dwPDOutDataLength;
    EC_T_PBYTE          pbyPDInData;
    EC_T_DWORD          dwPDInDataLength;

    EC_T_PFMEMREQ       pfPDOutDataReadRequest;
    EC_T_PFMEMREL       pfPDOutDataReadRelease;
    EC_T_PFMEMREQ       pfPDInDataWriteRequest;
    EC_T_PFMEMREL       pfPDInDataWriteRelease;
} EC_T_MEMPROV_DESC;

typedef EC_T_VOID (*EC_T_PFMEMREQ)(EC_T_PVOID pvContext, EC_T_BYTE** ppbyPDData);
typedef EC_T_VOID (*EC_T_PFMEMREL)(EC_T_PVOID pvContext);
```

Description

pvContext

[in] Context pointer. This pointer is used every time when one of the callback functions (e.g. pfPDOutReadRequest) is called.

bUseMasterShadowInUserSyncNowaitMode

[in] This parameter is obsolete and has to be set to EC_FALSE.

Fixed buffers:

pbyPDOutData

[in] Pointer to the fixed output process data buffer (values transferred from the master to the slaves). A value of EC_NULL may be given in case the pointer will be provided later when function pfPDOutDataReadRequest() is called.

dwPDOutDataLength

[in] Length of the output process data buffer. The function ecatIoControl - EC_IOCTL_GET_PDMEMORYSIZE should be executed to determine the appropriate size.

pbyPDInData

[in] Pointer to the fixed input process data buffer (values transferred from the slaves to the master). A value of EC_NULL may be given in case the pointer will be provided later when function pfPDInDataWriteRequest() is called.

dwPDInDataLength

[in] Length of the output process data buffer. The function ecatIoControl - EC_IOCTL_GET_PDMEMORYSIZE should be executed to determine the appropriate size.

Callback functions:

These functions will be called by the master within the cyclic process data transfer. The application has to assure that these functions will not block.

EC_T_VOID pfPDOutReadRequest(EC_T_PVOID pvContext, EC_T_BYTE** ppbyPDOutData)

[in] This function will be called by the master cyclically within the process data transfer cycle prior to read data from the output process data buffer. If pfPDOutReadRequest is set to EC_NULL the fixed buffer pointer pbyPDOutData will be used.

Parameters:

pvContext

[in] Context pointer, see above.

ppbyPDOutData

[out] Pointer to the output process data buffer to be used. If the function returns EC_NULL the master will use the fixed buffer pointer pbyPDOutData. The provided buffer size must be at least dwPDOutDataLength bytes.

EC_T_VOID pfPDOutReadRelease(EC_T_PVOID pvContext)

[in] This function will be called by the master cyclically within the process data transfer cycle after all data were read from the output process data buffer.

Parameters:

pvContext

[in] Context pointer, see above.

EC_T_VOID pfPDInWriteRequest(EC_T_PVOID pvContext, EC_T_BYTE** ppbyPDInData)

[in] This function will be called by the master cyclically within the process data transfer cycle prior to write new data into the input process data buffer. If pfPDInWriteRequest is set to EC_NULL the fixed buffer pointer pbyPDInData will be used.

Parameters:

pvContext

[in] Context pointer, see above.

ppbyPDInData

[out] Pointer to the input process data buffer to be used. If the function returns EC_NULL the master will use the fixed buffer pointer pbyPDInData. The provided buffer size must be at least dwPDInDataLength bytes.

EC_T_VOID pfPDInWriteRelease(EC_T_PVOID pvContext)

[in] This function will be called by the master cyclically within the process data transfer cycle after all data were written into the input process data buffer.

Parameters:

pvContext

[in] Context pointer, see above.

4.3.31 ecatIoControl – 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 memory provider has to be registered after calling ecatInitMaster() before starting the job task.

Parameters

pbyInBuf
[in] Cyclic frame received callback descriptor (EC_T_CYCFRAME_RX_CBDESC)
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

Return

EC_E_NOERROR or error code.
EC_E_INVALIDSIZE: length of data buffer too small.

EC_T_CYCFRAME_RX_CBDESC
Descriptor of the cyclic frame received callback.

```
typedef struct _EC_T_CYCFRAME_RX_CBDESC {
    EC_T_VOID*                  pCallbackParm;
    EC_PF_CYCFRAME_RECV         pfnCallback;
} EC_T_CYCFRAME_RX_CBDESC;

typedef EC_T_VOID (*EC_PF_CYCFRAME_RECV)(EC_T_VOID*);
```

Description

pvCallbackParm
[in] Context pointer. This pointer is used as parameter every time when the callback function is called.

EC_T_VOID pfCallback(EC_T_VOID* pvCallbackParm)
[in] This function will be called by the master after the cyclic frame is received. (if more than one cyclic frame after the last frame). The application has to assure that these functions will not block. Typically a event is given to wakeup the job task to continue.

Parameters:
pvCallbackParm
[in] Context pointer, see above.

4.3.32 ecatIoControl – EC_IOCTL_ISLINK_CONNECTED

Determine whether link between the EtherCAT master and the first slave is connected.

Parameters

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

Comment

With Redundancy support enabled, EC_FALSE is only set if main **and** redundancy link are down.

4.3.33 ecatIoControl – EC_IOCTL_GET_LINKLAYER_MODE

This call allows the application to determine whether the LinkLayer is currently running in polling or in interrupt mode.

Parameters

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

Comment

EC_T_LINKLAYER_MODE_DESC

```
typedef struct _EC_T_LINKLAYER_MODE_DESC{
    EC_T_LINKMODE       eLinkMode;
    EC_T_LINKMODE       eLinkModeRed;
} EC_T_LINKLAYER_MODE_DESC;
```

Description

eLinkMode
[in] Operation mode of currently used Link Layer, which maybe one of *EcLinkMode_INTERRUPT* or *EcLinkMode_POLLING*.
eLinkModeRed
[in] Operation mode of currently used Redundancy Interface. If no redundant interface is used *EcLinkMode_UNDEFINED* is returned.

4.3.34 ecatIoControl – EC_IOCTL_GET_CYCLIC_CONFIG_INFO

Determine cyclic configuration details from ENI configuration file.

Parameters

pbyInBuf
[in] Pointer to dwCycEntryIndex: cyclic entry index for which to get information
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[out] Pointer to EC_T_CYC_CONFIG_DESC data type.
dwOutBufSize
[in] Size of the output buffer provided at *pbyOutBuf* in bytes.
pdwNumOutData
[out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Comment

EC_T_CYC_CONFIG_DESC
Cyclic configuration descriptor.

```
typedef struct _EC_T_CYC_CONFIG_DESC {
    EC_T_DWORD      dwNumCycEntries;    /* total number of cyclic entries */
    EC_T_DWORD      dwTaskId;           /* task id of selected cyclic entry */
    EC_T_DWORD      dwPriority;         /* priority of selected cyclic entry */
    EC_T_DWORD      dwCycleTime;        /* cycle time of selected cyclic entry */
} EC_T_CYC_CONFIG_DESC;
```

4.3.35 ecatIoControl – EC_IOCTL_IS_SLAVETOSLAVE_COMM_CONFIGURED

Determine if any slave to slave communication is configured.

Parameters

pbyInBuf
[] Should be set to EC_NULL.
dwInBufSize
[] 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.

4.3.36 ecatIoControl – EC_LINKIOCTL_...

The generic control interface ecatIoControl() provides access to the main network adapter when adding EC_IOCTL_LINKLAYER_MAIN to the EC_LINKIOCTL_... parameter at dwCode. EC_IOCTL_LINKLAYER_RED specifies the redundant network adapter.

Parameters
pbyInBuf, dwInBufSize, pbyOutBuf, *dwOutBufSize, pdwNumOutData* are specific to the EC_LINKIOCTL_....

4.3.37 ecatIoControl – EC_LINKIOCTL_GET_ETHERNET_ADDRESS

Provides MAC addresses of main or red line.

Parameters
pbyInBuf
[] Should be set to EC_NULL.
dwInBufSize
[] 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.

4.3.38 ecatIoControl – EC_LINKIOCTL_GET_SPEED

Returns actual speed of main or red line in MBits at *pbyOutBuf*.

Parameters
pbyInBuf
[] Should be set to EC_NULL.
dwInBufSize
[] 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.

4.3.39 ecatIoControl – EC_IOCTL_SET_CYCFRAME_LAYOUT

Set the cyclic frames layout.

Parameters
pbyInBuf
[in] Pointer to a EC_T_CYCFRAME_LAYOUT value containing the cyclic frame layout.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

EC_T_CYCFRAME_LAYOUT
Cyclic configuration descriptor.

```
typedef enum _EC_T_CYCFRAME_LAYOUT
{
    eCycFrameLayout_STANDARD,
    /* Layout according ENI with command add/reordering, no relationship to PD */

    eCycFrameLayout_DYNAMIC,
    /* Layout is dynamically modified to send as less as possible cyclic frames and commands */

    eCycFrameLayout_FIXED,
    /* Layout strictly match ENI, frame buffers and PD area overlapped */

    eCycFrameLayout_IN_DMA,
    /* Layout strictly match ENI, frame buffers and PD area overlapped, frame buffers in DMA */
} EC_T_CYCFRAME_LAYOUT;
```

4.3.40 ecatIoControl – EC_IOCTL_SET_MASTER_DEFAULT_TIMEOUTS

Set master default timeouts in milliseconds.

Parameters

pbyInBuf
[in] Pointer to EC_T_MASTERDEFAULTTIMEOUTS_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

Comment

EC_T_MASTERDEFAULTTIMEOUTS_DESC
Master default timeouts descriptor.

```
typedef struct _EC_T_MASTERDEFAULTTIMEOUTS_DESC
{
    EC_T_DWORD    dwMasterStateChange;    /* Timeout to SetMasterState if called with EC_NOWAIT */
    EC_T_DWORD    dwInitCmdRetry;         /* Timeout to retry InitCmds */
    EC_T_DWORD    dwMbxCmd;               /* Timeout to retry mailbox commands */
    EC_T_DWORD    dwMbxPolling;           /* Mailbox polling cycle */
    EC_T_DWORD    dwReserved[12];

} EC_T_MASTERDEFAULTTIMEOUTS_DESC;
```

Setting a value of this descriptor to zero resets the default timeout value to the initial value.

4.3.41 ecatIoControl – EC_IOCTL_SET_COPYINFO_IN_SENDCYCFRAMES

Set copy info processed in either *SendCycFrames* or in *ProcessAllRxFrames*.

Parameters

pbyInBuf
[in] Pointer to EC_T_BOOL. EC_TRUE: *SendCycFrames,* EC_FALSE*: ProcessAllRxFrames*

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

pbyOutBuf
[] Should be set to EC_NULL.

dwOutBufSize
[] Should be set to 0.

pdwNumOutData
[] Should be set to EC_NULL.

Comment

Default: Set by *ProcessAllRxFrames*.

4.3.42 ecatIoControl – EC_IOCTL_SET_BUS_CYCLE_TIME

Set bus cycle time in usec master parameter without calling ecatInitMaster again.

Parameters

pbyInBuf
[in] Pointer to value of EC_T_DWORD. Value may not be 0!

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

pbyOutBuf
[] Should be set to EC_NULL.

dwOutBufSize
[] Should be set to 0.

pdwNumOutData
[] Should be set to EC_NULL.

Comment

Implicitly recalculates Order Timeout and EcatCmdTimeout.

4.3.43 ecatIoControl – EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES

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

Parameters

pbyInBuf
[in] Pointer to value of EC_T_BOOL. EC_TRUE: enable, EC_FALSE: disable.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

Comment

Additional variables are added to the process image for the following data types:

- FSOE_4096
- FSOE_4098
- FSOE_4099
- FB Info 1
- FB Info 3

4.3.44 ecatIoControl – EC_IOCTL_SLV_ALIAS_ENABLE

Enables slave alias addressing for all slaves.

Parameters

pbyInBuf
[in] Should be set to EC_NULL.
dwInBufSize
[in] Should be set to 0.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

Comment

Caution: All slaves need to have the correct alias address set! If in doubt, don't use this IOCTL.

4.4 Process Data Access Functions

4.4.1 EC_COPYBITS

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

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

Parameters

pbyDst
[out] destination buffer
nDstBitOffs
[in] bit offset within destination buffer
pbySrc
[in] source buffer
nSrcBitOffs
[in] bit offset in source buffer
nBitSize
[in] block size in bits

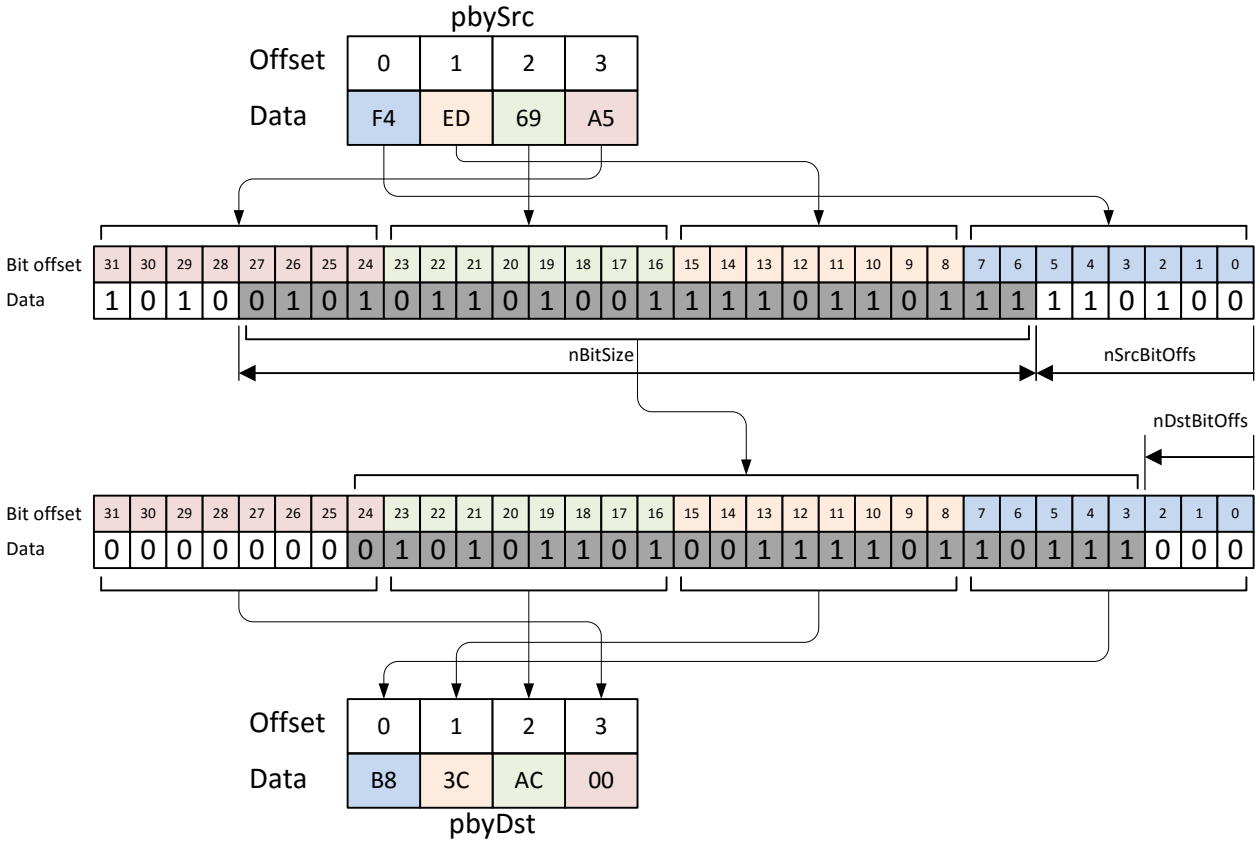
Return

-

Comment

The memory buffers must be allocated before. ***Important:*** The buffers must be big enough to hold the block starting at the given offsets! The buffers are not checked for overrun.
See 4.4.8 EC_SETBITS and 4.4.9 EC_GETBITS.

Example




```
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 */
```

4.4.2 EC_GET_FRM_WORD

Returns a value of type “EC_T_WORD” (16 bits) at given pointer. The value is swapped on PPC systems.

EC_GET_FRM_WORD(ptr)

Parameters

ptr

[in] source buffer

Return

EC_T_WORD value (16 bits) from buffer pointed by *ptr*.

Comment

The pointer has to point to a EC_T_WORD value (2 bytes).

Examples

```
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 */
```

4.4.3 EC_GET_FRM_DWORD

Returns a value of type “EC_T_DWORD” (32 bits) at given pointer. The value is swapped on PPC systems.

EC_GET_FRM_DWORD(ptr)

Parameters

ptr

[in] source buffer

Return

EC_T_DWORD value (32 bits) from buffer pointed by *ptr*.

Comment

The pointer has to point to a EC_T_DWORD value (4 bytes).

Examples

```
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 */
```

4.4.4 EC_GET_FRM_QWORD

Returns a value of type “EC_T_QWORD” (64 bits) at given pointer. The value is swapped on PPC systems.

EC_GET_FRM_QWORD(ptr)

Parameters

ptr

[in] source buffer

Return

EC_T_QWORD value (64 bits) from buffer pointed by *ptr*.

Comment

The pointer has to point to a EC_T_QWORD value (8 bytes).

Examples

```
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 */
```

4.4.5 EC_SET_FRM_WORD

Writes a value of type “EC_T_WORD” (16 bits) at given pointer. The value is swapped on PPC systems.

EC_SET_FRM_WORD(ptr,w)

Parameters

ptr

[in] destination buffer

w

[in] value of type “EC_T_WORD”

Return

-

Comment

The pointer has to point to a buffer to hold at least EC_T_WORD (2 bytes).

Examples

```
EC_T_BYTE byFrame[32];

/* Initialize the frame buffer */
OsMemset(byFrame, 0xFF, 32);

EC_SET_FRM_WORD(byFrame + 1, 0x1234);
/* byFrame = FF 34 12 FF FF FF … */
```

4.4.6 EC_SET_FRM_DWORD

Writes a value of type “EC_T_DWORD” (32 bits) at given pointer. The value is swapped on PPC systems.

EC_SET_FRM_DWORD(ptr,dw)

Parameters

ptr
[in] destination buffer
dw
[in] value of type “EC_T_DWORD”

Return

-

Comment

The pointer has to point to a buffer to hold at least EC_T_DWORD (4 bytes).

Examples

```
EC_T_BYTE byFrame[32];

/* Initialize the frame buffer */
OsMemset(byFrame, 0xFF, 32);

EC_SET_FRM_DWORD(byFrame + 1, 0x12345678);
/* byFrame = FF 78 56 34 12 FF … */
```

4.4.7 EC_SET_FRM_QWORD

Writes a value of type “EC_T_QWORD” (64 bits) at given pointer. The value is swapped on PPC systems.

EC_SET_FRM_QWORD(ptr,qw)

Parameters

ptr
[in] destination buffer
qw
[in] value of type “EC_T_QWORD”

Return

-

Comment

The pointer has to point to a buffer to hold at least EC_T_QWORD (8 bytes).

Examples

```
EC_T_BYTE byFrame[32];

/* Initialize the frame buffer */
OsMemset(byFrame, 0xFF, 32);

EC_SET_FRM_QWORD(byFrame + 1, 0xFEDCBA9876543210);
/* byFrame = FF 10 32 54 76 98 BA DC FE FF FF … */
```

4.4.8 EC_SETBITS

Copies nBitSize bits from pbySrcData starting at first bit to pbyDstBuf starting at nDstBitOffs.

EC_SETBITS(pbyDstBuf, pbySrcData, nDstBitOffs, nBitSize)

Parameters

pbyDstBuf
[out] destination where data is copied to (write!)
pbySrcData
[in] source data to be copied (read), starting with first bit
nDstBitOffs
[in] skipped bits at pbyDstBuf
nBitSize
[in] bit count to be copied
)

Comment

This function should be used to get bit aligned data, not byte aligned data as the byte aligned Process Data Functions are faster. See e.g. 4.4.2 EC_GET_FRM_WORD.

4.4.9 EC_GETBITS

Copies nBitSize bits from pbySrcBuf starting at nSrcBitOffs to pbyDstData starting at first bit.

EC_GETBITS(pbySrcBuf, pbyDstData, nSrcBitOffs, nBitSize)

Parameters

pbySrcBuf,
[in] source where data is copied from (read)
pbyDstData
[out] destination where data is copied to (write!), starting with first bit
nSrcBitOffs
[in] skipped bits at pbySrcBuf
nBitSize
[in] bit count to be copied

Comment

This function should be used to get bit aligned data, not byte aligned data as the byte aligned Process Data Functions are faster. See e.g. 4.4.2 EC_GET_FRM_WORD.

4.5 Generic notification interface

One of the parameters the client has to set when registering with the EtherCAT master is a generic notification callback function (ecatNotify). The master calls this function every time a event (for example an error event) occurs about which the client has to be informed (see also section 4.5.1).

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.

This callback function is usually called in the context of the EtherCAT master timer thread or the EtherCAT Link Layer receiver thread. It may also be called within the context of a user thread (when calling an EtherCAT master function).
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.

4.5.1 Notification callback: ecatNotify

When a client registers with the EtherCAT master the client has to determine a generic notification callback function. The master calls this function every time an event (for example an error event or operational state change event) occurs about which the client has to be informed.
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 ecatNotify() may be called.
A further important rule exists due to the fact that this callback function is usually called in the context of the EtherCAT master timer thread. As the whole EtherCAT operation is blocked while calling this function the notification handler must not use much CPU time or even call operating system functions that may block. Time consuming operations should be executed in separate application threads.

```
EC_T_DWORD ecatNotify(
    EC_T_DWORD          dwCode,
    EC_T_NOTIFYPARMS*   pParms
);
```

Parameters

dwCode
[in] Notification code
pParms
[in] Notification code depending data

Return

EC_E_NOERROR or error code

Comment

EC_T_NOTIFYPARMS
Data structure filled with detailed information about the according notification.

```
typedef struct _EC_T_NOTIFYPARMS{
    EC_T_VOID*          pCallerData;
    EC_T_BYTE*          pbyInBuf;
    EC_T_DWORD          dwInBufSize;
    EC_T_BYTE*          pbyOutBuf;
    EC_T_DWORD          dwOutBufSize;
    EC_T_DWORD*         pdwNumOutData;
} EC_T_NOTIFYPARMS;
```

Description
pCallerData
[in] Client depending caller data parameter. This pointer is one of the parameters when the client registers with the master.
pbyInBuf
[in] Notification input parameters.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[out] Notification output (result).
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.

4.5.2 ecatNotify – EC_NOTIFY_STATECHANGED

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

Parameters

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

[] Set to EC_NULL.

dwOutBufSize

[] Set to 0.

pdwNumOutData

[] Set to EC_NULL.

Comment

EC_T_STATECHANGE

Data structure containing the old and the new EtherCAT master operational state.

```
typedef struct _EC_T_STATECHANGE{
    EC_T_STATE          oldState;   /* old operational state */
    EC_T_STATE          newState;  /* new operational state */
} EC_T_STATECHANGE;
```

4.5.3 ecatNotify – EC_NOTIFY_XXXX (error notification)

Notification about an error. See 4.7 "Diagnosis, error detection, error notifications" for more information.

Parameters

pbyInBuf

[in] Pointer to data of type EC_T_NOTIFICATION_DESC.
A detailed description of all error notifications and the corresponding data can be found in 4.7 "Diagnosis, error detection, error notifications".

dwInBufSize

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

pbyOutBuf

[] Set to EC_NULL.

dwOutBufSize

[] Set to 0.

pdwNumOutData

[] Set to EC_NULL.

4.5.4 ecatNotify – EC_NOTIFY_MBOXRCV (mailbox notification)

Indicates a mailbox transfer completion.

The notification EC_NOTIFY_MBXRCV_INVALID_DATA is obsolete and not generated any more.

See section 4.8.5 for more information.

4.5.5 ecatNotifyApp

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

```
EC_T_DWORD ecatNotifyApp(
    EC_T_DWORD          dwCode,
    EC_T_NOTIFYPARMS*   pParms
    );
```

Parameters

dwCode

[in] Application specific notification code

pParms

[in] Parameters for application notification.

Return

EC_E_NOERROR or error code

Comment

The maximum value for *dwCode* is defined by EC_NOTIFY_APP_MAX_CODE.

4.5.6 ecatIoControl – EC_IOCTL_SET_NOTIFICATION_ENABLED

The following notifications can be enabled or disabled using EC_IOCTL_SET_NOTIFICATION_ENABLED

- EC_NOTIFY_SLAVE_STATECHANGED (default *Off*)
- EC_NOTIFY_SLAVES_STATECHANGED (default *Off*)
- EC_NOTIFY_SLAVE_UNEXPECTED_STATE (default *On*)
- EC_NOTIFY_SLAVES_UNEXPECTED_STATE (default *Off*)
- EC_NOTIFY_SLAVE_PRESENCE (default *On*)
- EC_NOTIFY_SLAVES_PRESENCE (default *Off*)
- EC_NOTIFY_SLAVE_ERROR_STATUS_INFO (default *On*)
- EC_NOTIFY_SLAVES_ERROR_STATUS (default *Off*)
- EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL (default *On*)
- EC_NOTIFY_CYCCMD_WKC_ERROR (default *On*)
- EC_NOTIFY_SB_MISMATCH (default *On*)
- EC_NOTIFY_SB_STATUS (default *On*)
- EC_NOTIFY_STATUS_SLAVE_ERROR (default *On*)
- EC_NOTIFY_FRAME_RESPONSE_ERROR (default *On*)
- EC_NOTIFY_HC_TOPOCHGDONE (default *On*)
- EC_NOTIFY_STATECHANGED (default *On*)
- EC_NOTIFY_COE_INIT_CMD (default *Off*)
- EC_NOTIFY_JUNCTION_RED_CHANGE (default *Off*)
- EC_NOTIFY_ALL_DEVICES_OPERATIONAL (default *Off*)
- EC_NOTIFY_DC_STATUS (default *On*)
- EC_NOTIFY_DC_SLV_SYNC (default *On*)
- EC_NOTIFY_DCM_SYNC (default *On*)
- EC_NOTIFY_SLAVE_INITCMD_RESPONSE_ERROR (default *On*)
- EC_NOTIFY_REFCLOCK_PRESENCE (default *Off*)

Parameters

pbyInBuf

[in] pointer to EC_T_SET_NOTIFICATION_ENABLED_PARMS.

dwInBufSize

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

pbyOutBuf

[] Should be set to EC_NULL.

dwOutBufSize

[] Should be set to 0.

pdwNumOutData
[] Should be set to EC_NULL.

Comment

```
typedef struct _EC_T_SET_NOTIFICATION_ENABLED_PARMS
{
   EC_T_DWORD      dwClientId;     /*< Client ID, 0: Master */
   EC_T_DWORD      dwCode;         /*< Notification code */
   EC_T_DWORD      dwEnabled;      /*< EC_NOTIFICATION_DISABLED, …_ENABLED, …_DEFAULT */
} EC_T_SET_NOTIFICATION_ENABLED_PARMS;
```

Notifications are given to clients if enabled for dwClientId = 0 AND corresponding dwClientId.

4.5.7 ecatIoControl – EC_IOCTL_GET_NOTIFICATION_ENABLED

The enabled state of notifications can be retrieved using EC_IOCTL_GET_NOTIFICATION_ENABLED. See EC_IOCTL_SET_NOTIFICATION_ENABLED for the list of supported notifications.

Parameters

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.

Comment

```
typedef struct _EC_T_GET_NOTIFICATION_ENABLED_PARMS
{
   EC_T_DWORD        dwClientId;      /*< Client ID, 0: Master */
   EC_T_DWORD        dwCode;          /*< Notification code */
} EC_T_GET_NOTIFICATION_ENABLED_PARMS;
```

4.6 Slave control and status functions

4.6.1 ecatGetNumConfiguredSlaves

Returns amount of slaves which are configured in the XML configuration file.

```
EC_T_DWORD ecatGetNumConfiguredSlaves (
    EC_T_VOID
);
```

Parameters

–

Return

Amount of slaves which are configured in the XML configuration file.

4.6.2 ecatGetNumConnectedSlaves

Returns amount of currently connected slaves.

```
EC_T_DWORD ecatGetNumConnectedSlaves (
    EC_T_VOID
);
```

Parameters

–

Return

Amount of currently connected slaves.

4.6.3 ecatGetSlaveId

Determines the slave ID using the slave station address set in the XML configuration file.

```
EC_T_DWORD ecatGetSlaveId (
    EC_T_WORD wStationAddress
);
```

Parameters

wStationAddress

[in] The slave's node address (set in the XML configuration file).

Return

Slave ID (INVALID_SLAVE_ID in case the slave does not exist)

Comment

The station address or the alias address is given here. A slave must have a station address larger than 0. If for example a mailbox transfer is going to be issued, the station address 0 returns the "pseudo" slave Identifier to address the master stack directly, which is needed in case of a master stack Object Dictionary access or SDO upload (see ➔ history objects in master stack 0x10f3, 0x10f4).

4.6.4 ecatGetSlaveIdAtPosition

Determines the slave ID using the slave auto increment address set in the XML configuration file.

```
EC_T_DWORD ecatGetSlaveIdAtPosition (
    EC_T_WORD wAutoIncAddress
);
```

Parameters

wAutoIncAddress
[in] The slave's auto increment address (set in the XML configuration file).

Return

Slave ID (INVALID_SLAVE_ID in case the slave does not exist)

4.6.5 ecatGetSlaveState

Get the slave's state.

```
EC_T_DWORD ecatGetSlaveState(
    EC_T_DWORD      dwSlaveId,
    EC_T_WORD*      pwCurrDevState,
    EC_T_WORD*      pwReqDevState
);
```

Parameters

dwSlaveId
[in] Slave ID
pwCurrDevState
[out] Current operating state
pwReqDevState
[out] Set value of the operating state

Return

EC_E_NOERROR on success.
EC_E_SLAVE_NOT_PRESENT if the slave is not present on bus.
EC_E_NOTFOUND if the slave with ID dwSlaveId does not exist in the XML file.
Error code otherwise.

Comment

The following state definitions (bit mask) exist: DEVICE_STATE_INIT, DEVICE_STATE_PREOP, DEVICE_STATE_BOOTSTRAP, DEVICE_STATE_SAFEOP, DEVICE_STATE_OP and DEVICE_STATE_ERROR.
If the state is not known the value DEVICE_STATE_UNKNOWN will be returned.

Important hint

It is also allowed to use this function inside the notification callback routine. The EC-Master reads the AL_STATUS register whenever needed automatically, disregarding if "ecatGetSlaveState" was called.

4.6.6 ecatSetSlaveState

Set a specified slave into the requested state.

```
EC_T_DWORD ecatSetSlaveState(
    EC_T_DWORD      dwSlaveId,
    EC_T_WORD       wNewReqDevState,
    EC_T_DWORD      dwTimeout
);
```

Parameters

dwSlaveId
[in] Slave ID
wNewReqDevState
[in] Requested state
dwTimeout
[in] Timeout in ms until the state should have been reached. Maybe set to EC_NOWAIT.

Return

EC_E_NOERROR if successful.
EC_E_BUSY if the master cannot execute the request at this time, the function has to be called at a later time.
EC_E_NOTFOUND if the slave with ID dwSlaveId does not exist.
EC_E_NOTREADY if the working counter was not set when requesting the slave's state (slave may not be connected or did not respond).
EC_E_TIMEOUT if the slave did not enter the requested state in time.
EC_E_INVALIDSTATE if the master denies the requested state, see comments below.
EC_E_INVALIDPARM if BOOTSTRAP was requested for a slave that does not support it.

Comment

The following state definitions exist: DEVICE_STATE_INIT, DEVICE_STATE_PREOP, DEVICE_STATE_BOOTSTRAP, DEVICE_STATE_SAFEOP and DEVICE_STATE_OP.
The requested state shall not be higher than the Master's operational state, e.g. the EC-Master denies requests to DEVICE_STATE_SAFEOP while the Master's operational state is PREOP.

DEVICE_STATE_BOOTSTRAP can only be requested if the slave's state is INIT.

Caveat: in multithreading systems ecatSetSlaveState() must be executed only after all other EtherCAT operations (e.g. mailbox transfers) on this slave are completely terminated.

This function may not be called from within the JobTask's context.

4.6.7 ecatIsSlavePresent

Returns whether a specific slave is currently connected to the Bus. This function may be called from within the JobTask. Since Slave Id is a parameter, valid response only can be retrieved after calling ecatConfigureMaster.

```
EC_T_DWORD ecatIsSlavePresent (
    EC_T_DWORD              dwSlaveId,
    EC_T_BOOL *             pbPresence
);
```

Parameters

dwSlaveId
[in] Slave ID.

pbPresence
[out] Pointer to Bool value: EC_TRUE if slave is currently connected to the bus, EC_FALSE if not.

Return

EC_E_NOERROR if successful.
EC_E_INVALIDSTATE if the master is not initialized.
EC_E_NOTFOUND if the slave with ID dwSlaveId does not exist or no ENI File was loaded.

Comment

-

4.6.8 ecatGetSlaveProp

Determines the properties of the slave device.

```
EC_T_BOOL ecatGetSlaveProp(
    EC_T_DWORD              dwSlaveId,
    EC_T_SLAVE_PROP*        pSlaveProp
);
```

Parameters

dwSlaveId
[in] Slave ID.
pSlaveProp
[out] Slave properties.

```
typedef struct _EC_T_SLAVE_PROP{
        EC_T_WORD       wStationAddress;    /* station address */
        EC_T_WORD       wAutoIncAddr;       /* auto increment address */
        EC_T_CHAR       achName[];          /* name of the slave device
                                               (NULL terminated string) */
} EC_T_SLAVE_PROP;
```

Return

EC_TRUE if the slave with ID dwSlaveId exists, *EC_FALSE* if this slave does not exist.

Comment

–

4.6.9 ecatGetSlavePortState

Returns the state of the slave ports.

```
EC_T_DWORD ecatGetSlavePortState(
    EC_T_DWORD              dwSlaveId,
    EC_T_WORD*              pwPortState
);
```

Parameters

dwSlaveId
[in] Slave ID.
pwPortState
[out] Slave port state.
Format: wwww xxxx yyyy zzzz
(each nibble : port 3210)
wwww : Signal detected 1=yes, 0=no
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)
(Bus Topology Scan)

Return

EC_E_NOERROR if successful.
EC_E_INVALIDSTATE if master is not initialized
EC_E_NOTFOUND if the slave with ID dwSlaveId does not exist.

Comment

–

4.6.10 ecatSlaveSerializeMbxTfers

All mailbox transfers to the spezified slave are serialized. The parallel (overlapped) usage of more than one protocol (CoE, EoE, FoE, etc.) is blocked by the master.

```
EC_T_DWORD ecatSlaveSerializeMbxTfers (
    EC_T_DWORD          dwSlaveId,
);
```

Parameters

dwSlaveId
[in] Slave ID

Return

EC_E_NOERROR if successful.
EC_E_INVALIDPARM if master is not initialized
EC_E_NOTFOUND if the slave with ID dwSlaveId does not exist.
EC_E_NO_MBX_SUPPORT if slave does not support mailbox transfers

Comment

By default parallel mailbox transfers are enabled.

4.6.11 ecatSlaveParallelMbxTfers

Reenable the parallel (overlapped) usage of more than one protocol (CoE, EoE, FoE, etc.).

```
EC_T_DWORD ecatSlaveParallelMbxTfers (
    EC_T_DWORD          dwSlaveId,
);
```

Parameters

dwSlaveId
[in] Slave ID

Return

EC_E_NOERROR if successful.
EC_E_INVALIDPARM if master is not initialized
EC_E_NOTFOUND if the slave with ID dwSlaveId does not exist.
EC_E_NO_MBX_SUPPORT if slave does not support mailbox transfers

Comment

By default parallel mailbox transfers are enabled.

4.6.12 ecatGetSlaveInpVarInfoNumOf

Gets the number of input variables of a specific slave. This function mainly will be used in connection with ecatGetSlaveInpVarInfo.

```
EC_T_DWORD ecatGetSlaveInpVarInfoNumOf(
   EC_T_BOOL          bFixedAddressing,
   EC_T_WORD          wSlaveAddress,
   EC_T_WORD*         pwSlaveInpVarInfoNumOf );
```

Parameters

bFixedAddressing
[in] *EC_TRUE: Use station address, EC_FALSE: use AutoInc address*
wSlaveAddress
[in] Slave address
pwSlaveInpVarInfoNumOf
[out] Number of found process variables

Return

EC_E_NOERROR or error code

4.6.13 ecatGetSlaveOutpVarInfoNumOf

Gets the number of output variables of a specific slave. This function mainly will be used in connection with ecatGetSlaveOutpVarInfo.

```
EC_T_DWORD ecatGetSlaveOutpVarInfoNumOf(
   EC_T_BOOL          bFixedAddressing,
   EC_T_WORD          wSlaveAddress,
   EC_T_WORD*         pwSlaveOutpVarInfoNumOf );
```

Parameters

bFixedAddressing
[in] *EC_TRUE: Use station address, EC_FALSE: use AutoInc address*
wSlaveAddress
[in] Slave address
pwSlaveInpVarInfoNumOf
[out] Number of found process variable

Return

EC_E_NOERROR or error code

4.6.14 ecatGetSlaveInpVarInfo

Gets the input process variable information entries of a specific slave.

```
EC_T_DWORD ecatGetSlaveInpVarInfo(
   EC_T_BOOL                bFixedAddressing,
   EC_T_WORD                wSlaveAddress,
   EC_T_WORD                wNumOfVarsToRead,
   EC_T_PROCESS_VAR_INFO* pSlaveProcVarInfoEntries,
   EC_T_WORD*               pwReadEntries);
```

Parameters

bFixedAddressing
[in] EC_TRUE: Use station address, EC_FALSE: use AutoInc address
wSlaveAddress
[in] Slave address
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

Return

EC_E_NOERROR or error code

Comment

–

EC_T_PROCESS_VAR_INFO

```
typedef struct _EC_T_PROCESS_VAR_INFO
{
   EC_T_CHAR          szName[MAX_PROCESS_VAR_NAME_LEN];
   EC_T_WORD          wDataType;
   EC_T_INT           nBitSize;
   EC_T_INT           nBitOffs;
   EC_T_WORD          wFixedAddr;
   EC_T_BOOL          bIsInputData;
} EC_T_PROCESS_VAR_INFO, *EC_PT_PROCESS_VAR_INFO;
```

Description

szName
[out] Name of the found process variable
wDataType
[out] Data type of the found process variable (according to ETG.1000, section 5). See also EcCommon.h, DEFTYPE_BOOLEAN, …. .
nBitSize
[out] Size in bit of the found process variable
nBitOffs
[out] Bit offset in the process data image
wFixedAddr
[out] Station address of the slave that is owner of this variable
bIsInputData
[out] Determines whether the found process variable is an input variable or an output variable

4.6.15 ecatGetSlaveInpVarInfoEx

Gets the input process variable extended information entries of a specific slave.

```
EC_T_DWORD ecatGetSlaveInpVarInfoEx(
    EC_T_BOOL                   bFixedAddressing,
    EC_T_WORD                   wSlaveAddress,
    EC_T_WORD                   wNumOfVarsToRead,
    EC_T_PROCESS_VAR_INFO_EX*   pSlaveProcVarInfoEntriesEx,
    EC_T_WORD*                  pwReadEntries);
```

Parameters

bFixedAddressing
[in] EC_TRUE: Use station address, EC_FALSE: use AutoInc address
wSlaveAddress
[in] Slave address
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

Return

EC_E_NOERROR or error code

Comment

```
typedef struct _EC_T_PROCESS_VAR_INFO_EX
{
   EC_T_CHAR        szName[MAX_PROCESS_VAR_NAME_LEN_EX];
   EC_T_WORD        wDataType;
   EC_T_INT         nBitSize;
   EC_T_INT         nBitOffs;
   EC_T_WORD        wFixedAddr;
   EC_T_BOOL        bIsInputData;
   EC_T_WORD        wIndex;
   EC_T_WORD        wSubIndex;
   EC_T_WORD        wPdoIndex;
   EC_T_WORD        wWkcStateDiagOffs;
   EC_T_DWORD       dwRes1;
   EC_T_DWORD       dwRes2;
} EC_T_PROCESS_VAR_INFO_EX, *EC_PT_PROCESS_VAR_INFO_EX;
```

Description

szName
[out] Name of the found process variable
wDataType
[out] Data type of the found process variable
nBitSize
[out] Size in bit of the found process variable
nBitOffs
[out] Bit offset in the process data image
wFixedAddr
[out] Station address of the slave that is owner of this variable
bIsInputData
[out] Determines whether the found process variable is an input or an output variable
wIndex
[out] Object index
wSubIndex
[out] Object sub index
wPdoIndex
[out] Index of PDO (process data object)
wWkcStateDiagOffs
[out] Bit offset in the diagnostic image (ecatGetDiagnosisImagePtr())

4.6.16 ecatGetSlaveOutpVarInfo

Gets the output process variable information entries of a specific slave.

```
EC_T_DWORD ecatGetSlaveOutpVarInfo(
    EC_T_BOOL               bFixedAddressing,
    EC_T_WORD               wSlaveAddress,
    EC_T_WORD               wNumOfVarsToRead,
    EC_T_PROCESS_VAR_INFO*  pSlaveProcVarInfoEntries,
    EC_T_WORD*              pwReadEntries) ;
```

Parameters

bFixedAddressing
[in] EC_TRUE: Use station address, EC_FALSE: use AutoInc address
wSlaveAddress
[in] Slave address
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

Return

EC_E_NOERROR or error code

Comment

See **EC_T_PROCESS_VAR_INFO**.

4.6.17 ecatGetSlaveOutpVarInfoEx

Gets the output process variable extended information entries of a specific slave.

```
EC_T_DWORD ecatGetSlaveOutpVarInfoEx(
  EC_T_BOOL                  bFixedAddressing,
  EC_T_WORD                  wSlaveAddress,
  EC_T_WORD                  wNumOfVarsToRead,
  EC_T_PROCESS_VAR_INFO_EX*  pSlaveProcVarInfoEntriesEx,
  EC_T_WORD*                 pwReadEntries) ;
```

Parameters

bFixedAddressing
[in] EC_TRUE: Use station address, EC_FALSE: use AutoInc address
wSlaveAddress
[in] Slave address
wNumOfVarsToRead
[in] Number of found process variable entries
pSlaveProcVarInfoEntriesEx
[out] The read process variable extended information entries
pwReadEntries
[out] The number of read process variable information entries

Return

EC_E_NOERROR or error code

Comment

See **EC_T_PROCESS_VAR_INFO_EX**.

4.6.18 ecatFindInpVarByName

Finds an input process variable information entry by the variable name.

```
EC_T_DWORD ecatFindInpVarByName(
   EC_T_CHAR*                  szVariableName,
   EC_T_PROCESS_VAR_INFO*      pProcessVarInfoEntry);
```

Parameters

szVariableName
[in] Variable name
pProcessVarInfoEntry
[out] Process variable information entry

Return

EC_E_NOERROR or error code

Comment

See **EC_T_PROCESS_VAR_INFO**.

4.6.19 ecatFindInpVarByNameEx

Finds an input process variable extended information entry by the variable name.

```
EC_T_DWORD ecatFindInpVarByName(
   EC_T_CHAR*                  szVariableName,
   EC_T_PROCESS_VAR_INFO_EX*   pProcessVarInfoEntryEx);
```

Parameters

szVariableName
[in] Variable name
pProcessVarInfoEntryEx
[out] Process variable extended information entry

Return

EC_E_NOERROR or error code

Comment

See **EC_T_PROCESS_VAR_INFO_EX**.

4.6.20 ecatFindOutpVarByName

Finds an output process variable information entry by the variable name.

```
EC_T_DWORD ecatFindOutpVarByName(
   EC_T_CHAR*                 szVariableName,
   EC_T_PROCESS_VAR_INFO*  pProcessVarInfoEntry);
```

Parameters

szVariableName
[in] Variable name
pProcessVarInfoEntry
[out] Process variable information entry

Return

EC_E_NOERROR or error code

Comment

See **EC_T_PROCESS_VAR_INFO**.

4.6.21 ecatFindOutpVarByNameEx

Finds an output process variable extended information entry by the variable name.

```
EC_T_DWORD ecatFindOutpVarByNameEx(
   EC_T_CHAR*                 szVariableName,
   EC_T_PROCESS_VAR_INFO_EX*  pProcessVarInfoEntryEx);
```

Parameters

szVariableName
[in] Variable name
pProcessVarInfoEntryEx
[out] Process variable extended information entry

Return

EC_E_NOERROR or error code

Comment

See **EC_T_PROCESS_VAR_INFO_EX**.

4.6.22 ecatNotify – EC_NOTIFY_SLAVE_STATECHANGED

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

Parameters

pbyInBuf
[in] Pointer to EC_T_SLAVE_STATECHANGED_NTFY_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Is set to EC_NULL.
dwOutBufSize
[] Is set to 0.
pdwNumOutData
[] Is set to EC_NULL.

Comment

```
typedef struct _EC_T_SLAVE_STATECHANGED_NTFY_DESC
{
    EC_T_SLAVE_PROP SlaveProp;      /*< slave properties. See EC_NOTIFY_CYCCMD_WKC_ERROR */
    EC_T_STATE      newState;       /*< new state */
} EC_T_SLAVE_STATECHANGED_NTFY_DESC;
```

This notification is disabled by default. See EC_IOCTL_SET_NOTIFICATION_ENABLED for how to control the activation.

4.6.23 ecatNotify – EC_NOTIFY_SLAVES_STATECHANGED

Collects EC_NOTIFY_SLAVE_STATECHANGED.

Parameters

pbyInBuf
[in] Pointer to EC_T_SLAVES_STATECHANGED_NTFY_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Is set to EC_NULL.
dwOutBufSize
[] Is set to 0.
pdwNumOutData
[] Is set to EC_NULL.

Comment

```
typedef struct _EC_T_SLAVES_STATECHANGED_NTFY_DESC_ENTRY
{
   EC_T_WORD   wStationAddress;
   EC_T_STATE  eState;
} EC_T_SLAVES_STATECHANGED_NTFY_DESC_ENTRY;
typedef struct _EC_T_SLAVES_STATECHANGED_NTFY_DESC
{
   EC_T_WORD   wCount;
   EC_T_WORD   wRes;
   EC_T_SLAVES_STATECHANGED_NTFY_DESC_ENTRY
SlaveStates[MAX_SLAVES_STATECHANGED_NTFY_ENTRIES];
} EC_T_SLAVES_STATECHANGED_NTFY_DESC;
```

Description

wCount
[in] Contained EC_NOTIFY_SLAVE_STATECHANGED count.
wRes
[in] 0 (reserved)
SlavePresence
[in] Array of EC_T_SLAVE_PRESENCE_NTFY_DESC.
SlavePresence[…].wStationAddress
[in] Entry's station address.
SlavePresence[…].eState
[in] Entry's new state.

This notification is disabled by default. See EC_IOCTL_SET_NOTIFICATION_ENABLED for how to control the activation.

4.6.24 ecatWriteSlaveRegister

Writes data into the ESC memory of a specified slave.

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

```
EC_T_DWORD ecatWriteSlaveRegister (
    EC_T_BOOL       bFixedAddressing
    EC_T_WORD       wSlaveAddress
    EC_T_WORD       wRegisterOffset
    EC_T_VOID*      pvData,
    EC_T_WORD       wLen,
    EC_T_DWORD      dwTimeout,
);
```

Parameters

bFixedAddressing
[in] EC_TRUE: used station address, EC_FALSE: use Auto Increment address.
wSlaveAddress
[in] Address of slave (Station or Auto Increment depending on *bFixedAddressing*)
wRegisterOffset
[in] Register offset. I.e. use 0x0120 to write to the AL Control register.
pvData
[in] Pointer to the data to send.
wLen
[in] Number of bytes to send.
dwTimeout
[in] Timeout in msec.

Return

EC_E_NOERROR if successful.
EC_E_SLAVE_NOT_PRESENT if slave not present.
EC_E_BUSY another transfer request is already pending.
EC_E_NOTFOUND if the slave with the given address does not exist.
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_TIMEOUT if the slave did not respond to the command.
EC_E_BUSY if the master or the corresponding slave is currently changing its operational state .
EC_E_INVALIDPARM if the command is not supported or the timeout value is set to EC_NOWAIT.
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.

Comment

This function may not be called from within the JobTask's context.

4.6.25 ecatReadSlaveRegister

Reads data from the ESC memory of a specified slave.

```
EC_T_DWORD ecatReadSlaveRegister (
    EC_T_BOOL       bFixedAddressing,
    EC_T_WORD       wSlaveAddress,
    EC_T_WORD       wRegisterOffset,
    EC_T_VOID*      pvData,
    EC_T_WORD       wLen,
    EC_T_DWORD      dwTimeout,
);
```

Parameters

bFixedAddressing
[in] EC_TRUE: used station address, EC_FALSE: use Auto Increment address.
wSlaveAddress
[in] Address of slave (Station or Auto Increment depending on *bFixedAddressing*)
wRegisterOffset
[in] Register offset. I.e. use 0x0130 to read the AL Status register.
pvData
[out] Pointer to the receive buffer.
wLen
[in] Number of bytes to receive.
dwTimeout
[in] Timeout in msec.

Return

EC_E_NOERROR if successful.
EC_E_SLAVE_NOT_PRESENT if slave not present.
EC_E_BUSY another transfer request is already pending.
EC_E_NOTFOUND if the slave with the given address does not exist.
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_TIMEOUT if the slave did not respond to the command.
EC_E_BUSY if the master or the corresponding slave is currently changing its operational state .
EC_E_INVALIDPARM if the command is not supported or the timeout value is set to EC_NOWAIT.
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.

Comment

This function may not be called from within the JobTask's context.

4.6.26 ecatReadSlaveEEPRom

Read EEPROM data from slave.

```
EC_T_DWORD ecatReadSlaveEEPRom (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wEEPRomStartOffset,
    EC_T_WORD* pwReadData,
    EC_T_DWORD dwReadLen,
    EC_T_DWORD* pdwNumOutData,
    EC_T_DWORD dwTimeout
);
```

Parameters

bFixedAddressing
[in] EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing

wSlaveAddress
[in] Slave Address, station or auto increment address depending on *bFixedAddressing*

wEEPRomStartOffset
[in] Word address to start EEPROM read from.

pwReadData
[out] Pointer to EC_T_WORD array to carry the read data.

dwReadLen
[in] Size of the EC_T_WORD array provided at *pwReadData* (in EC_T_WORDs)

pdwNumOutData
[out] Pointer to EC_T_DWORD carrying actually read data (in EC_T_WORDs) after completion

dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

Comment

This function may not be called from within the JobTask's context.

4.6.27 ecatWriteSlaveEEPRom

Write EEPROM data to slave.

```
EC_T_DWORD ecatWriteSlaveEEPRom (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wEEPRomStartOffset,
    EC_T_WORD* pwWriteData,
    EC_T_DWORD dwWriteLen,
    EC_T_DWORD dwTimeout
);
```

Parameters

bFixedAddressing
[in] EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing

wSlaveAddress
[in] Slave Address, station or auto increment address depending on *bFixedAddressing*

wEEPRomStartOffset
[in] Word address to start EEPROM Write from.

pwWriteData
[in] Pointer to WORD array carrying the write data.

dwWriteLen
[in] Sizeof Write Data WORD array (in WORDS)

dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

Comment

The EEPROM's CRC is updated automatically.
ecatResetSlaveController() is needed to reload the alias address in register 0x12.
This function may not be called from within the JobTask's context.

4.6.28 ecatAssignSlaveEEPRom

Set EEPROM Assignment to PDI or EtherCAT Master.

```
EC_T_DWORD ecatAssignSlaveEEPRom (
    EC_T_BOOL       bFixedAddressing,
    EC_T_WORD       wSlaveAddress,
    EC_T_BOOL       bSlavePDIAccessEnable,
    EC_T_BOOL       bForceAssign,
    EC_T_DWORD      dwTimeout
);
```

Parameters

bFixedAddressing
[in] EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing

wSlaveAddress
[in] Slave Address, station or auto increment address depending on *bFixedAddressing*

bSlavePDIAccessEnable
[in] EC_TRUE: EEPROM assigned to slave PDI application, EC_FALSE: EEPROM assigned to Ecat Master

bForceAssign
[in] Force Assignment of EEPROM (only for ECat Master Assignment)

dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

Comment

This function may not be called from within the JobTask's context.

4.6.29 ecatActiveSlaveEEPRom

Check whether EEPROM is marked access active by Slave PDI application.

```
EC_T_DWORD ecatActiveSlaveEEPRom(
    EC_T_BOOL       bFixedAddressing,
    EC_T_WORD       wSlaveAddress,
    EC_T_BOOL*      pbSlavePDIAccessActive,
    EC_T_DWORD      dwTimeout
);
```

Parameters

bFixedAddressing
[in] EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing

wSlaveAddress
[in] Slave Address, station or auto increment address depending on *bFixedAddressing*

pbSlavePDIAccessActive
[out] Pointer to Boolean value: EC_TRUE: EEPROM active by PDI application, EC_FALSE: EEPROM not active

dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

Comment

This function may not be called from within the JobTask's context.

4.6.30 ecatReloadSlaveEEPRom

Causes a slave to reload its EEPROM values to ESC registers.
Attention: Alias address at 0x12 is not reloaded through this command. This is prevented by the slave hardware. To reload the alias address issue ecatResetSlaveController () .

```
EC_T_DWORD ecatReloadSlaveEEPRom(
    EC_T_BOOL       bFixedAddressing,
    EC_T_WORD       wSlaveAddress,
    EC_T_DWORD      dwTimeout
);
```

Parameters

bFixedAddressing
[in] EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing
wSlaveAddress
[in] Slave Address, station or auto increment address depending on *bFixedAddressing*
dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

Comment

This function may not be called from within the JobTask's context.

4.6.31 ecatResetSlaveController

Resets a ESC (e.g., ET1100, ET1200, and IP Core) if it is capable of issuing a hardware reset. A special sequence of three independent and consecutive frames/commands has to be sent do the slave (Reset register ECAT 0x0040 or PDI 0x0041). Afterwards, the slave is reset.

```
EC_T_DWORD  ecatResetSlaveController(
    EC_T_BOOL       bFixedAddressing,
    EC_T_WORD       wSlaveAddress,
    EC_T_DWORD      dwTimeout
);
```

Parameters

bFixedAddressing
[in] EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing
wSlaveAddress
[in] Slave Address, station or auto increment address depending on *bFixedAddressing*
dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

Comment

Please make sure that your slave ESC supports the reset. The EC-Master should be in INIT when issuing this API call. Set EC_T_INITMASTERPARMS.dwMaxSentQueuedFramesPerCycle >= 3.
This function may not be called from within the JobTask's context.

4.6.32 ecatIoControl – EC_IOCTL_ALL_SLAVES_MUST_REACH_MASTER_STATE

Specifies if all the slaves must reach the requested master state.

Parameters

pbyInBuf
[in] Pointer to EC_T_BOOL variable. If set to EC_TRUE all slaves must reach the master requested state, if set to EC_FALSE the master can reach the requested state even if some slaves are missing or cannot reach the requested state.

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

pbyOutBuf
[] Should be set to EC_NULL.

dwOutBufSize
[] Should be set to 0.

pdwNumOutData
[] Should be set to EC_NULL.

Comment

Missing mandatory slaves will be signalized by **EC_NOTIFY_SLAVE_PRESENCE**. Slaves who cannot reach the requested master state will be signalized by **EC_NOTIFY_SLAVE_UNEXPECTED_STATE**. **EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL** will not be generated anymore if this ioctl is called with EC_FALSE, **EC_NOTIFY_CYCCMD_WKC_ERROR** will be still generated.

4.6.33 ecatGetCfgSlaveInfo

Return information about a configured slave from the ENI file

```
EC_T_DWORD ecatGetCfgSlaveInfo(
    EC_T_BOOL               bStationAddress,
    EC_T_WORD               wSlaveAddress,
    EC_T_CFG_SLAVE_INFO*    pSlaveInfo
);
```

Parameters

bStationAddress
[in] EC_TRUE: used station address, EC_FALSE: use auto increment address

wSlaveAddress
[in] Address of slave (Station or auto increment depending on bStationAddress)

pSlaveInfo
[out] Information from the slave.

Return

EC_E_NOERROR if successful.
EC_E_NOTFOUND if the slave with the given address does not exist.

Comment

Size of EC_T_CFG_SLAVE_INFO content is subject to be extended.

EC_CFG_SLAVE_PD_SECTIONS = 4

```
typedef struct _EC_T_CFG_SLAVE_INFO{
    EC_T_DWORD      dwSlaveId;
    EC_T_CHAR       abyDeviceName[ECAT_DEVICE_NAMESIZE];
    EC_T_DWORD      dwHCGroupIdx;
    EC_T_BOOL       bIsPresent;
    EC_T_BOOL       bIsHCGroupPresent;
    EC_T_DWORD      dwVendorId;
    EC_T_DWORD      dwProductCode;
    EC_T_DWORD      dwRevisionNumber;
    EC_T_DWORD      dwSerialNumber;
    EC_T_WORD       wStationAddress;
    EC_T_WORD       wAutoIncAddress;
    EC_T_DWORD      dwPdOffsIn;
    EC_T_DWORD      dwPdSizeIn;
    EC_T_DWORD      dwPdOffsOut;
    EC_T_DWORD      dwPdSizeOut;
    EC_T_DWORD      dwPdOffsIn2;
    EC_T_DWORD      dwPdSizeIn2;
    EC_T_DWORD      dwPdOffsOut2;
    EC_T_DWORD      dwPdSizeOut2;
    EC_T_DWORD      dwPdOffsIn3;
    EC_T_DWORD      dwPdSizeIn3;
    EC_T_DWORD      dwPdOffsOut3;
    EC_T_DWORD      dwPdSizeOut3;
    EC_T_DWORD      dwPdOffsIn4;
    EC_T_DWORD      dwPdSizeIn4;
    EC_T_DWORD      dwPdOffsOut4;
    EC_T_DWORD      dwPdSizeOut4;
    EC_T_DWORD      dwMbxSupportedProtocols;
    EC_T_DWORD      dwMbxOutSize;
    EC_T_DWORD      dwMbxInSize;
    EC_T_DWORD      dwMbxOutSize2;
    EC_T_DWORD      dwMbxInSize2;
    EC_T_BOOL       bDcSupport;
    EC_T_WORD       wNumProcessVarsInp;
    EC_T_WORD       wNumProcessVarsOutp;
    EC_T_WORD       wPrevStationAddress;
    EC_T_WORD       wPrevPort;
    EC_T_WORD       wIdentifyAdo;
    EC_T_WORD       wIdentifyData;
    EC_T_BYTE       byPortDescriptor;
    EC_T_BYTE       abyReserved[3];
    EC_T_WORD       wWkcStateDiagOffsIn[EC_CFG_SLAVE_PD_SECTIONS];
    EC_T_WORD       wWkcStateDiagOffsOut[EC_CFG_SLAVE_PD_SECTIONS];
    EC_T_DWORD      adwReserved[3];
} EC_T_CFG_SLAVE_INFO;
```

Description

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

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

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

bIsPresent
: [out] Slave is currently present on bus

bIsHCGroupPresent
: [out] Slave's hot connect group is currently present on bus

dwVendorId
: [out] Vendor identification (from ENI file)

dwProductCode
: [out] Product code (from ENI file)

dwRevisionNumber
[out] Revision number (from ENI file)
dwSerialNumber
[out] Serial number (from ENI file)
wStationAddress
[out] The slave's station address (from ENI file)
wAutoIncAddress
[out] The slave's auto increment address (from ENI file)
dwPdOffsIn
[out] Process input data offset (in Bits) (from ENI file)
dwPdSizeIn
[out] Process input data data (in Bits) (from ENI file)
dwPdOffsOut
[out] Process output data offset (in Bits) (from ENI file)
dwPdSizeOut
[out] Process output data size (in Bits) (from ENI file)
dwPdOffsIn2
[out] 2nd sync unit process input data offset (in Bits) (from ENI file)
dwPdSizeIn2
[out] 2nd sync unit process input data size (in Bits) (from ENI file)
dwPdOffsOut2
[out] 2nd sync unit process output data offset (in Bits) (from ENI file)
dwPdSizeOut2
[out] 2nd sync unit process output data size (in Bits) (from ENI file)
dwPdOffsIn3
[out] 3rd sync unit process input data offset (in Bits) (from ENI file)
dwPdSizeIn3
[out] 3rd sync unit process input data size (in Bits) (from ENI file)
dwPdOffsOut3
[out] 3rd sync unit process output data offset (in Bits) (from ENI file)
dwPdSizeOut3
[out] 3rd sync unit process output data size (in Bits) (from ENI file)
dwPdOffsIn4
[out] 4th sync unit process input data offset (in Bits) (from ENI file)
dwPdSizeIn4
[out] 4th sync unit process input data size (in Bits) (from ENI file)
dwPdOffsOut4
[out] 4th sync unit process output data offset (in Bits) (from ENI file)
dwPdSizeOut4
[out] 4th sync unit process output data size (in Bits) (from ENI file)
dwMbxSupportedProtocols
[out] Mailbox protocol supported by the slave (from ENI file)
dwMbxOutSize
[out] Mailbox output size (in Bytes) (from ENI file)
dwMbxInSize
[out] Mailbox input size (in Bytes) (from ENI file)
dwMbxOutSize2
[out] Bootstrap mailbox output size (in Bytes) (from ENI file)
dwMbxInSize2
[out] Bootstrap mailbox input size (in Bytes) (from ENI file)
bDcSupport
[out] Slave supports DC (from ENI file)
wNumProcessVarsInp
[out] Number of input process data variables (from ENI file)
wNumProcessVarsOutp
[out] Number of output process data variables (from ENI file)
wPrevStationAddress
[out] Station address of the previous slave (from ENI file)
wPrevPort
[out] Connected port of the previous slave (from ENI file)
wIdentifyAdo

[out] ADO used for identification command (from ENI file)
wIdentifyData
[out] Identification value to be validated (from ENI file)
byPortDescriptor
[out] Port descriptor (ESC register 0x0007) (from ENI file)
wWkcStateDiagOffsIn
[out] Offset of WkcState bit in diagnosis image (ENI: ProcessData/Recv[1..4]/BitStart)
wWkcStateDiagOffsOut
[out] Offset of WkcState bit in diagnosis image (ENI: ProcessData/Send[1..4]/BitStart)

4.6.34 ecatGetBusSlaveInfo

Return information about a slave connected to the EtherCAT bus

```
EC_T_DWORD ecatGetBusSlaveInfo(
    EC_T_BOOL               bStationAddress,
    EC_T_WORD               wSlaveAddress,
    EC_T_BUS_SLAVE_INFO*    pSlaveInfo
);
```

Parameters

bStationAddress

[in] EC_TRUE: used station address, EC_FALSE: use auto increment address

wSlaveAddress

[in] Address of slave (Station or auto increment depending on bStationAddress)

pSlaveInfo

[out] Information from the slave.

Return

EC_E_NOERROR if successful.
EC_E_NOTFOUND if the slave with the given address does not exist.

Comment

Size of EC_T_BUS_SLAVE_INFO content is subject to be extended.

```
typedef struct _EC_T_BUS_SLAVE_INFO{
    EC_T_DWORD      dwSlaveId;
    EC_T_DWORD      adwPortSlaveIds[4];
    EC_T_WORD       wPortState;
    EC_T_WORD       wAutoIncAddress;
    EC_T_BOOL       bDcSupport;
    EC_T_BOOL       bDc64Support;
    EC_T_DWORD      dwVendorId;
    EC_T_DWORD      dwProductCode;
    EC_T_DWORD      dwRevisionNumber;
    EC_T_DWORD      dwSerialNumber;
    EC_T_BYTE       byESCType;
    EC_T_BYTE       byESCRevision;
    EC_T_WORD       wESCBuild;
    EC_T_BYTE       byPortDescriptor;
    EC_T_BYTE       byReserved;
    EC_T_WORD       wFeaturesSupported;
    EC_T_WORD       wStationAddress;
    EC_T_WORD       wAliasAddress;
    EC_T_WORD       wAlStatus;
    EC_T_WORD       wAlStatusCode;
    EC_T_DWORD      dwSystemTimeDifference;
    EC_T_WORD       wMbxSupportedProtocols;
    EC_T_WORD       wDlStatus;
    EC_T_WORD       wPrevPort;
    EC_T_WORD       wIdentifyData;
    EC_T_BOOL       bLineCrossed;
    EC_T_DWORD      dwSlaveDelay;
    EC_T_DWORD      dwPropagDelay;
    EC_T_DWORD      adwReserved[5];
} EC_T_BUS_SLAVE_INFO;
```

Description

dwSlaveId
[out] The slave's ID to bind bus slave and config slave information
adwPortSlaveIds
[out] The slave's ID of the slaves connected to ports, including MASTER_SLAVE_ID, MASTER_RED_SLAVE_ID or EL9010_SLAVE_ID
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)
(Bus Topology Scan)
wAutoIncAddress
[out] The slave's auto increment address
bDcSupport
[out] Slave supports DC.
(Bus Topology Scan)
bDc64Support
[out] Slave supports 64Bit DC.
(Bus Topology Scan)
dwVendorId
[out] Vendor Identification stored in the E²PROM at offset 0x0008
dwProductCode
[out] Product Code stored in the E²PROM at offset 0x000A
dwRevisionNumber
[out] Revision number stored in the E²PROM at offset 0x000C
dwSerialNumber
[out] Serial number stored in the E²PROM at offset 0x000E
byESCType
[out] Type of ESC (Value of slave ESC register 0x0000)
byESCRevision
[out] Revision number of ESC (Value of slave ESC register 0x0001)
wESCBuild
[out] Build number of ESC (Value of slave ESC register 0x0002)
byPortDescriptor
[out] Port descriptor (Value of slave ESC register 0x0007)
wFeaturesSupported
[out] Features supported (Value of slave ESC register 0x0008)
wStationAddress
[out] The slave's station address (Value of slave ESC register 0x0010)
wAliasAddress
[out] The slave's alias address (Value of slave ESC register 0x0012)
wAlStatus
[out] AL status (Value of slave ESC register 0x0130)
wAlStatusCode
[out] AL status code. (Value of slave ESC register 0x0134 during last error acknowledge). This value is reset after a slave state change
dwSystemTimeDifference
[out] System time difference. (Value of slave ESC register 0x092C)
wMbxSupportedProtocols
[out] Supported Mailbox Protocols stored in the E²PROM at offset 0x001C
wDlStatus
[out] DL status (Value of slave ESC register 0x0110)
wPrevPort
[out] Connected port of the previous slave
wIdentifyData
[out] Last read identification value see EC_T_CFG_SLAVE_INFO.wIdentifyAdo

bLineCrossed
[out] Line crossed was detetected at this slave
dwSlaveDelay
[out] Delay behind slave in ns. This value is only valid if a DC configuration is used
dwPropagDelay
[out] Propagation delay in ns. This value is only valid if a DC configuration is used

wMbxSupportedProtocols is 0 after calling `ecatConfigureMaster`, use `EC_T_CFG_SLAVE_INFO.dwMbxSupportedProtocols` from ecatGetCfgSlaveInfo instead.

4.7 Diagnosis, error detection, error notifications

4.7.1 Introduction

In case of errors on the bus or in one or multiple slaves the EtherCAT master stack will notify the application about such an event. The master automatically detects unexpected slaves states by evaluating the AL Status event interrupt. If the interrupt is set, the master reads the state of each slave and compares it to the expected (required) state. In case of a state mismatch the master generates the notification EC_NOTIFY_SLAVE_UNEXPECTED_STATE. The application will then have to enter an error handling procedure.
The error notifications can be separated into two classes:

a) Slave unrelated errors
b) Slave related errors

A slave related error notification will also contain the information about which slave has generated an error. If for example a slave could not be set into the requested state the application will get the EC_NOTIFY_SLAVE_INITCMD_RESPONSE_ERROR error notification including slave related information. A slave unrelated error does not contain this information even if one specific slave caused the 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 EC_NOTIFY_CYCCMD_WKC_ERROR error notification will be generated.

4.7.2 Example Error Scenario: Slave is powered off or disconnected while bus is operational

If the master 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 master will generate the EC_NOTIFY_CYCCMD_WKC_ERROR to indicate such an event. Also the master detects a DL status event and performs a bus scan as reaction on this. For the not reachable slaves (powered off or disconnected) the master generates the notification 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:

a) Master calls ecatNotify(EC_NOTIFY_CYCCMD_WKC_ERROR)
   → application gets informed

b) Use case: Slave is disconnected or powered off:
   Master detects a DL status event interrupt an performs a bus scan.
   Master calls ecatNotify(EC_NOTIFY_SLAVE_PRESENCE)
   → application gets informed and could set the whole master into a lower state, e. g. eEcatState_INIT

c) Use case: Slave state is not OPERATIONAL anymore
   Master calls ecatNotify(EC_NOTIFY_SLAVE_UNEXPECTED_STATE)
   → application gets informed and could either set the whole master into lower state (e. g. eEcatState_PREOP), or calls ecatSetSlaveState(…,DEVICE_STATE_OP) to repair the failed slave.

d) Use case: Slave is re-connected or powered on:
   Master detects a DL status event interrupt an performs a bus scan.
   Master calls ecatNotify(EC_NOTIFY_SLAVE_PRESENCE)

   Application could wait until all slaves are re-connected by calling the functions ecatGetNumConnectedSlaves() and ecatGetNumConfiguredSlaves().

   After all slaves are re-connected the application could either set the whole master to eEcatState_INIT and afterwards to eEcatState_OP, or the application uses ecatSetSlaveState() to repair only the failed slaves.

4.7.3 ecatEthDbgMsg

Send a debug message to the EtherCAT Link Layer. This feature can be used for debugging purposes. This function replaced the “ecatIoControl – EC_IOCTL_LINKLAYER_DBG_MSG”.

```
EC_T_DWORD ecatEthDbgMsg(
    EC_T_BYTE       byEthTypeByte0,
    EC_T_BYTE       byEthTypeByte1,
    EC_T_CHAR*      szMsg
);
```

Parameters

byEthTypeByte0
[in] Ethernet type byte 0.

byEthTypeByte1
[in] Ethernet type byte 1.

szMsg
[in] Message to send to Link Layer.

Return

EC_E_NOERROR if successful.

4.7.4 ecatIoControl – EC_IOCTL_SET_CYC_ERROR_NOTIFY_MASK

Sets a bit mask to enable or disable the generation of specific error notifications for cyclic commands.
An error will be signalled to the application as long as it exists. If for example one slave is intended to work not in operational state an error would be generated for every IO cycle. The application then can decide to suppress those error messages.
By default all errors are enabled (the notification mask is set to EC_CYC_ERR_MASK_ALL).

Parameters

pbyInBuf
[in] pointer to a EC_T_DWORD type value containing the new error mask.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

The following cyclic error notification mask values exist:

- **EC_CYC_ERR_MASK_NOT_ALL_DEVICES_OPERATIONAL**
  → mask for cyclic EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL notifications
- **EC_CYC_ERR_MASK_STATUS_SLAVE_ERROR**
  → mask for cyclic EC_NOTIFY_STATUS_SLAVE_ERROR notifications
- **EC_CYC_ERR_MASK_CYCCMD_WKC_ERROR**
  → mask for cyclic EC_NOTIFY_CYCCMD_WKC_ERROR notifications
- **EC_CYC_ERR_MASK_UNEXPECTED_FRAME_RESPONSE**
  → mask for cyclic EC_NOTIFY_FRAME_RESPONSE_ERROR notifications with error type eRspErr_UNEXPECTED
- **EC_CYC_ERR_MASK_NO_FRAME_RESPONSE_ERROR**
  → mask for cyclic EC_NOTIFY_FRAME_RESPONSE_ERROR notifications with error type eRspErr_NO_RESPONSE
- **EC_ERR_MASK_ETH_LINK_NOT_CONNECTED**
  → mask for EC_NOTIFY_ETH_LINK_NOT_CONNECTED notifications
- **EC_CYC_ERR_MASK_ALL**
  → mask for all cyclic notifications

Comment

–

4.7.5 ecatIoControl – EC_IOCTL_GET_SLVSTATISTICS

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

Parameters

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.

Comment

```
EC_T_SLVSTATISTICS_DESC
typedef struct _EC_T_SLVSTATISTICS_DESC
{
    EC_T_WORD           wRxErrorCounter[4];
    EC_T_BYTE           byFwdRxErrorCounter[4];
    EC_T_BYTE           byEcatProcUnitErrorCounter;
    EC_T_BYTE           byPDIErrorCounter;
    EC_T_WORD           wAlStatusCode;
    EC_T_BYTE           byLostLinkCounter[4];
} EC_T_SLVSTATISTICS_DESC;
```

Description

wRxErrorCounter[4]
[out] RX Error Counters per Slave Port
byFwdRxErrorCounter[4];
[out] Forwarded RX Error Counters per Slave Port
byEcatProcUnitErrorCounter
[out] EtherCAT Processing unit error counter
byPDIErrorCounter;
[out] PDI Error counter
wAlStatusCode;
[out] AL status code
byLostLinkCounter[4];
[out] Lost Link Counter per Slave Port

See also EC_IOCTL_SET_SLVSTAT_PERIOD and EC_IOCTL_CLR_SLVSTATISTICS.

4.7.6 ecatIoControl – EC_IOCTL_CLR_SLVSTATISTICS

Clear all error registers in all slaves

Parameters

pbyInBuf
[] Should be set to EC_NULL.
dwInBufSize
[] Should be set to 0.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

4.7.7 ecatIoControl – EC_IOCTL_SET_SLVSTAT_PERIOD

Update Slave Statistics collection period.
It implicitly forces an immediate collection of slave statistics if performed successful.

Parameters

pbyInBuf
[in] pointer to a EC_T_DWORD type variable containing the slave statistics collection period [ms] to set.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf

[] Should be set to EC_NULL.

dwOutBufSize

[] Should be set to 0.

pdwNumOutData

[] Should be set to EC_NULL.

Comment

A Period of 0 disables automatic slave statistics collection.

4.7.8 ecatIoControl – EC_IOCTL_FORCE_SLVSTAT_COLLECTION

Sends datagrams to collect slave statistics counters.

Parameters

pbyInBuf

[] Should be set to EC_NULL.

dwInBufSize

[] Should be set to 0.

pbyOutBuf

[] Should be set to EC_NULL.

dwOutBufSize

[] Should be set to 0.

pdwNumOutData

[] Should be set to EC_NULL.

4.7.9 ecatIoControl – EC_IOCTL_SET_FRAME_LOSS_SIMULATION

This IO Control is introduced for testing and debugging purposes. It enables an application to simulate the loss of EtherCAT frames on both transmit and receive bus direction.

Attention: Do not activate this on shipped releases. Frameloss has significant influence on performance and reliablility of your application!

Parameters

pbyInBuf

[in] Array of four EC_T_DWORDs (arrDword), see below.

dwInBufSize

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

pbyOutBuf

[] Should be set to EC_NULL.

dwOutBufSize

[] Should be set to 0.

pdwNumOutData

[] Should be set to EC_NULL.

Comment

The parameters configurable are :

arrDword [0] ➔ dwNumGoodFramesAfterStart

Number of good frames before frame loss simulation starts

arrDword [1] ➔ dwFrameLossLikelihoodPpm

Random loss simulation: frame loss likelihood (ppm)

arrDword [2] ➔ dwFixedLossNumGoodFrames

Fixed loss simulation: number of good frames before frame loss

arrDword [3] ➔ dwFixedLossNumLostFrames

Fixed loss simulation: number of lost frames after processing the good ones

4.7.10 ecatIoControl – EC_IOCTL_SET_RXFRAME_LOSS_SIMULATION

Same as ecatIoControl – EC_IOCTL_SET_FRAME_LOSS_SIMULATION but only enables receive direction frame losses.

4.7.11 ecatIoControl – EC_IOCTL_SET_TXFRAME_LOSS_SIMULATION

Same as ecatIoControl – EC_IOCTL_SET_FRAME_LOSS_SIMULATION but only enables transmit direction frame losses.

4.7.12 Error notifications – general information

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

```
typedef struct _EC_T_ERROR_NOTIFICATION_DESC{
    EC_T_DWORD          dwNotifyErrorCode;    /* error ID (same value as the notification code) */
    EC_T_CHAR           achErrorInfo[];       /* additional error string (may be empty) */
    union {
        EC_T_WKCERR_DESC    WkcErrDesc        /* detailed  error information, see below */
        EC_T_XXXXX          Xxxxx             /* detailed  error information, see below */
         :     :     :        :   :
         :     :     :        :   :
    }
} EC_T_ERROR_NOTIFICATION_DESC;
```

If the pointer to this descriptor exists (is not set to EC_NULL) 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.

The EtherCAT master will call ecatNotify every time an error is detected. In some cases this will lead to calling this function in every EtherCAT cycle (e.g. if there is no physical connection to the slaves).
Using the control interface ecatIoControl – EC_IOCTL_SET_CYC_ERROR_NOTIFY_MASK it is possible to determine which errors shall be signalled and which not.

4.7.13 EC_NOTIFY_CYCCMD_WKC_ERROR

To update the process data some EtherCAT commands will be sent cyclically by the 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 master. 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 master is determined by the EtherCAT configuration (XML) file for each cyclic EtherCAT command (section Config/Cyclic/Frame/Cmd/Cnt).
Detailed error information is stored in structure EC_T_WKCERR_DESC of the union element WkcErrDesc:

```
typedef struct _EC_T_WKCERR_DESC{
    EC_T_SLAVE_PROP SlaveProp;      /* slave properties, content not used (undefined) in case of cyclic
                                       WKC_ERROR  */
    EC_T_BYTE       byCmd;          /* EtherCAT command where this error occurred */
    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 */
} EC_T_WKCERR_DESC;

typedef struct _EC_T_SLAVE_PROP{
    EC_T_WORD       wStationAddress;/* station address */
    EC_T_WORD       wAutoIncAddr;   /* auto increment address */
    EC_T_CHAR       achName[];      /* name of the slave device (NULL terminated string) */
} EC_T_SLAVE_PROP;
```

Comment
See 3.3.8 “Cyclic cmd WKC validation”.

4.7.14 EC_NOTIFY_MASTER_INITCMD_WKC_ERROR

This error will be indicated in case of a working counter mismatch when sending master init commands. The working counter value expected by the master is determined by the EtherCAT configuration (XML) file for each master init command (section Config/Master/InitCmds/InitCmd/Cnt).
In case there is no „Cnt“ entry in the XML file for this init command there will be no working counter verification. The working counter has to be incremented by all slaves which have to process this init command.
Detailed error information is stored in structure EC_T_WKCERR_DESC of the union element WkcErrDesc.

4.7.15 EC_NOTIFY_SLAVE_INITCMD_WKC_ERROR

This error will be indicated in case of a working counter mismatch when sending slave init commands. The working counter value expected by the master is determined by the EtherCAT configuration (XML) file for each slave init command (section Config/Slave/InitCmds/InitCmd/Cnt).
In case there is no „Cnt“ entry in the XML file for this init command there will be no working counter verification.
Detailed error information is stored in structure EC_T_WKCERR_DESC of the union element WkcErrDesc. The structure member SlaveProp contains information about the corresponding slave device:

4.7.16 EC_NOTIFY_FOE_MBSLAVE_ERROR

This error will be indicated in case a slave notifies an error over FoE. See EC-Master Class A Manual.

4.7.17 EC_NOTIFY_EOE_MBXSND_WKC_ERROR

This error will be indicated in case the working counter of a EoE mailbox write command was not set to the expected value of 1.
Detailed error information is stored in structure EC_T_WKCERR_DESC of the union element WkcErrDesc. The structure member SlaveProp contains information about the corresponding slave device.

4.7.18 EC_NOTIFY_COE_MBXSND_WKC_ERROR

This error will be indicated in case the working counter of a CoE mailbox write command was not set to the expected value of 1.
Detailed error information is stored in structure EC_T_WKCERR_DESC of the union element WkcErrDesc. The structure member SlaveProp contains information about the corresponding slave device.

4.7.19 EC_NOTIFY_FOE_MBXSND_WKC_ERROR

This error will be indicated in case the working counter of a FoE mailbox write command was not set to the expected value of 1. See EC-Master Class A Manual.

4.7.20 EC_NOTIFY_VOE_MBXSND_WKC_ERROR

This error will be indicated in case the working counter of a VoE mailbox write command was not set to the expected value of 1.
Detailed error information is stored in structure EC_T_WKCERR_DESC of the union element WkcErrDesc. The structure member SlaveProp contains information about the corresponding slave device.

4.7.21 EC_NOTIFY_FRAME_RESPONSE_ERROR

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

Missing response (timeout, *eRspErr_NO_RESPONSE/eRspErr_FRAME_RETRY*) **acyclic frames**:
Acyclic Ethernet frames are internally queued by the master and sent to the slaves at a later time (usually after sending cyclic frames).
The master will monitor the time between queueing such a frame and receiving the result. If a maximum time is exceeded then this error will be indicated. This maximum time will be determined by the parameter dwEcatCmdTimeout when the master is initialized (see also section 4.3.1).
The master will retry sending the frame if the master configuration parameter dwEcatCmdMaxRetries is set to a value greater than 1. In case of a retry the *eRspErr_FRAME_RETRY* error is signalled, if the number of retries has elapsed the *eRspErr_NO_RESPONSE* error is signalled.
Possible reasons:

a) the frame was not received at all (due to bus problems)
   In this case the *achErrorInfo[]* member of the error notification descriptor *will contain the string "L".*
b) the frame was sent too late by the master due to a improper configuration.
   In this case the *achErrorInfo[]* member of the error notification descriptor *will contain the string "T".*
   To avoid this error the configuration may be changed as follows:
   → higher value for master configuration parameter dwMaxSentQueuedFramesPerCyc
   → shorter master timer cycle, i.e. shorter period between two calls to ecatExecJob(eUsrJob_MasterTimer)
   → higher timeout value (master configuration parameter dwEcatCmdTimeout)

If the frame was sent too late by the master (due to improper configuration values) it will also be received too late and the master then signals an *eRspErr_WRONG_IDX* or *eRspErr_UNEXPECTED* error (as the master then doesn't expect to receive this frame).

Missing response (timeout, *eRspErr_NO_RESPONSE*) **cyclic frames**:
A response to all cyclic frames must occur until the next cycle starts. If the first cyclic frame is sent the master checks whether all cyclic frames of the last cycle were received. If there is one frame missing this error is indicated.
Possible reasons:

a) the frame was not received (due to bus problems)
b) too many or too long acyclic frames are sent in between sending cyclic frames by the master due to a improper configuration, to avoid these error notifications the configuration may be changed as follows:
   → lower value for master configuration parameter dwMaxSentQueuedFramesPerCyc
   → higher cyclic timer period, i.e. less calls to ecatExecJob(*eUsrJob_SendAllCycFrames*)
c) non-deterministic sending of acyclic frames.
   Sending acyclic frames by calling ecatExecJob(eUsrJob_SendAcycFrames) have to be properly scheduled with sending cyclic frames by calling ecatExecJob(eUsrJob_SendAllCycFrames).

Detailed error information is stored in structure EC_T_FRAME_RSPERR_DESC of the union element FrameRspErrDesc:

```
typedef struct _EC_T_FRAME_RSPERR_DESC{
    EC_T_BOOL     bIsCyclicFrame;          /* EC_TRUE if the frame contains cyclic commands */
    EC_T_FRAME_RSPERR_TYPE
                                  EErrorType;  /* Error type, see below */
    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
                                          – eRspErr_UNEXPECTED */
} EC_T_FRAME_RSPERR_DESC;

typedef enum _EC_T_FRAME_RSPERR_TYPE{
    eRspErr_NO_RESPONSE                   /* No Ethernet frame received (timeout, frame loss) */
    eRspErr_WRONG_IDX                     /* Wrong IDX value in acyclic frame */
    eRspErr_UNEXPECTED                    /* Unexpected frame was received */
    eRspErr_FRAME_RETRY                   /* Ethernet frame will be re-sent (timeout, frame loss) */
    eRspErr_RETRY_FAIL                    /* all retry mechanism fails to re-sent acyclic frames */
} EC_T_FRAME_RSPERR_TYPE;
```

4.7.22 EC_NOTIFY_SLAVE_INITCMD_RESPONSE_ERROR

This error code will be indicated if a slave does not respond appropriately while sending slave init commands. The slave init commands are defined in the EtherCAT configuration (XML) file (Config/Slave/InitCmds/InitCmd). A timeout value for these commands may also be defined in the configuration file (Config/Slave/InitCmds/InitCmd/Timeout). If there is no timeout value defined here the frame response is expected within one single cycle.

Detailed error information is stored in structure EC_T_INITCMD_ERR_DESC of the union element InitCmdErrDesc:

```
typedef struct _EC_T_INITCMD_ERR_DESC{
    EC_T_SLAVE_PROP SlaveProp;            /* Slave properties */
    EC_T_INITCMD_ERR_TYPE
                                  EErrorType;  /* Error type, see below */
    EC_T_CHAR     achStateChangeName[]; /* State change description when the error occurred */
    EC_T_CHAR     szComment[];            /*< comment (ENI) */
} EC_T_INITCMD_ERR_DESC;

typedef enum _EC_T_INITCMD_ERR_TYPE{
    eInitCmdErr_NO_RESPONSE               /* No Ethernet frame received (timeout) */
    eInitCmdErr_VALIDATION_ERR            /* Validation error (invalid slave command response) */
    eInitCmdErr_FAILED                    /* Init commands failed (state could not be reached) */
}EC_T_INITCMD_ERR_TYPE;
```

4.7.23 EC_NOTIFY_MBSLAVE_INITCMD_TIMEOUT

This error is identical to error code EC_NOTIFY_SLAVE_INITCMD_RESPONSE_ERROR but it will be indicated in case of timeouts when processing mailbox init commands.
The timeout value used for CoE mailbox slaves is defined in the EtherCAT configuration (XML) file (Config/Slave/Mailbox/CoE/InitCmds/InitCmd/Timeout). In case this value is set to 0 a fixed timeout value of 500 msec will be used by the EtherCAT master. The timeout value used for EoE mailbox slaves will be set fixed to a value of 5000 msec.

4.7.24 EC_NOTIFY_MASTER_INITCMD_RESPONSE_ERROR

This error code will be indicated if a missing or wrong command response was detected while sending master init commands. The master init commands are defined in the EtherCAT configuration (XML) file (Config/Master/InitCmds/InitCmd). A timeout value for these commands may also be defined in the configuration file (Config/Master/InitCmds/InitCmd/Timeout). If there is no timeout value defined here the frame response is expected within one single cycle.
Detailed error information is stored in structure EC_T_INITCMD_ERR_DESC of the union element InitCmdErrDesc.

4.7.25 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.

4.7.26 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 EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL and all the slaves are back in OPERATIONAL state.

4.7.27 EC_NOTIFY_STATUS_SLAVE_ERROR

When processing cyclic frames the EtherCAT master checks if at least one slave has the ERROR bit in the AL-STATUS register set. In that case this error will be indicated. The master will then automatically determine detailed error information of the slave(s) indicating an error and acknowledge the error status. The application will get a EC_NOTIFY_SLAVE_ERROR_STATUS_INFO notification for each such slave. Usually those slaves will enter safe-operational state in this case. It is the application's response how to further handle such error cases.

4.7.28 EC_NOTIFY_SLAVE_ERROR_STATUS_INFO

Every time the master detects a slave error, the Error bit on the specific slave is cleared and this error code will be signalled to the application.
Detailed error information is stored in structure EC_T_SLAVE_ERROR_INFO_DESC of the union element SlaveErrInfoDesc.

```
typedef struct _ EC_T_SLAVE_ERROR_INFO_DESC{
    EC_T_SLAVE_PROP SlaveProp;          /* Slave properties, see above */
    EC_T_WORD    wStatus;               /* Slave status (AL STATUS) */
    EC_T_WORD    wStatusCode;           /* Slave status code (AL STATUS CODE) */
} EC_T_SLAVE_ERROR_INFO_DESC;
```

4.7.29 EC_NOTIFY_SLAVES_ERROR_STATUS

This notification collects notifications of type EC_NOTIFY_SLAVE_ERROR_STATUS_INFO.
Notification is given on either collection full or master state changed whatever comes first.
This notification is disabled by default. See EC_IOCTL_SET_NOTIFICATION_ENABLED for how to control the activation.

```
typedef struct _EC_T_SLAVES_ERROR_DESC
{
   EC_T_WORD wCount;
   EC_T_WORD wRes;
   EC_T_SLAVES_ERROR_DESC_ENTRY SlaveError[MAX_SLAVES_ERROR_NTFY_ENTRIES];
} EC_T_SLAVES_ERROR_DESC;
```

4.7.30 EC_NOTIFY_SLAVE_UNEXPECTED_STATE

This error is signalized every time a slave changes into an unexpected state.
Detailed error information is stored in structure EC_T_SLAVE_UNEXPECTED_STATE_DESC of the union element SlaveErrInfoDesc.

```
typedef struct _ EC_T_SLAVE_UNEXPECTED_STATE_DESC{
    EC_T_SLAVE_PROP SlaveProp;          /* Slave properties, see above */
    EC_T_STATE    curState;             /* Current state */
    EC_T_STATE    expState;             /* Expected state */
} EC_T_SLAVE_UNEXPECTED_STATE_DESC;
```

4.7.31 EC_NOTIFY_SLAVES_UNEXPECTED_STATE

This notification collects notifications of type EC_NOTIFY_SLAVE_UNEXPECTED_STATE.
Notification is given on either collection full or master state changed whatever comes first.
This notification is disabled by default. See EC_IOCTL_SET_NOTIFICATION_ENABLED for how to control the activation.

```
typedef struct _EC_T_SLAVES_UNEXPECTED_STATE_DESC_ENTRY
{
   EC_T_WORD  wStationAddress;
   EC_T_STATE curState;
   EC_T_STATE expState;
} EC_T_SLAVES_UNEXPECTED_STATE_DESC_ENTRY;
typedef struct _EC_T_SLAVES_UNEXPECTED_STATE_DESC
{
   EC_T_WORD wCount;
   EC_T_WORD wRes;
   EC_T_SLAVES_UNEXPECTED_STATE_DESC_ENTRY
SlaveStates[MAX_SLAVES_UNEXPECTED_STATE_NTFY_ENTRIES];
} EC_T_SLAVES_UNEXPECTED_STATE_DESC;
```

4.7.32 EC_NOTIFY_ETH_LINK_NOT_CONNECTED

This error code will be indicated if the Ethernet link is disconnected. This error is never indicated if the Link Layer does not support detection of a missing link cable.

4.7.33 EC_NOTIFY_SLAVE_NOT_ADDRESSABLE

If the master cannot get the slave device state using its station address (if the working counter is not incremented) this error will be generated.
This may happen if the slave was removed from the bus or powered off during normal operation. Detailed error information is stored in structure EC_T_WKCERR_DESC of the union element WkcErrDesc. The structure member SlaveProp contains information about the corresponding slave device.

4.7.34 EC_NOTIFY_CLIENTREGISTRATION_DROPPED

This notification will be indicated if the client registration was dropped because ecatConfigureMaster was called by another thread. The notification has the following parameter:

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

4.7.35 EC_NOTIFY_EEPROM_CHECKSUM_ERROR

This error is signalized every time a EEPROM checksum error is detected.
Detailed error information is stored in structure EC_T_EEPROM_CHECKSUM_ERROR_DESC of the union element SlaveErrInfoDesc.

```
typedef struct _ EC_T_EEPROM_CHECKSUM_ERROR_DESC{
    EC_T_SLAVE_PROP SlaveProp;          /* Slave properties */
} EC_T_ EEPROM_CHECKSUM_ERROR _DESC;
```

4.7.36 EC_NOTIFY_PDIWATCHDOG

This error is signalized every time a PDI watchdog error is detected.
Detailed error information is stored in structure EC_T_PDIWATCHDOG_DESC of the union element SlaveErrInfoDesc.

```
typedef struct _EC_T_PDIWATCHDOG_DESC {
    EC_T_SLAVE_PROP SlaveProp;          /* Slave properties */
} EC_T_PDIWATCHDOG_DESC;
```

4.7.37 ecatGetText

Return text tokens by ID from master stack.

EC_T_CHAR* ecatGetText(EC_T_WORD wTextId);

Parameters

wTextId

[in] Text enumeration ID

Return

Master stack stored text. To find the subordinate texts see source code in file EcError.h.

4.7.38 ecatPerfMeasInit

Initialize performance measurement.

```
EC_T_VOID ecatPerfMeasInit(
    EC_T_TSC_MEAS_DESC*    pTscMeasDesc,
    EC_T_UINT64            dwlFreqSet,
    EC_T_DWORD             dwNumMeas,
    EC_T_FNMESSAGE         pfnMessage
);
```

Parameters

pTscMeasDesc
[in,out] measurement descriptor
dwlFreqSet
[in] TSC frequency, 0: auto-calibrate
dwNumMeas
[in] number of elements to be allocated in in pTscMeasDesc->aTscTime
pfnMessage
[in] Reserved. Set to EC_NULL.

Return

-

4.7.39 ecatPerfMeasDeinit

De-initialize performance measurement.

```
EC_T_VOID ecatPerfMeasDeinit(
    EC_T_TSC_MEAS_DESC*    pTscMeasDesc
);
```

Parameters

pTscMeasDesc
[in,out] measurement descriptor

Return

-

4.7.40 ecatPerfMeasEnable

Enable performance measurement.

```
EC_T_VOID ecatPerfMeasEnable(
    EC_T_TSC_MEAS_DESC*    pTscMeasDesc
);
```

Parameters

pTscMeasDesc
[in,out] measurement descriptor

Return

-

4.7.41 ecatPerfMeasDisable

Disable performance measurement.

```
EC_T_VOID ecatPerfMeasDisable(
    EC_T_TSC_MEAS_DESC*    pTscMeasDesc
);
```

Parameters

pTscMeasDesc
[in,out] measurement descriptor

Return

-

4.7.42 ecatPerfMeasStart

Start measurement.

```
EC_T_VOID ecatPerfMeasStart(
    EC_T_TSC_MEAS_DESC*    pTscMeasDesc,
    EC_T_DWORD             dwIndex
);
```

Parameters

pTscMeasDesc
[in,out] measurement descriptor
dwIndex
[in] measurement index

Return

-

4.7.43 ecatPerfMeasEnd

Stop measurement.

```
EC_T_TSC_TIME* ecatPerfMeasEnd(
    EC_T_TSC_MEAS_DESC*    pTscMeasDesc,
    EC_T_DWORD             dwIndex
);

typedef struct _EC_T_TSC_TIME
{
   EC_T_UINT64     qwStart;          /* start time */
   EC_T_UINT64     qwEnd;            /* end time */
   EC_T_DWORD      dwCurr;           /* [1/10 usec] */
   EC_T_DWORD      dwMax;            /* [1/10 usec] */
   EC_T_DWORD      dwAvg;            /* [1/100 usec] */
   EC_T_BOOL       bMeasReset;       /* EC_TRUE if measurement values shall be reset */
   EC_T_INT        nIntLevel;        /* for interrupt lockout handling */
} EC_T_TSC_TIME;
```

Parameters

pTscMeasDesc
[in,out] measurement descriptor
dwIndex
[in] measurement index

Return

Pointer to corresponding time descriptor.

4.7.44 ecatPerfMeasReset

Request measurement reset. Reset is done within ecatPerfMeasEnd().

```
EC_T_VOID ecatPerfMeasReset(
    EC_T_TSC_MEAS_DESC*    pTscMeasDesc,
    EC_T_DWORD             dwIndex
);
```

Parameters

pTscMeasDesc
[in,out] measurement descriptor
dwIndex
[in] measurement index, 0xFFFFFFFF: all indexes

Return

-

4.7.45 ecatPerfMeasShow

Log current performance measurement values using OsDbgMsg.

```
EC_T_VOID ecatPerfMeasShow(
    EC_T_TSC_MEAS_DESC*    pTscMeasDesc,
    EC_T_DWORD             dwIndex,
    EC_T_CHAR**            aszMeasCaption
);
```

Parameters

pTscMeasDesc
[in] measurement descriptor
dwIndex
[in] measurement index, 0xFFFFFFFF: all indexes
aszMeasCaption
[in] captions as array of zero terminated strings.

Return

-

4.7.46 ecatLogFrameEnable

Setup a callback function to log the EtherCAT network traffic. The callback function is called by the cyclic task. Therefore the code inside the callback has to be fast and non-blocking. The callback parameter *dwLogFlags* can be used as a filter to log just specific frames.

```
EC_T_DWORD ecatLogFrameEnable (
    EC_T_PFLOGFRAME_CB    pvLogFrameCallBack,
    EC_T_VOID*            pvContext
);
```

Parameters

pvLogFrameCallBack
[in] Pointer to frame logging callback function
pvContext
[in] Pointer to function specific context

Return

EC_E_NOERROR if successful.

```
typedef EC_T_VOID (*EC_T_PFLOGFRAME_CB)(EC_T_VOID* pvContext, EC_T_DWORD dwLogFlags,
EC_T_DWORD dwFrameSize, EC_T_BYTE* pbyFrame);
```

Description

pvContext
[in] Context pointer. This pointer is used as parameter when the callback function is called.
dwLogFlags
[in] Flags (defined in AtEtherCAT.h) containing the master state and other information.
dwFrameSize
[in] Size of frame in bytes.
pbyFrame
[in] Pointer to frame data.

Example

```
/*****************************************************************************/
/** \brief  Handler to log frames.
*
*   CAUTION: Called by cyclic task!!! Do not consume to much CPU time!!!
* \return  Status value.
*/
EC_T_VOID LogFrameHandler(EC_T_VOID* pvContext, EC_T_DWORD dwLogFlags, EC_T_DWORD dwFrameSize,
EC_T_BYTE* pbyFrame)
{
    EC_T_STATE         eMasterState;

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

    /* is this a rx frame? */
    if((S_dwLogFrameLevel == 3) && !(dwLogFlags & EC_LOG_FRAME_FLAG_RX_FRAME))
    {
        /* don't log this frame */
        return;
    }

    /* is this a acyclic frame? */
    if((S_dwLogFrameLevel == 2) && !(dwLogFlags & EC_LOG_FRAME_FLAG_ACYC_FRAME))
    {
        /* don't log this frame */
        return;
    }

    /* is this a red frame? */
    if(dwLogFlags & EC_LOG_FRAME_FLAG_RED_FRAME)
    {

    }

    return;
}
```

4.7.47 ecatLogFrameDisable

Disable the frame logging callback.

EC_T_DWORD ecatLogFrameDisable (EC_T_VOID);

Return

EC_E_NOERROR if successful.

4.8 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.
Each mailbox transfer object is related to a specific mailbox transfer type, it cannot be used for other transfer types. For example it is not possible to use the same mailbox transfer object for both SDO upload and SDO download.
The one client that initiated the mailbox transfer will be notified about a mailbox transfer completion by the *ecatNotify* 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.) MbxTferDesc.dwDataLen = 10
2.) MbxTferDesc. pbyMbxTferDescData=(EC_T_PBYTE)OsMalloc(MbxTferDesc.dwDataLen)
3.) pMbxTfer = ecatMbxTferCreate(&MbxTferDesc)
→ state of the transfer object = Idle

4.) OsMemcpy(pMbxTfer->pbyMbxTferData, „0123456789“, 10)
5.) pMbxTfer->dwTferId = 1;
6.) pMbxTfer->dwClntId = dwClntId;
7.) pMbxTfer->dwDataLen=10;
8.) dwResult = ecatCoeSdoDownloadReq(pMbxTfer, dwSlaveId, wObIndex, …)
→ state of the transfer object = Pend or TferReqError
9.) if(dwResult != EC_E_NOERROR) { … }

10.) ecatNotify(EC_NOTIFY_MBOXRCV, pParms)
→ state of the transfer object = TferDone
11.) if(pMbxTfer->dwErrorCode != EC_E_NOERROR) { … }
→ In ecatNotify: application may set transfer object state to Idle

12.) ecatMbxTferDelete(pMbxTfer)

Steps 1 to 3: create a transfer object (for example a SDO download transfer object).

Steps 4 to 8: copy the data to be transferred to the slave into the transfer object, determine the transfer ID, store the client ID in the object and initiate the transfer (e.g. a SDO download). A transfer may only be initiated if the state of the transfer object is Idle.
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.

Steps 9 to 10: If the mailbox transfer is completed the notification callback function of the corresponding client (*ecatNotify*) will be called with a pointer to the mailbox transfer object. The state of the transfer object is set to TferDone prior to calling *ecatNotify*.
Step 11: In case of errors the appropriate error handling has to be executed. Application must set the transfer object state to Idle.

Step 12: Delete the transfer object. Alternatively this object can be used for the next transfer.

4.8.1 Mailbox transfer object states

The following states exist for a mailbox transfer object:

eMbxTferStatus_Idle
The object is not in use and can be used for a new mailbox transfer. The object is owned by the application.

eMbxTferStatus_Pend
The object is currently used by the EtherCAT master and the transfer is not completed. The client may not access the transfer object (except for reading the object's state).
The client's notification function (ecatNotify) will be called for progress information by the master in this state for some types of mailbox transfers. The object is owned by the EtherCAT master.

eMbxTferStatus_TferWaitingForContinue
The EtherCAT master awaits more data from the application or provided data or information that must be acknowledged by the application. The object is currently owned by application.

eMbxTferStatus_TferDone
A mailbox transfer was completed (with or without error). The client's notification function (ecatNotify) will be called by the master after setting the transfer object into this state. The object is owned by the application.

eMbxTferStatus_TferReqError
The master was not able to initiate a mailbox transfer. After some kind of error handling the client has to set the object state back to eMbxTferStatus_Idle. The object is owned by the application.

Comment

A mailbox transfer will be processed by the master 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, see 4.8.3 "ecatMbxTferAbort".

After completion of the mailbox transfer (with timeout and the client may finally set the transfer object into the state eMbxTferStatus_Idle.

New mailbox transfers can only be requested if the object is in the state eMbxTferStatus_Idle.

4.8.2 ecatMbxTferCreate

Creates a mailbox transfer object.

```
EC_T_MBXTFER* ecatMbxTferCreate (
    EC_T_MBXTFER_DESC* pMbxTferDesc
);
```

Parameters

pMbxTferDesc

[in] Pointer to the mailbox transfer descriptor. Determines details of the mailbox transfer.

Return

Pointer to the mailbox transfer object. This object is needed when a mailbox transfer shall be initiated. In case of an error the value *EC_NULL* will be returned.

Comment

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.

EC_T_MBXTFER_DESC

Mailbox transfer descriptor; the content determines which kind of mailbox transfer the object is related to and the maximum amount of data that can be transferred using the object.

```
typedef struct _EC_T_MBXTFER_DESC {
    EC_T_DWORD          dwMaxDataLen;
    EC_T_BYTE*          pbyMbxTferDescData;
} EC_T_MBXTFER_DESC;
```

Description

dwMaxDataLen

[in] Maximum amount of data bytes that shall be transferred using this object. A mailbox transfer type without data transfer will ignore this parameter.

pbyMbxTferDescData

[in] Pointer to byte stream carrying in and out data of mailbox content.

EC_T_MBXTFER

Mailbox transfer object.

```
typedef struct _EC_T_MBXTFER {
    EC_T_DWORD              dwClntId;
    EC_T_MBXTFER_DESC       MbxTferDesc;
    EC_T_MBXTFER_TYPE       eMbxTferType;
    EC_T_DWORD              dwDataLen;
    EC_T_BYTE*              pbyMbxTferData;
    EC_T_MBXTFER_STATUS     eTferStatus;
    EC_T_DWORD              dwErrorCode;
    EC_T_DWORD              dwTferId;
    EC_T_MBX_DATA           MbxData;
} EC_T_MBXTFER;
```

Description

dwClntId

[] Client ID.

MbxTferDesc

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

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.

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 (ecatNotify) this value will contain the amount of data that was actually transferred.

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.

eTferStatus

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

dwErrorCode

[out] Error code of a mailbox transfer that was terminated with error.

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.

MbxData

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

EC_T_MBXTFER_TYPE

Enumeration containing mailbox transfer types.

```
typedef enum _EC_T_MBXTFER_TYPE {
    eMbxTferType_COE_SDO_DOWNLOAD,      /* CoE SDO Download */
    eMbxTferType_COE_SDO_UPLOAD,        /* CoE SDO Upload */
    eMbxTferType_COE_GETODLIST,         /* CoE Get Object Dictionary */
    eMbxTferType_COE_GETOBDESC          /* CoE Get Object Description */
    eMbxTferType_COE_GETENTRYDESC,      /* CoE Get Object Entry Description */
    eMbxTferType_COE_EMERGENCY,         /* CoE Emergency Request */
    eMbxTferType_FOE_FILE_UPLOAD,       /* FOE Upload */
    eMbxTferType_FOE_FILE_DOWNLOAD,     /* FOE Download */
    eMbxTferType_FOE_SEG_DOWNLOAD       /* FoE Segmented Download */
    eMbxTferType_FOE_SEG_UPLOAD         /* FoE Segmented Upload */
    eMbxTferType_VOE_MBX_READ,          /* VoE read */
    eMbxTferType_VOE_MBX_WRITE,         /* VoE write */
    eMbxTferType_SOE_READREQUEST,       /* SoE read request */
    eMbxTferType_SOE_READRESPONSE,      /* SoE read response */
    eMbxTferType_SOE_WRITEREQUEST,      /* SoE write request */
    eMbxTferType_SOE_WRITERESPONSE,     /* SoE write response */
    eMbxTferType_SOE_NOTIFICATION,      /* SoE notification */
    eMbxTferType_SOE_EMERGENCY,         /* SoE emergency */
    eMbxTferType_AOE_READ,              /* AoE read  */
    eMbxTferType_AOE_WRITE,             /* AoE write  */
    eMbxTferType_AOE_READWRITE,         /* AoE read/write  */
    eMbxTferType_AOE_WRITECONTROL,      /* AoE write control  */
    eMbxTferType_RAWMBX                 /* Raw Mbx */
} EC_T_MBXTFER_TYPE;
```

EC_T_MBXTFER_STATUS
Enumeration containing mailbox status values.

```
enum _EC_T_MBXTFER_STATUS:
        eMbxTferStatus_Idle = 0,                    /* Mailbox transfer object not in use */
        eMbxTferStatus_Pend = 1,                    /* Mailbox transfer in process */
        eMbxTferStatus_TferDone = 2,                /* Mailbox transfer completed */
        eMbxTferStatus_TferReqError = 3,            /* Mailbox transfer request error */

        /* Transfer waiting for continue, object owned by application */
        eMbxTferStatus_TferWaitingForContinue = 4,
```

EC_T_MBX_DATA
Mailbox data in notifications.

```
typedef union _EC_T_MBX_DATA {
        EC_T_AOE_CMD_RESPONSE       AoE_Response;   /* AoE, Class A */
        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, Class A */
        EC_T_SOE_NOTIFICATION       SoE_Notification; /* SoE notification request */
        EC_T_SOE_EMERGENCY          SoE_Emergency;  /* SoE emergency request */
} EC_T_MBX_DATA;
```

4.8.3 ecatMbxTferAbort

Abort a running mailbox transfer.

```
EC_T_DWORD ecatMbxTferAbort (
    EC_T_MBXTFER* pMbxTfer
);
```

Parameters

pMbxTfer
[in] Pointer to the transfer object.

Return

EC_E_NOERROR if successful.

Comment

Currently only supported for FoE Transfer, CoE Download and CoE Upload.
This function may not be called from within the JobTask's context.

4.8.4 ecatMbxTferDelete

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

```
EC_T_VOID ecatMbxTferDelete (
    EC_T_MBXTFER* pMbxTfer
);
```

Parameters

pMbxTfer
[in] Pointer to the transfer object.

Comment

This function should not be called from within the JobTask's context.

4.8.5 ecatNotify – EC_NOTIFY_MBOXRCV

Indicates a mailbox transfer completion.

Parameters

pbyInBuf
[in] pMbxTfer – 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
[] Set to EC_NULL.

dwOutBufSize
[] Set to 0.

pdwNumOutData
[] Set to EC_NULL.

Comment

The element *pMbxTfer–>dwClntId* contains the corresponding ID of the client that is notified.
The element *pMbxTfer–>dwTferId* contains the corresponding transfer ID.

The transfer result is stored in *pMbxTfer->dwErrorCode*.

pMbxTfer->eTferStatus is *eMbxTferStatus_TferReqError* on error and *eMbxTferStatus_TferDone* on success. **In order to reuse the transfer object the application must set it back to *eMbxTferStatus_Idle*.**

The *pMbxTfer->eMbxTferType* element determines the mailbox transfer type (e.g. eMbxTferType_COE_SDO_DOWNLOAD for a completion of a CoE SDO download transfer).

4.8.6 ecatNotify – EC_NOTIFY_COE_INIT_CMD

Indicates a COE mailbox transfer completion during slave state transition.

Parameters

pbyInBuf
[in] pMbxTfer – 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
[] Set to EC_NULL.

dwOutBufSize
[] Set to 0.

pdwNumOutData
[] Set to EC_NULL.

Comment

The *pMbxTfer->MbxData.CoE_InitCmd* element of type *EC_T_MBX_DATA_COE_INITCMD* gives further information.

This init command as to be enable using **EC_IOCTL_SET_NOTIFICATION_ENABLED.**

MAX_STD_STRLEN = 80

```
typedef struct _EC_T_MBX_DATA_COE_INITCMD
{
        EC_T_SLAVE_PROP   SlaveProp;
        EC_T_DWORD dwHandle;
        EC_T_WORD   wTransition;
        EC_T_CHAR   szComment[MAX_STD_STRLEN];
        EC_T_DWORD dwErrorCode;
        EC_T_BOOL   bFixed;
        EC_T_BYTE   byCcs;
        EC_T_BOOL   bCompleteAccess;
        EC_T_WORD   wIndex;
        EC_T_BYTE   bySubIndex;
        EC_T_DWORD dwDataLen;
        EC_T_BYTE*   pbyData;
} EC_T_MBX_DATA_COE_INITCMD;
```

SlaveProp
[out] Slave properties
dwHandle
[out] Handle passed by EC_IOCTL_ADD_COE_INITCMD, otherwise zero
wTransition
[out] Transition, e.g. ECAT_INITCMD_I_P
dwErrorCode
out] InitCmd result
bFixed
[out] Fixed flag (ENI)
byCcs
[out] Client command specifier (read or write access)
bCompleteAccess
[out] Complete access
wIndex
[out] Object Index
bySubIndex
[out] Object SubIndex
dwDataLen
[out] InitCmd data length
pbyData
[out] InitCmd data

4.9 CanOpen over EtherCAT

4.9.1 ecatCoeSdoDownload

Performs a CoE SDO download to an EtherCAT slave device.

```
EC_T_DWORD ecatCoeSdoDownload(
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTE byObSubIndex,
    EC_T_BYTE* pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout,
    EC_T_DWORD dwFlags
);
```

Parameters

dwSlaveId
[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used.
wObIndex
[in] Object index.
byObSubIndex
[in] Object sub index. 0 or 1 if Complete Access.
pbyData
[in] Data to be transferred.
dwDataLen
[in] Data size of pbyData
dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time.
dwFlags
[in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Return

EC_E_NOERROR or error code, e.g.:
EC_E_SDO_ABORTCODE_INDEX if wObIndex, byObSubIndex invalid.

Comment

This function may not be called from within the JobTask's context.

4.9.2 ecatCoeSdoDownloadReq

Initiates a CoE SDO download to an EtherCAT slave device and returns immediately.

```
EC_T_DWORD ecatCoeSdoDownloadReq (
    EC_T_MBXTFER* pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTE byObSubIndex,
    EC_T_DWORD dwTimeout,
    EC_T_DWORD dwFlags
);
```

Parameters

pMbxTfer
[in] Pointer to the corresponding mailbox transfer object created with ecatMbxTferCreate
The SDO data to write have to be stored at pMbxTfer->pbyMbxTferData.

dwSlaveId
[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used.

wObIndex
[in] Object index.

byObSubIndex
[in] Object sub index. 0 or 1 if Complete Access.

dwTimeout
[in] Timeout in milliseconds.

dwFlags
[in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Return

EC_E_NOERROR or error code

Comment

The amount of data bytes to write has to be stored in pMbxTfer->dwDataLen. A unique transfer ID must be written into pMbxTfer–>dwTferId. A ecatNotify – EC_NOTIFY_MBOXRCV is given on completion, see below.

4.9.3 ecatNotify – eMbxTferType_COE_SDO_DOWNLOAD

SDO download transfer completion.

Parameters

pbyInBuf
[in] pMbxTfer – 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
[] Set to EC_NULL.

dwOutBufSize
[] Set to 0.

pdwNumOutData
[] Set to EC_NULL.

Comment

The corresponding transfer ID can be found in pMbxTfer–>dwTferId. The transfer result is stored in pMbxTfer->dwErrorCode.

4.9.4 ecatCoeSdoUpload

Performs a CoE SDO upload from an EtherCAT slave device to the master.

```
EC_T_DWORD ecatCoeSdoUpload(
    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
);
```

Parameters

dwSlaveId
[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used.
wObIndex
[in] Object index.
byObSubIndex
[in] Object sub index. 0 or 1 if Complete Access.
pbyData
[out] Data buffer to upload data to.
dwDataLen
[in] Size of data buffer carried in pbyData
pdwOutDataLen
[out] Pointer returning size of data uploaded from slave.
dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time.
dwFlags
[in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Return

EC_E_NOERROR or error code, e.g.:
EC_E_SDO_ABORTCODE_INDEX if wObIndex, byObSubIndex invalid.

Comment

This function may not be called from within the JobTask's context.

4.9.5 ecatCoeSdoUploadReq

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

```
EC_T_DWORD ecatCoeSdoUploadReq (
    EC_T_MBXTFER* pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTE byObSubIndex,
    EC_T_DWORD dwTimeout,
    EC_T_DWORD dwFlags
);
```

Parameters

pMbxTfer
[in] Pointer to the corresponding mailbox transfer object created with ecatMbxTferCreate

dwSlaveId
[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used.

wObIndex
[in] Object index

byObSubIndex
[in] Object sub index. 0 or 1 if Complete Access.

dwTimeout
[in] Timeout in milliseconds.

dwFlags
[in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Return

EC_E_NOERROR or error code

Comment

The amount of data bytes to read has to be stored in pMbxTfer->dwDataLen. A unique transfer ID must be written into pMbxTfer–>dwTferId. A ecatNotify – EC_NOTIFY_MBOXRCV is given on completion, see below.

4.9.6 ecatNotify – eMbxTferType_COE_SDO_UPLOAD

SDO upload transfer completion.

Parameters

pbyInBuf
[in] pMbxTfer – Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object. The SDO data are stored in pMbxTfer->pbyMbxTferData.

dwInBufSize
[in] Size of the transfer object in bytes.

pbyOutBuf
[] Set to EC_NULL.

dwOutBufSize
[] Set to 0.

pdwNumOutData
[] Set to EC_NULL.

Comment

The corresponding transfer ID can be found in pMbxTfer–>dwTferId. The transfer result is stored in pMbxTfer->dwErrorCode. The SDO data stored in pMbxTfer->pbyMbxTferData may have to be buffered by the client. After ecatNotify returns the pointer and thus the data is invalid. Access to the memory area pointed to by pMbxTfer->pbyMbxTferData after returning from ecatNotify is illegal and the results are undefined.

4.9.7 ecatCoeGetODList

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

```
EC_T_DWORD ecatGetODList (
    EC_T_MBXTFER* pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_COE_ODLIST_TYPE eListType,
    EC_T_DWORD dwTimeout,
);
```

Parameters

pMbxTfer

[in] Pointer to the corresponding mailbox transfer object.

dwSlaveId

[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used.

eListType

[in] Object list type.
eODListType_Lengths:Lengths of each list type
eODListType_ALL: Get all objects
eODListType_RxPdoMap: Get PDO mappable objects
eODListType_TxPdoMap: Get objects that can be changed
eODListType_StoredFRepl: Get objects that are stored for a device replacement
eODListType_StartupParm: Get objects that can be used as startup parameter

dwTimeout

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

Return

EC_E_NOERROR or error code

Comment

A unique transfer ID must be written into pMbxTfer–>dwTferId. A ecatNotify – EC_NOTIFY_MBOXRCV is given when new data arrives, see below.
This function may not be called from within the JobTask's context.

4.9.8 ecatNotify – eMbxTferType_COE_GETODLIST

Object dictionary list upload transfer completion.

Parameters

pbyInBuf

[in] pMbxTfer – Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.
The object list is stored in pMbxTfer->MbxData.CoE_ODList.

dwInBufSize

[in] Size of the transfer object in bytes.

pbyOutBuf

[] Set to EC_NULL.

dwOutBufSize

[] Set to 0.

pdwNumOutData

[] Set to EC_NULL.

Comment

The corresponding transfer ID can be found in pMbxTfer–>dwTferId. The transfer result is stored in pMbxTfer->dwErrorCode. The object list stored in pMbxTfer->MbxData.CoE_ODList may have to be buffered by the client. After ecatNotify returns the pointer and thus the data is invalid. Access to this element after returning from ecatNotify is invalid and the results are undefined.

EC_T_COE_ODLIST (pMbxTfer-> MbxData.CoE_ODList)
List containing object Ids available in the slave

```
typedef struct _ EC_T_COE_ODLIST {
    EC_T_COE_ODLIST_TYPE    eOdListType;    /* list type */
    EC_T_WORD               wLen;           /* amount of object IDs */
    EC_T_WORD*              pwOdList;       /* array containing object IDs */
} EC_T_COE_ODLIST;
```

EC_T_COE_ODLIST_TYPE
List type enumeration.

```
typedef enum _ EC_T_COE_ODLIST_TYPE {
  eODListType_Lengths       = 0,  /*< lengths of each list type */
  eODListType_ALL           = 1,  /* list contains all objects */
  eODListType_RxPdoMap      = 2,  /* list with PDO mappable objects*/
  eODListType_TxPdoMap      = 3,  /* list with objects that can be changed */
  eODListType_StoredFRepl   = 4,  /* only stored for a device replacement objects */
  eODListType_StartupParm   = 5,  /* only startup parameter objects */
} EC_T_COE_ODLIST_TYPE;
```

4.9.9 ecatCoeGetObjectDesc

Determines the description of a specific object.

```
EC_T_DWORD ecatGetObjectDesc (
    EC_T_MBXTFER* pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_DWORD dwTimeout,
);
```

Parameters

pMbxTfer
[in] Pointer to the corresponding mailbox transfer object.
dwSlaveId
[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used.
wObIndex
[in] Object index.
dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time. If the timeout value is set to EC_NOWAIT the function will return immediately.

Return

EC_E_NOERROR or error code

Comment

A unique transfer ID must be written into pMbxTfer–>dwTferId. A ecatNotify – EC_NOTIFY_MBOXRCV is given when new data arrives, see below.
This function may not be called from within the JobTask's context.

4.9.10 ecatNotify – eMbxTferType_COE_GETOBDESC

Completion of a SDO information service transfer to get a object description.

Parameters

pbyInBuf

[in] pMbxTfer – Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.
The object description is stored in pMbxTfer->MbxData.CoE_ObDesc.

dwInBufSize

[in] Size of the transfer object in bytes.

pbyOutBuf

[] Set to EC_NULL.

dwOutBufSize

[] Set to 0.

pdwNumOutData

[] Set to EC_NULL.

Comment

The corresponding transfer ID can be found in pMbxTfer–>dwTferId. The transfer result is stored in pMbxTfer->dwErrorCode. The object description stored in pMbxTfer->MbxData.CoE_ObDesc may have to be buffered by the client. After ecatNotify returns the pointer and thus the data is invalid. Access to this element after returning from ecatNotify is invalid and the results are undefined.

EC_T_COE_OBDESC (pMbxTfer-> MbxData.CoE_ObDesc)
Object description. A more detailed description of the values for data type, object code etc. can be found in the EtherCAT specification ETG.1000, section 5. See also EcCommon.h, DEFTYPE_BOOLEAN, …. .

```
typedef struct _ EC_T_COE_OBDESC {
    EC_T_WORD       wObIndex;           /* Index in the object dictionary */
    EC_T_WORD       wDataType;          /* Data type of the object */
    EC_T_BYTE       byObjCode;          /* Object code, see Table 62, ETG.1000 section 6 */
    EC_T_BYTE       byObjCategory;      /* Object category */
    EC_T_BYTE       byMaxNumSubIndex;   /* Maximum sub index number /
    EC_T_WORD       wObNameLen;         /* Length of the object name */
    EC_T_CHAR*      pchObName;          /* Object name (not NULL terminated!) */
} EC_T_COE_OBDESC;
```

4.9.11 ecatCoeGetEntryDesc

Determines the description of a specific object entry.

```
EC_T_DWORD ecatCoeGetEntryDesc (
    EC_T_MBXTFER* pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTE byObSubIndex,
    EC_T_BYTE byValueInfo,
    EC_T_DWORD dwTimeout,
);
```

Parameters

pMbxTfer
[in] Pointer to the corresponding mailbox transfer object.

dwSlaveId
[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used.

wObIndex
[in] Object index.

byObSubIndex
[in] Object sub index.

byValueInfo
[in] Bit mask to define which information to determine. The bit values are defined as follows:

EC_COE_ENTRY_ObjAccess	→ Obect access rights
EC_COE_ENTRY_ObjCategory	→ Object category
EC_COE_ENTRY_PdoMapping	→ Information if the object is PDO mappable
EC_COE_ENTRY_UnitType	→ Unit
EC_COE_ENTRY_DefaultValue	→ Default value
EC_COE_ENTRY_MinValue	→ Minimum value
EC_COE_ENTRY_MaxValue	→ Maximum value

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

Return

EC_E_NOERROR or error code

Comment

A unique transfer ID must be written into pMbxTfer–>dwTferId. A ecatNotify – EC_NOTIFY_MBOXRCV is given when new data arrives, see below.
This function may not be called from within the JobTask's context.

4.9.12 ecatNotify – eMbxTferType_COE_GETENTRYDESC

Completion of a SDO information service transfer to get a object entry description.

Parameters

pbyInBuf

[in] pMbxTfer – Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.
The object description is stored in pMbxTfer->MbxData.CoE_EntryDesc.

dwInBufSize

[in] Size of the transfer object in bytes.

pbyOutBuf

[] Set to EC_NULL.

dwOutBufSize

[] Set to 0.

pdwNumOutData

[] Set to EC_NULL.

Comment

The corresponding transfer ID can be found in pMbxTfer–>dwTferId. The transfer result is stored in pMbxTfer->dwErrorCode. The object entry description stored in pMbxTfer->MbxData.CoE_EntryDesc may have to be buffered by the client. After ecatNotify returns the pointer and thus the data is invalid. Access to this element after returning from ecatNotify is invalid and the results are undefined.

EC_T_COE_ENTRYDESC (pMbxTfer-> MbxData.CoE_EntryDesc)
Object entry description. A more detailed description of the values can be found in the EtherCAT specification ETG.1000, section 5.

```
typedef struct _ EC_T_COE_ENTRYDESC {
    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 to define which information is
                                               available */
    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 */
    EC_T_BOOL       bRxPdoMapping;          /* Is the object PDO-mappable? */
    EC_T_BOOL       bTxPdoMapping;          /* Can the PDO be changed */
    EC_T_BOOL       bObCanBeUsedForBackup;      /* Parameter may be back upped */
    EC_T_BOOL       bObCanBeUsedForSettings;    /* Parameter may be used for Settings */
    EC_T_WORD       wReserved;              /* Unit*/
    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) */
} EC_T_COE_ENTRYDESC;
```

See szUnitType, szDefaultValue, szMinValue, szMaxValue, szDescription in CoeReadObjectDictionary in ecatDemoCommon.cpp as an example for evaluating EC_T_COE_ENTRYDESC.pbyData.

4.9.13 CoE Emergency (ecatNotify – eMbxTferType_COE_EMERGENCY)

Indication of a CoE emergency request. A ecatNotify – EC_NOTIFY_MBOXRCV is given with pMbxTfer->eMbxTferType = eMbxTferType_COE_EMERGENCY.

Parameters

pbyInBuf

[in] pMbxTfer – Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.
The emergency request is stored in pMbxTfer->MbxData.CoE_Emergency.

dwInBufSize

[in] Size of the transfer object in bytes.

pbyOutBuf

[] Set to EC_NULL.

dwOutBufSize

[] Set to 0.

pdwNumOutData

[] Set to EC_NULL.

Comment

In case of an emergency notification all registered clients will get this notification.
The corresponding mailbox transfer object will be created inside the EtherCAT master. The content in pMbxTfer–>dwTferId is undefined as it is not needed by the client and the master. The transfer result is stored in pMbxTfer->dwErrorCode.
The emergency data stored in pMbxTfer->MbxData.CoE_Emergency may have to be buffered by the client. After ecatNotify returns the pointer and thus the data is invalid. Access to this element after returning from ecatNotify is invalid and the results are undefined.

EC_T_COE_EMERGENCY (pMbxTfer-> MbxData.CoE_Emergency)
Emergency description. A more detailed description of the values can be found in the EtherCAT specification ETG.1000, section 5.

```
typedef struct _ EC_T_COE_ENTRYDESC {
    EC_T_WORD       wErrorCode;       /* Error code according to EtherCAT specification */
    EC_T_BYTE       byErrorRegister;  /* Error register */
    EC_T_BYTE       abyData[5];       /* Error data */
    EC_T_WORD       wStationAddress;  /* Slave node address of the faulty slave*/
} EC_T_COE_EMERGENCY;
```

4.9.14 CoE Abort (ecatNotify – EC_NOTIFY_MBSLAVE_COE_SDO_ABORT)

The application can abort asynchronous CoE Uploads and Downloads, see ecatMbxTferAbort.

The slave may abort CoE Uploads and Downloads which is indicated at the return code of ecatCoeSdoUpload, … .

The notification EC_NOTIFY_MBSLAVE_COE_SDO_ABORT is raised if an SDO transfer aborts while sending init commands.

Detailed error information is stored in structure EC_T_MBOX_SDO_ABORT_DESC of the union element SdoAbortDesc.

```
typedef struct _EC_T_MBOX_SDO_ABORT_DESC
{
    EC_T_SLAVE_PROP SlaveProp;      /*< slave properties */
    EC_T_DWORD      dwErrorCode;    /*< error code EC_E_ */
    EC_T_WORD       wObjIndex;      /*< SDO object index */
    EC_T_BYTE       bySubIndex;     /*< SDO object sub index */
} EC_T_MBOX_SDO_ABORT_DESC;
```

4.10 Servo Drive Profil according to IEC61491 over EtherCAT (SoE)

The SoE Service Channel (SSC) is equivalent to the IEC61491 Service Channel used for non-cyclic data exchange. The SSC uses the EtherCAT mailbox mechanism. It allows accessing IDNs and their elements.

For extended informations about SoE see the IEC IEC61800-7-300 document 22G185eFDIS.

Following services are available:

Write IDN:
With ecatSoeWrite the data and elements of an IDN which are writeable can be written.

Read IDN:
With ecatSoeRead the data and elements of an IDN can be read.

Procedure command Execution:
With ecatSoeWrite also procedure commands can be started. Procedure commands are special IDNs, which invokes fixed functional processes. When proceeding is finished, a notification will be received. To abort a running command execution ecatSoeAbortProcCmd has to be called.

Notification:
In case of an notification all registered clients will get this notification. A notification will be received when proceeding of a command has finished. To register a client ecatRegisterClient must be called.

Emergency:
The main purpose of this service is to provide additional information about the slave for debugging and maintenance. In case of an emergency, all registered clients will get notified. To register a client ecatRegisterClient must be called.

Abbreviations:

IDN:
identification **n**umber: Designation of operating data under which a data block is preserved with its attribute, name, unit, minimum and maximum input values, and the data.

SoE:
IEC 61491 **S**ervo Profile **o**ver **E**therCAT

SSC:
SoE Service Channel (non-cyclic data exchange)

4.10.1 SoE ElementFlags

With the ElementFlags each element of an IDN can be addressed. The ElementFlags indicating which elements of an IDN are read or written. The ElementFlags indicating which data will be transmitted in the SoE data buffer.

SOE_BM_ELEMENTFLAG_DATATSTATE
Shall be set in case of Notify SoE Service Channel Command Execution.

SOE_BM_ELEMENTFLAG_NAME
Name of operation data. The name consist of 64 octets maximum.

SOE_BM_ELEMENTFLAG_ATTRIBUTE
Attribute of operation data. The attribute contain all information which is needed to display operation data intelligibly.

SOE_BM_ELEMENTFLAG_UNIT
Unit of operation data.

SOE_BM_ELEMENTFLAG_MIN
The IDN minimum input value shall be the smallest numerical value for the operation data which the slave is able to process.

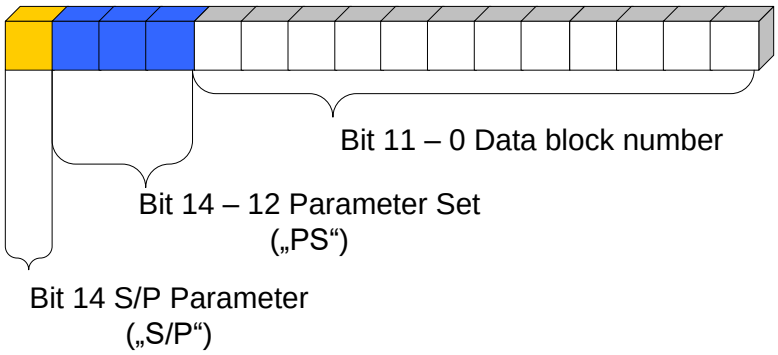
SOE_BM_ELEMENTFLAG_MAX
The IDN maximum input value shall be the largest numerical value for the operation data which the slave is able to process.

SOE_BM_ELEMENTFLAG_VALUE
Operation data.

SOE_BM_ELEMENTFLAG_DEFAULT
The IDN default value.

4.10.2 SoE IDN coding

The parameter addressing area consist of 4096 different standard IDNs, each with 8 parameter sets and 4096 manufacturer specific IDNs, each with 8 parameter sets.



The Control unit cycle time (TNcyc) which is an standard IDN **S-0-0001** equates to ***wIDN* = 0x1**
The first manufacturer specific IDN **P-0-0001** equates to ***wIDN* = *0x*8001**

4.10.3 ecatSoeWrite

Performs an SoE SSC Write service which download data to an EtherCAT slave device. The function returns after dwTimeout in msecs are expired or download is completed successfully (Write response is received).
EcatSoeWrite can also perform a SoE SSC Procedure Command. After a procedure command has started, the slave generates a normal SSC Write Response, and the function returns.

```
EC_T_DWORD  ecatSoeWrite(
    EC_T_DWORD      dwSlaveId,
    EC_T_BYTE       byDriveNo,
    EC_T_BYTE*      pbyElementFlags,
    EC_T_WORD       wIDN,
    EC_T_BYTE*      pbyData,
    EC_T_DWORD      dwDataLen,
    EC_T_DWORD      dwTimeout
);
```

Parameters

dwSlaveId
[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used.
byDriveNo
[in] Number of the drive inside the slave device that is addressed.
pbyElementFlags
[in/out] ElementFlag for each element of an IDN indicating which elements of the object are written. For more informations see 4.10.1 SoE ElementFlags.
wIDN
[in] IDN (see 4.10.2 SoE IDN) of the object which is addressed in the slave device.
pbyData
[in] Data buffer include the data of the element to be written.
dwDataLen
[in] Data Buffer length in bytes.
dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time.

Return

EC_E_NOERROR or error code

Comment

If the data to be sent with the write service exceeds the mailbox size, the data will be sent fragmented. The fragmented write operation consists of several Write SSC Fragment Services. Therefore the selected Timeout should be increasing with the count of fragments.
This function may not be called from within the JobTask's context.

4.10.4 ecatSoeWriteReq

Requests an SoE SSC Write and returns immediately. See also ecatSoeWrite.

```
EC_T_DWORD  ecatSoeWriteReq(
    EC_T_MBXTFER*   pMbxTfer
    EC_T_DWORD      dwSlaveId,
    EC_T_BYTE       byDriveNo,
    EC_T_BYTE*      pbyElementFlags,
    EC_T_WORD       wIDN,
    EC_T_DWORD      dwTimeout
);
```

Parameters

pMbxTfer
[in] Handle to the Mailbox Transfer Object. See chapter 4.8 "EtherCAT Mailbox Transfer".
The other parameters are according to ecatSoeWrite.

Comment

This function may be called from within the JobTask's context. See also ecatSoeWrite.

4.10.5 ecatSoeRead

Performs an SoE SCC Read service which upload data from an EtherCAT SoE slave device. The function returns after dwTimeout in msecs are expired or upload is completed successfully (Read response is received). The received data can resist of several fragments. The reserved data buffer (pbyData) must have space for all received data segments.

```
EC_T_DWORD ecatSoeRead(
    EC_T_DWORD      dwSlaveId,
    EC_T_BYTE       byDriveNo,
    EC_T_BYTE *     pbyElementFlags,
    EC_T_WORD       wIDN,
    EC_T_BYTE*      pbyData,
    EC_T_DWORD      dwDataLen,
    EC_T_DWORD*     pdwOutDataLen,
    EC_T_DWORD      dwTimeout
);
```

Parameters

dwSlaveId
[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used.

byDriveNo
[in] Number of the drive inside the slave device that is addressed.

pbyElementFlags
[in/out] ElementFlag for each element of an IDN indicating which elements of the object are read. For more informations see 4.10.1 SoE ElementFlags.

wIDN
[in] IDN (see 4.10.2 SoE IDN) of the object which is addressed in the slave device.

pbyData
[out] Data buffer to store uploaded file to.

dwDataLen
[in] Data Buffer length in bytes.

pdwOutDataLen
[out] Pointer returning size of data uploaded from slave.

dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time.

Return

EC_E_NOERROR or error code

Comment

If the data to be read with the read service exceeds the mailbox size, the received data will be fragmented. The fragmented read operation sonsists of severel Read SSC Fragment Services. Therefore the selected Timeout should be increasing with the count of fragments.
This function may not be called from within the JobTask's context.

4.10.6 ecatSoeReadReq

Requests an SoE SSC Read and returns immediately. See also ecatSoeRead.

```
EC_T_DWORD  ecatSoeReadReq(
    EC_T_MBXTFER*   pMbxTfer
    EC_T_DWORD      dwSlaveId,
    EC_T_BYTE       byDriveNo,
    EC_T_BYTE *     pbyElementFlags,
    EC_T_WORD       wIDN,
    EC_T_DWORD      dwTimeout
);
```

Parameters

pMbxTfer
[in] Handle to the Mailbox Transfer Object. See chapter 4.8 "EtherCAT Mailbox Transfer".

The other parameters are according to ecatSoeRead.

Comment

This function may be called from within the JobTask's context. See also ecatSoeRead.

4.10.7 ecatSoeAbortProcCmd

Abort SSC Procedure Command sequence. A Procedure Command take up some time. After a procedure command has started, the slave generates a normal SSC Write Response. The end of a procedure command is indicated by the Notify SSC Command Execution Service. To abort a procedure command earlier ecatSoeAbortProcCmd can be called.

```
EC_T_DWORD ecatSoeAbortProcCmd(
    EC_T_DWORD      dwSlaveId,
    EC_T_BYTE       byDriveNo,
    EC_T_BYTE*      pbyElementFlags,
    EC_T_WORD       wIDN,
    EC_T_DWORD      dwTimeout,
);
```

Parameters

dwSlaveId

[in] EtherCAT slave ID. To determine the slave ID the function *ecatGetSlaveId* has to be used. Should be the same as in the request.

byDriveNumber

[in] Number of the drive inside the slave device that is addressed. Should be the same as in the request.

pbyElementFlags

[in/out] ElementFlag for each element of an IDN indicating which elements of the object are written. Should be the same as in the request. For more informations see 4.10.1 SoE ElementFlags.

wIDN

[in] IDN (see 4.10.2 SoE IDN) of the object which is addressed in the slave device. Should be the same as in the request.

dwTimeout

[in] Timeout in milliseconds. The function will block at most for this time.

Return

EC_E_NOERROR or error code

Comment

This function may not be called from within the JobTask's context.

4.10.8 Error notifications

Following error notifications are defined for SoE.

4.10.8.1 EC_NOTIFY_SOE_MBXSND_WKC_ERROR

This error will be indicated in case the working counter of a SoE mailbox write command was not set to the expected value of 1.
Detailed error information is stored in structure EC_T_WKCERR_DESC of the union element WkcErrDesc. The structure member SlaveProp contains information about the corresponding slave device.

4.10.8.2 EC_NOTIFY_SOE_WRITE_ERROR

This error will be indicated in case SoE mailbox write command responded with an error.
Detailed error information is stored in structure EC_T_INITCMD_ERR_DESC of the union element InitCmdErrDesc:

4.11 Vendor specific protocol over EtherCAT (VoE)

The mailbox protocol VoE is part of the Class A user manual.

4.12 Automation Device Specification over EtherCAT (AoE)

The mailbox protocol AoE is part of the Class A user manual.

4.13 Raw command transfer

4.13.1 ecatTferSingleRawCmd

Transfers a single raw EtherCAT command to one or multiple slaves and waits for the result. Using this function it is possible exchange arbitrary data between the master and the slaves.
When the master receives the response to the queued frame it raises EC_NOTIFY_RAWCMD_DONE to all clients. This command may be called after ecatConfigureMaster and the master is in a stable state (no pending transition), if master is not stable, EC_E_BUSY is returned. If this command is called before ecatConfigureMaster, no slave objects do exist within the master, which prevents the master from sending data.

```
EC_T_DWORD ecatTferSingleRawCmd (
    EC_T_DWORD      dwSlaveId,
    EC_T_BYTE       byCmd,
    EC_T_DWORD      dwMemoryAddress,
    EC_T_VOID*      pvData,
    EC_T_WORD       wLen,
    EC_T_DWORD      dwTimeout,
);
```

Parameters

dwSlaveId
[in] Slave ID (ecatGetSlaveId(0) in case of broadcast commands).
byCmd
[in] EtherCAT command.
dwMemoryAddress
[in] Slave memory address, depending on the command to be sent this is either a physical or logical address.
pvData
[in,out] Pointer to the data to transfer (read and/or write).
wLen
[in] Number of bytes to transfer.
dwTimeout
[in] Timeout in msec.

Return

EC_E_NOERROR if successful.
EC_E_BUSY another transfer request is already pending.
EC_E_NOTFOUND if the slave with ID dwSlaveId does not exist.
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_TIMEOUT if the slave did not respond to the command.
EC_E_BUSY if the master or the corresponding slave is currently changing its operational state .
EC_E_INVALIDPARM if the command is not supported or the timeout value is set to EC_NOWAIT.
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.

Comment

This function blocks until the command is completely processed. In case of read commands the slave data will be written back into the given memory area (pvData).
If a timeout occurs (e.g. due to a bad line quality) the corresponding frame will be sent again.
The timeout value and retry counter can be set using the master configuration parameters dwEcatCmdTimeout and dwEcatCmdMaxRetries. The call will return in any case (without waiting for the number of retries specified in dwEcatCmdMaxRetries) if the time determined with the dwTimeout parameter elapsed.
The following EtherCAT commands are supported:

- eRawCmd_APRD Auto Increment Physical Read (avoid to use, see below!)
- eRawCmd_APWR Auto Increment Physical Write (avoid to use, see below!)
- eRawCmd_APRW Auto Increment Physical Read/Write (avoid to use, see below!)
- eRawCmd_FPRD Fixed addressed Physical Read
- eRawCmd_FPWR Fixed addressed Physical Write
- eRawCmd_FPRW Fixed addressed Physical Read/Write
- eRawCmd_BRD Broadcast (wire or'ed) Read
- eRawCmd_BWR Broadcast Write
- eRawCmd_BRW Broadcast Read/Write
- eRawCmd_LRD Logical Read
- eRawCmd_LWR Logical Write
- eRawCmd_LRW Logical Read/Write
- eRawCmd_ARMW Auto Increment Physical Read, multiple Write

Caveat: Using auto increment addressing (APRD, APWR, APRW) may lead to unexpected results in case the selected slave does not increment the working counter. In such cases the EtherCAT command would be handled by the slave directly behind the selected one.

This function may not be called from within the JobTask's context.

4.13.2 ecatClntQueueRawCmd

Transfers a raw EtherCAT command to one or multiple slaves. Using this function it is possible to exchange data between the master and the slaves. When the response to the queued frame is received, the notification EC_NOTIFY_RAWCMD_DONE is given for the appropriate client. This command may be called after ecatConfigureMaster and the master is in a stable state (no pending transition), if master is not stable, EC_E_BUSY is returned. If this command is called before ecatConfigureMaster, no slave objects do exist within the master, which prevents the master from sending data.

```
EC_T_DWORD ecatClntQueueRawCmd (
    EC_T_DWORD      dwClntId,
    EC_T_DWORD      dwSlaveId,
    EC_T_WORD       wInvokeId,
    EC_T_BYTE       byCmd,
    EC_T_DWORD      dwMemoryAddress,
    EC_T_VOID*      pvData,
    EC_T_WORD       wLen
);
```

Parameters

dwClntd
[in] ID of the client to be notified (0 if all registered clients shall be notified).

dwSlaveId
[in] Slave ID (the master ID which is returned by ecatGetSlaveId(0) shall be used for broadcast commands like eRawCmd_BRD).

wInvokeId
[in] Invoke ID to reassign the results to the sent cmd.

byCmd
[in] EtherCAT command, see below

dwMemoryAddress
[in] Slave memory address, depending on the command to be sent this is either a physical or logical address.

pvData
[in] Pointer to the data to transfer (read and/or read/write). In case a read-only command is queued (e.g. APRD) this pointer should be set to a value of EC_NULL.

wLen
[in] Number of bytes to transfer.

Return

EC_E_NOERROR if successful.
EC_E_NOTFOUND if the slave with ID dwSlaveId does not exist.
EC_E_BUSY if the master or the corresponding slave is currently changing its operational state.
EC_E_INVALIDPARM if the command is not supported.
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.

Comment
This function queues a single EtherCAT command. Queued raw commands will be sent after sending cyclic process data values.
If a timeout occurs (e.g. due to a bad line quality) the corresponding frame will be sent again.
The timeout value and retry counter can be set using the master configuration parameters dwEcatCmdTimeout and dwEcatCmdMaxRetries.
The following EtherCAT commands are supported:

- eRawCmd_APRD Auto Increment Physical Read (avoid to use, see below!)
- eRawCmd_APWR Auto Increment Physical Write (avoid to use, see below!)
- eRawCmd_APRW Auto Increment Physical Read/Write (avoid to use, see below!)
- eRawCmd_FPRD Fixed addressed Physical Read
- eRawCmd_FPWR Fixed addressed Physical Write
- eRawCmd_FPRW Fixed addressed Physical Read/Write
- eRawCmd_BRD Broadcast (wire or'ed) Read
- eRawCmd_BWR Broadcast Write
- eRawCmd_BRW Broadcast Read/Write
- eRawCmd_LRD Logical Read
- eRawCmd_LWR Logical Write
- eRawCmd_LRW Logical Read/Write
- eRawCmd_ARMW Auto Increment Physical Read, multiple Write

Caveat: Using auto increment addressing (APRD, APWR, APRW) may lead to unexpected results in case the selected slave does not increment the working counter. In such cases the EtherCAT command would be handled by the slave directly behind the selected one.

This function may not be called from within the JobTask's context.

4.13.3 ecatQueueRawCmd

This function is identical to ecatClntQueueRawCmd except that the notification function of all registered clients will be called.

```
EC_T_DWORD ecatQueueRawCmd (
    EC_T_DWORD      dwSlaveId,
    EC_T_WORD       wInvokeId,
    EC_T_BYTE       byCmd,
    EC_T_DWORD      dwMemoryAddress,
    EC_T_VOID*      pvData,
    EC_T_WORD       wLen
);
```

For more information, see chapter 4.13.2

This function may not be called from within the JobTask's context.

4.13.4 ecatNotify – EC_NOTIFY_RAWCMD_DONE

This notification is given when the response to an ecatQueueRawCmd is received.

Parameters

pbyInBuf
[in] Pointer to EC_T_RAWCMDRESPONSE_NTFY_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

EC_T_RAWCMDRESPONSE_NTFY_DESC

```
typedef struct _EC_T_RAWCMDRESPONSE_NTFY_DESC{
    EC_T_DWORD          dwInvokeId;
    EC_T_DWORD          dwResult;
    EC_T_DWORD          dwWkc;
    EC_T_DWORD          dwCmdIdx;
    EC_T_DWORD          dwAddr;
    EC_T_DWORD          dwLength;
    EC_T_BYTE*          pbyData;
} EC_T_RAWCMDRESPONSE_NTFY_DESC;
```

Description

dwInvokeId
[in] Invoke Id from callee (ecatQueueRawCmd). Only lower 16 bits are relevant.
dwResult
[in] EC_E_NOERROR on success, error code otherwise
dwWkc
[in] Received working counter
dwCmdIdx
[in] Command Index Field
Format:

```
0000 0000 0000 0000 iiii iiii cccc cccc
rrrr rrrr rrrr rrrr .... .... .... ....    Reserved
.... .... .... .... .... .... cccc cccc    Command
.... .... .... .... iiii iiii .... ....    Index
```

dwAddr
[in] Address Field
Format:

```
oooo oooo oooo oooo pppp pppp pppp pppp
.... .... .... .... pppp pppp pppp pppp    ADP/Slave address
oooo oooo oooo oooo .... .... .... ....    ADO/Offset
```

dwLength
[in] Length of data portion (11 relevant bits)
pbyData
[in] Pointer to data portion within a PDU. The callback function has to store the data into application memory, the data pointer will be invalid after returning from the callback.

4.14 Distributed Clocks (DC)

Distributed Clocks is part of the Class A user manual.

4.15 EtherCAT Bus Scan

The acontis EtherCAT Master stack supports scanning the EtherCAT Bus to determine which devices are available. This functionality may be used to create a bus configuration description file (XML).

See 4.3.5 ecatScanBus.

4.15.1 ecatIoControl – EC_IOCTL_SB_ENABLE

Enables Busscan support.

Parameters

pbyInBuf
[in] Pointer to Timeout Parameter Value in MSec (EC_T_DWORD). Timeout Parameter is used for timeout during Bus Topology determination.

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

pbyOutBuf
[] Should be set to EC_NULL.

dwOutBufSize
[] Should be set to 0.

pdwNumOutData
[] Should be set to EC_NULL.

Comment

The Scanbus support is enabled by default.

4.15.2 ecatIoControl – EC_IOCTL_SB_RESTART

This call will restart the bus scanning cycle. On completion the Notfication EC_NOTIFY_SB_STATUS is given.

Parameters

–

Comment

The timeout value given by EC_IOCTL_SB_ENABLE will be used.
This function may be called prior to running ecatConfigureMaster(). In such a case a first bus scan will be executed before master configuration. This feature may be used to dynamically create or adjust the XML configuration file. When issuing this IoControl, the application has to take care ecatExecJob is called cyclically to trigger master state machines, timers, send acyc and receive frames accordingly.

4.15.3 ecatIoControl – EC_IOCTL_SB_STATUS_GET

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

Parameters

pbyInBuf
[] Should be set to EC_NULL.
dwInBufSize
[] 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.

Comment

Description see 4.15.12 “ecatNotify – EC_NOTIFY_SB_STATUS”.

4.15.4 ecatIoControl – EC_IOCTL_SB_SET_BUSCNF_VERIFY_PROP

This call will determine whether a single slave property will be used to verify the bus configuration.
The scan bus verifies the vendor id and product code of the slave device if it is not disabled using the IO-Control “EC_IOCTL_SB_SET_BUSCNF_VERIFY_PROP”. The ENI file may contain other EEPROM checks like Revision or Serial Number realized using init commands, they are not part of the scan bus.

Parameters

pbyInBuf
[in] Pointer to EC_T_SCANBUS_PROP_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

Comment

```
typedef struct _EC_T_SCANBUS_PROP_DESC
{
   EC_T_eEEPENTRY  eEEPROMEntry;
   EC_T_DWORD      dwVerify;
} EC_T_SCANBUS_PROP_DESC;
```

Description

eEEPROMEntry
[in] E²PROM entry (slave property) to add.
dwVerify
[in] if set to EC_TRUE the actual slave property (stored in the E²PROM) will be compared with the appropriate value in the XML configuration file.

EC_T_eEEPENTRY

```
typedef enum _EC_T_eEEPENTRY{
    eEEP_VendorId          = 0x0008, /* Checked by scan bus */
    eEEP_ProductCode       = 0x000A, /* Checked by scan bus */
    eEEP_RevisionNumber    = 0x000C, /* Checked by init command */
    eEEP_SerialNumber      = 0x000E, /* Checked by init command */
} EC_T_eEEPENTRY;
```

4.15.5 ecatIoControl – EC_IOCTL_SB_BUSCNF_GETSLAVE_INFO

This call will return the basic slave info determined in the last bus scan. It may be called after retrieving the EC_NOTIFY_SB_STATUS notification.
Attention: The serial number and the revision number will not be read by default (0 will be returned). Please use the IO-Control EC_IOCTL_SB_SET_BUSCNF_VERIFY_PROP to activate the reading from the Slave EEPROM.

Parameters

pbyInBuf
[in] Pointer to auto-increment address of the slave (EC_T_WORD).
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[out] Pointer to slave description (EC_T_SB_SLAVEINFO_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.

Comment

EC_T_SB_SLAVEINFO_DESC

```
typedef struct _EC_T_SB_SLAVEINFO_DESC{
    EC_T_DWORD          dwScanBusStatus;
    EC_T_DWORD          dwVendorId;
    EC_T_DWORD          dwProductCode;
    EC_T_DWORD          dwRevisionNumber;
    EC_T_DWORD          dwSerialNumber;
} EC_T_SB_SLAVEINFO_DESC;
```

Description

dwScanBusStatus
[out] Scan bus status (determined in the latest scan bus)
See 4.15.12 “ecatNotify – EC_NOTIFY_SB_STATUS”.
dwVendorId
[out] Vendor Identification stored in the E²PROM at offset 0x0008
dwProductCode
[out] Product Code stored in the E²PROM at offset 0x000A
dwRevisionNumber
[out] Revision number stored in the E²PROM at offset 0x000C (Not read by default!)
dwSerialNumber
[out] Serial number stored in the E²PROM at offset 0x000E (Not read by default!)

4.15.6 ecatIoControl – EC_IOCTL_SB_BUSCNF_GETSLAVE_INFO_EEP

This call will return extended slave info determined in the last bus scan. It may be called after retrieving the EC_NOTIFY_SB_STATUS notification.

Parameters

pbyInBuf
[in] Pointer to EC_T_SB_SLAVEINFO_EEP_REQ_DESC.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[out] Pointer to slave description (EC_T_SB_SLAVEINFO_EEP_RES_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.

Comment

EC_T_SB_SLAVEINFO_EEP_REQ_DESC

```
typedef struct _EC_T_SB_SLAVEINFO_EEP_REQ_DESC{
    EC_T_eSBSlaveInfoType     eSbSlaveInfoType;
    EC_T_WORD                 wAutoIncAddress;
    EC_T_eEEPENTRY            eEEPROMEntry;
} EC_T_SB_SLAVEINFO_EEP_REQ_DESC;
```

Description

eSbSlaveInfoType
[in] Selection whether to use Auto-Increment address of Bus or XML Configuration
wAutoIncAddress
[in] Auto-Increment address of the slave
eEEPROMEntry
[in] E²PROM entry to read (only valid if entry was selected by ecatIoControl – EC_IOCTL_SB_SET_BUSCNF_VERIFY_PROP).

EC_T_eSBSlaveInfoType

```
typedef enum _EC_T_eSBSlaveInfoType{
    sbsit_unknown           = 0,
    sbsit_bustopology       = 1,        /* info from bus */
    sbsit_configuration     = 2,        /* info from XML configuration */
} EC_T_eSBSlaveInfoType;
```

Return

EC_E_NOERROR if successful.
EC_E_NOTFOUND if there is no scan bus result to the requested Auto-Increment address available.

EC_T_SB_SLAVEINFO_EEP_RES_DESC

```
typedef struct _EC_T_SB_SLAVEINFO_RES_DESC{
    EC_T_DWORD              dwScanBusStatus;
    EC_T_eEEPENTRY          eEEPROMEntry;
    EC_T_DWORD              dwEEPROMValue;
} EC_T_SB_SLAVEINFO_EEP_RES_DESC;
```

Description

dwScanBusStatus

[out] Scan bus status (determined in the latest scan bus)
See 4.15.12 “ecatNotify – EC_NOTIFY_SB_STATUS”.

eEEPROMEntry

[out] Select E²PROM Entry description from Request

dwEEPROMValue

[out] E²PROM entry value

4.15.7 ecatIoControl – EC_IOCTL_SB_BUSCNF_GETSLAVE_INFO_EX

This call will return extended slave info determined in the last bus scan. It may be called after retrieving the EC_NOTIFY_SB_STATUS notification.

Parameters

pbyInBuf
[in] Pointer to EC_T_SB_SLAVEINFO_REQ_DESC.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[out] Pointer to slave description (EC_T_SB_SLAVEINFO_RES_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.

Comment

```
typedef struct _EC_T_SB_SLAVEINFO_REQ_DESC
{
   EC_T_eINFOENTRY eInfoEntry;
   EC_T_WORD       wAutoIncAddress;
} EC_T_SB_SLAVEINFO_REQ_DESC;
```

Description

wAutoIncAddress
[in] Auto-Increment address of the slave
eInfoEntry
[in] Info Entry to read

EC_T_SB_SLAVEINFO_RES_DESC

```
typedef struct _EC_T_SB_SLAVEINFO_RES_DESC{
    EC_T_eINFOENTRY          eInfoEntry;
    EC_T_PBYTE               pbyInfo;
    EC_T_DWORD               dwInfoLength;
} EC_T_SB_SLAVEINFO_RES_DESC;
```

Description

eInfoEntry
[out] Info entry read
pbyInfo
[out] Pointer to Info (-1 if no info found in XML file)
dwInfoLength
[in, out] Length of Info Field (buffer, actually read length)

EC_T_eINFOENTRY

```
typedef enum _EC_T_eINFOENTRY{
    eie_unknown             /** nothing / invalid */
    eie_pdoffs_in           /** config: get processdata offset of Input data (in Bits) */
    eie_pdsize_in           /** config: get processdata size of Input data (in Bits) */
    eie_pdoffs_out          /** config: get processdata offset of Output data (in Bits) */
    eie_pdsize_out          /** config: get processdata size of Output data (in Bits) */
    eie_pdoffs_in2          /** config: get PD offset of Input data (section 2)(in Bits) */
    eie_pdsize_in2          /** config: get PD size of Input data (section 2)(in Bits) */
    eie_pdoffs_out2         /** config: get PD offset of Output data (section 2)(in Bits) */
    eie_pdsize_out2         /** config: get PD size of Output data (section 2)(in Bits) */
    eie_pdoffs_in3          /** config: get PD offset of Input data (section 3)(in Bits) */
    eie_pdsize_in3          /** config: get PD size of Input data (section 3)(in Bits) */
    eie_pdoffs_out3         /** config: get PD offset of Output data (section 3)(in Bits) */
    eie_pdsize_out3         /** config: get PD size of Output data (section 3)(in Bits) */
    eie_pdoffs_in4          /** config: get PD offset of Input data (section 4)(in Bits) */
    eie_pdsize_in4          /** config: get PD size of Input data (section 4)(in Bits) */
    eie_pdoffs_out4         /** config: get PD offset of Output data (section 4)(in Bits) */
    eie_pdsize_out4         /** config: get PD size of Output data (section 4)(in Bits) */
    eie_phys_address        /** bus: get slave phys Address */
    eie_portstate           /** bus: get port link state (DL_STATUS,
                                needed e.g. for topology detection) */
    eie_dcsupport           /** bus: does slave support DC */
    eie_dc64support         /** bus: does slave support 64 Bit DC */
    eie_alias_address       /** bus: get slave alias Address */
    eie_cfgphy_address      /** config: get slave phys Address from config file */
    eie_device_name         /** get slave name from configuration */
    eie_ismailbox_slave     /** get whether slave support mailboxes */
    eie_mbx_outsize         /** get out mailbox 1 size */
    eie_mbx_insize          /** get in mailbox 1 size */
    eie_mbx_outsize2        /** get out mailbox 2 size */
    eie_mbx_insize2         /** get in mailbox 2 size */
    eie_isoptional          /** is slave optional */
    eie_ispresent           /** is slave present on bus */
    eie_esctype             /** Type of ESC controller */
} EC_T_eINFOENTRY;
```

Brief description of individual request:

eInfoEntry	Type	Coding	Description
eie_pdoffs_in	EC_T_DWORD	Offset in Bit	Process data offset for Input data
eie_pdsize_in	EC_T_DWORD	Size in Bit	Process data size of Input data
eie_pdoffs_out	EC_T_DWORD	Offset in Bit	Process data offset for Output data
eie_pdsize_out	EC_T_DWORD	Size in Bit	Process data size of Output data
eie_pdoffs_in2	EC_T_DWORD	Offset in Bit	PD offset for Input data (section 2)
eie_pdsize_in2	EC_T_DWORD	Size in Bit	PD size of Input data (section 2)
eie_pdoffs_out2	EC_T_DWORD	Offset in Bit	PD offset for Output data (section 2)
eie_pdsize_out2	EC_T_DWORD	Size in Bit	PD size of Output data (section 2)
eie_pdoffs_in3	EC_T_DWORD	Offset in Bit	PD offset for Input data (section 3)
eie_pdsize_in3	EC_T_DWORD	Size in Bit	PD size of Input data (section 3)
eie_pdoffs_out3	EC_T_DWORD	Offset in Bit	PD offset for Output data (section 3)
eie_pdsize_out3	EC_T_DWORD	Size in Bit	PD size of Output data (section 3)
eie_pdoffs_in4	EC_T_DWORD	Offset in Bit	PD offset for Input data (section 4)
eie_pdsize_in4	EC_T_DWORD	Size in Bit	PD size of Input data (section 4)
eie_pdoffs_out4	EC_T_DWORD	Offset in Bit	PD offset for Output data (section 4)
eie_pdsize_out4	EC_T_DWORD	Size in Bit	PD size of Output data (section 4)
eie_phys_address	EC_T_WORD	Address	Slave Address (effective address)
eie_portstate	EC_T_WORD	wwww xxxx yyyy zzzz	Portstate: 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)
eie_dcsupport	EC_T_BOOL	BOOL	EC_TRUE if slave supports DC
eie_dc64support	EC_T_BOOL	BOOL	EC_TRUE if slave supports 64 Bit DC
eie_alias_address	EC_T_WORD	Address	Slave Alias Address
eie_cfgphy_address	EC_T_WORD	Address	Slave Station Address (Configured)
eie_device_name	EC_T_CHAR[80]	Name	Name of Device from XML Configuration
eie_ismailbox_slave	EC_T_BOOL	BOOL	EC_TRUE if slave supports mailboxes
eie_mbx_outsize	EC_T_DWORD	DWORD	Size of out mailbox (0 if no mailbox slave)
eie_mbx_outsize2	EC_T_DWORD	DWORD	Size of boot out mailbox (0 if no mailbox slave)
eie_mbx_insize	EC_T_DWORD	DWORD	Size of in mailbox (0 if no mailbox slave)
eie_mbx_insize2	EC_T_DWORD	DWORD	Size of boot in mailbox (0 if no mailbox slave)
eie_isoptional	EC_T_BOOL	BOOL	EC_TRUE if slave is optional
eie_ispresent	EC_T_BOOL	BOOL	EC_TRUE if slave is present on bus
eie_esctype	EC_T_BYTE	BYTE	Type of ESC controller 0x01: ESC10 0x02: ESC20 0x04: IP Core 0x11: ET1100 0x12: ET1200
eie_unknown	-	-	Error / unknown request id (shall not be used as input to IOCTL)

Example values for topology detection (values are binary):

zzzz = 0001 only port 0 (IN port) is connected, no slave behind this device (device is at the end of a bus line)

zzzz = 0011 ports 0 and 1 are connected, one slave behind this device (device is in the middle of a bus line)

zzzz = 0111 ports 0, 1 and 2 are connected, two slaves behind this device (Y-device, e.g. EK1100 bus coupler with E-Bus line connected behind and a second slave or bus line connected to the OUT port of the EK1100)

4.15.8 ecatIoControl – EC_IOCTL_SB_SET_TOPOLOGY_CHANGED_DELAY

This call will set the topology changed delay value. This is time the master will wait to react on the topology change in msec. The default value is 1s.

Parameters

pbyInBuf

[in] Pointer to EC_T_DWORD containing the delay information in msec

dwInBufSize

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

pbyOutBuf

[] Should be set to EC_NULL.

dwOutBufSize

[] Should be set to 0.

pdwNumOutData

[] Should be set to EC_NULL.

4.15.9 ecatIoControl – EC_IOCTL_SB_SET_ERROR_ON_CROSSED_LINES

This call will enable or disable bus mismatch if IN and OUT connectors are swapped. If enabled the swapped IN and OUT connectors will lead to bus mismatch.
By default swapped IN and OUT connectors will lead to bus mismatch.

Parameters

pbyInBuf

[in] Pointer to EC_T_BOOL variable. If set to EC_TRUE swapped IN and OUT connectors will lead to bus mismatch, if set to EC_FALSE swapped IN and OUT connectors are tolerated.

dwInBufSize

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

pbyOutBuf

[] Should be set to EC_NULL.

dwOutBufSize

[] Should be set to 0.

pdwNumOutData

[] Should be set to EC_NULL.

Comment

-

4.15.10 ecatIoControl – EC_IOCTL_SB_SET_TOPOLOGY_CHANGE_AUTO_MODE

[This documentation is preliminary and is subject to change]

This call will enable or disable the automatical topology change mode. By default the automatical mode is enabled.

Parameters

pbyInBuf
[in] Pointer to EC_T_BOOL variable. If set to EC_TRUE the automatical mode is enabled.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Should be set to EC_NULL.
dwOutBufSize
[] Should be set to 0.
pdwNumOutData
[] Should be set to EC_NULL.

Comment

In automatical mode, new slaves will be discovered automatically. In manual mode, after new slaves have been connected, a **EC_NOTIFY_HC_TOPOCHGDONE** notification will be given without opening the ports of the slaves on bus. When the application is able to handle the new slaves, it should call **EC_IOCTL_SB_ACCEPT_TOPOLOGY_CHANGE**

4.15.11 ecatIoControl – EC_IOCTL_SB_ACCEPT_TOPOLOGY_CHANGE

[This documentation is preliminary and is subject to change]

This call will trigger a scan bus. On completion the Notfication **EC_NOTIFY_SB_STATUS** is given.

Parameters

–

Comment

This function may be called after a **EC_NOTIFY_HC_TOPOCHGDONE** notification was given if the automatical topology change mode was previously disabled using **EC_IOCTL_SB_SET_TOPOLOGY_CHANGE_AUTO_MODE**.
During this scan bus the ports of the slaves will be (re)open and new slaves can be detected.
The timeout value given by **EC_IOCTL_SB_ENABLE** will be used. When issuing this IoControl, the application has to take care ecatExecJob is called cyclically to trigger master state machines, timers, send acyc and receive frames accordingly.

4.15.12 ecatNotify – EC_NOTIFY_SB_STATUS

Scan bus status notification.

Parameters

pbyInBuf
[in] Pointer to EC_T_SB_STATUS_NTFY_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

EC_T_SB_STATUS_NTFY_DESC

```
typedef struct _EC_T_SB_STATUS_NTFY_DESC{
    EC_T_DWORD          dwResultCode;
    EC_T_DWORD          dwSlaveCount;
} EC_T_SB_STATUS_NTFY_DESC;
```

Description

dwResultCode

[in] EC_E_NOERROR: success
EC_E_NOTREADY: no bus scan executed
EC_E_BUSCONFIG_MISMATCH: bus configuration mismatch

dwSlaveCount

[in] number of slaves connected to the bus

4.15.13 ecatNotify - EC_NOTIFY_SB_MISMATCH

Scan bus mismatch was detected while scan. This notification will be initiated if the scan bus has found a slave that don't matches to the expected slave in the configuration.

Parameters

pbyInBuf

[in] Pointer to EC_T_SB_MISMATCH_DESC

dwInBufSize

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

pbyOutBuf

[] Set to EC_NULL.

dwOutBufSize

[] Set to 0.

pdwNumOutData

[] Set to EC_NULL.

Comment

```
typedef struct _EC_T_SB_MISMATCH_DESC
{
   EC_T_WORD        wPrevFixedAddress;
   EC_T_WORD        wPrevPort;
   EC_T_WORD        wPrevAIncAddress;
   EC_T_WORD        wBusAIncAddress;
   EC_T_DWORD       dwBusVendorId;
   EC_T_DWORD       dwBusProdCode;
   EC_T_DWORD       dwBusRevisionNo;
   EC_T_DWORD       dwBusSerialNo;
   EC_T_WORD        wBusFixedAddress;
   EC_T_WORD        wIdentificationVal;
   EC_T_WORD        wCfgFixedAddress
   EC_T_WORD        wCfgAIncAddress;
   EC_T_DWORD       dwCfgVendorId;
   EC_T_DWORD       dwCfgProdCode;
   EC_T_DWORD       dwCfgRevisionNo;
   EC_T_DWORD       dwCfgSerialNo;
   EC_T_BOOL        bIdentValidationError;
   EC_T_WORD        oIdentCmdHdr[5]; */
   EC_T_DWORD       dwCmdData; */
   EC_T_DWORD       dwCmdVMask;
   EC_T_DWORD       dwCmdVData;

} EC_T_SB_MISMATCH_DESC;
```

Description

wPrevFixedAddress

[in] Previous slave station address

wPrevPort

[in] Previous slave port

wPrevAIncAddress
[in] Previous slave auto-increment address
wBusAIncAddress;
[in] Unexpected slave (bus) auto-inc address
dwBusVendorId;
[in] Unexpected slave (bus) vendor ID
dwBusProdCode;
[in] Unexpected slave (bus) product code
dwBusRevisionNo;
[in] Unexpected slave (bus) revision number
dwBusSerialNo;
[in] Unexpected slave (bus) serial number
wBusFixedAddress;
[in] Unexpected slave (bus) station address
wIdentificationVal;
[in] last identification value read from slave according to the last used identification method
wCfgFixedAddress
[in] Missing slave (config) station Address
wCfgAIncAddress;
[in] Missing slave (config) Auto-Increment Address.
dwCfgVendorId;
[in] Missing slave (config) Vendor ID
dwCfgProdCode;
[in] Missing slave (config) Product code
dwCfgRevisionNo;
[in] Missing slave (config) Revision Number
dwCfgSerialNo;
[in] Missing slave (config) Serial Number
bIdentValidationError;
[in] Hotconnect Identification command sent to slave but failed
oIdentCmdHdr[5];
[in] Last HotConnect Identification command header (if bIdentValidationError)
dwCmdData;
[in] First DWORD of Data portion of last identification command
dwCmdVMask;
[in] First DWORD of Validation mask of last identification command
dwCmdVData;
[in] First DWORD of Validation data of last identification command

4.15.14 ecatNotify - EC_NOTIFY_SB_DUPLICATE_HC_NODE

Scan bus mismatch was detected while scan because of a duplicate hot connect group.
An application get this notification if the there are two hot connect groups on the bus with the same product code, vendor ID and identification value (alias address or switch id).

Parameters

pbyInBuf
[in] Pointer to EC_T_SB_MISMATCH_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

See section 4.15.13 (EC_T_SB_MISMATCH_DESC)

4.15.15 ecatNotify - EC_NOTIFY_SLAVE_PRESENCE

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

Parameters

pbyInBuf
[in] Pointer to EC_T_SLAVE_PRESENCE_NTFY_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

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

```
typedef struct _EC_T_SLAVE_PRESENCE_NTFY_DESC
{
   EC_T_WORD        wStationAddress;
   EC_T_BOOL        bPresent;
} EC_T_SLAVE_PRESENCE_NTFY_DESC;
```

Description

wStationAddress
[in] Slave station address
bPresent
[in] 0: absent 1: present

4.15.16 ecatNotify - EC_NOTIFY_SLAVES_PRESENCE

This notification collects notifications of type EC_NOTIFY_SLAVE_PRESENCE.
Notification is given on either collection full or master state changed whatever comes first.
This notification is disabled by default. See EC_IOCTL_SET_NOTIFICATION_ENABLED for how to control the activation.

Parameters

pbyInBuf
[in] Pointer to EC_T_SLAVES_PRESENCE_NTFY_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

Comment

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

```
typedef struct _EC_T_SLAVES_PRESENCE_NTFY_DESC
{
   EC_T_WORD        wCount;
   EC_T_WORD        wRes;
   EC_T_SLAVE_PRESENCE_NTFY_DESC SlavePresence[MAX_SLAVES_PRESENCE_NTFY_ENTRIES];
} EC_T_SLAVES_PRESENCE_NTFY_DESC;
```

Description

wCount
[in] Contained EC_NOTIFY_SLAVE_PRESENCE count.
wRes
[in] 0 (reserved)
SlavePresence
[in] Array of EC_T_SLAVE_PRESENCE_NTFY_DESC, see EC_NOTIFY_SLAVE_PRESENCE.

4.15.17 ecatNotify - EC_NOTIFY_LINE_CROSSED

Cable swapping detected. Port 0 must be the port leading to the Master.

Parameters

pbyInBuf
[in] Pointer to EC_T_LINE_CROSSED_DESC
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

```
typedef struct _EC_T_LINE_CROSSED_DESC
{
   EC_T_SLAVE_PROP SlaveProp;      /*< slave properties */
   EC_T_WORD       wInputPort;     /*< port where frame was received */
} EC_T_LINE_CROSSED_DESC;
```

4.15.18 ecatNotify - EC_NOTIFY_JUNCTION_RED_CHANGE

Slave with not connected port 0 detected. Either cabling error or junction redundancy.

Parameters

pbyInBuf
[in] Pointer to EC_T_ERROR_NOTIFICATION_DESC containing EC_T_JUNCTION_RED_CHANGE_DESC.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

```
typedef struct _EC_T_JUNCTION_RED_CHANGE_DESC
{
   EC_T_SLAVE_PROP SlaveProp;        /*< slave properties of the slave with disconnected port 0 */
   EC_T_BOOL       bLineBreak;       /*< EC_TRUE for line break, EC_FALSE for line fixed */
} EC_T_JUNCTION_RED_CHANGE_DESC;
```

4.15.19 ecatNotify - EC_NOTIFY_SLAVE_NOTSUPPORTED

Is currently generated during Bus Scan if ecatConfigureMaster(GenOp/Preop) and a wrong category type is detected in the EEPROM. This notification should only print a log message or be ignored (Master print log message itself).

Parameters

pbyInBuf
[in] Pointer to EC_T_ERROR_NOTIFICATION_DESC containing EC_T_SLAVE_NOTSUPPORTED_DESC.
dwInBufSize
[in] Size of the input buffer provided at *pbyInBuf* in bytes.
pbyOutBuf
[] Set to EC_NULL.
dwOutBufSize
[] Set to 0.
pdwNumOutData
[] Set to EC_NULL.

```
typedef struct _EC_T_SLAVE_NOTSUPPORTED_DESC
{
   EC_T_SLAVE_PROP SlaveProp;        /*< slave properties of the not supported slave */
} EC_T_SLAVE_NOTSUPPORTED_DESC;
```

4.15.20 ecatNotify – Bus Scan notifications for Feature Packs

The notifications EC_NOTIFY_RED_LINEBRK, EC_NOTIFY_RED_LINEFIXED belong to the Feature Pack Redundancy.
The notifications EC_NOTIFY_HC_DETECTADDGROUPS, EC_NOTIFY_HC_PROBEALLGROUPS, EC_NOTIFY_HC_TOPOCHGDONE belong to the Feature Pack Hot Connect.

4.15.21 ecatIoControl – EC_IOCTL_SB_NOTIFY_UNEXPECTED_BUS_SLAVES

Specifies if unexpected bus slaves must be notified as bus mismatch.

Parameters

pbyInBuf

[in] Pointer to EC_T_BOOL variable. If set to EC_TRUE unexpected bus slaves on the network will be notified by **EC_NOTIFY_SB_MISMATCH**.

dwInBufSize

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

pbyOutBuf

[] Should be set to EC_NULL.

dwOutBufSize

[] Should be set to 0.

pdwNumOutData

[] Should be set to EC_NULL.

4.15.22 ecatIsTopologyChangeDetected

Returns whether topology change detected.

```
EC_T_DWORD ecatIsTopologyChangeDetected (
    EC_T_BOOL *                 pbTopologyChangeDetected
);
```

Parameters

pbTopologyChangeDetected

[out] Pointer to Bool value: EC_TRUE if Topology Change Detected, EC_FALSE if not.

Return

EC_E_NOERROR if successful.
EC_E_INVALIDSTATE if the master is not initialized.

Comment

-

5 Error Codes

5.1 Groups

Nr.	Group	Abbreviation	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

5.2 Codes

5.2.1 Generic Error Codes

Code / Define	Text	Group	Possible error cause
0x00000000 EC_E_NOERROR	No Error	n. a.	Function call successful.
0x98110001 EC_E_NOTSUPPORTED	Feature not supported	APP	Function or property not available
0x98110002 EC_E_INVALIDINDEX	Invalid Index	APP	CoE: invalid SDO index.
0x98110003 EC_E_INVALIDOFFSET	Invalid Offset	ISW	Invalid offset, while accessing Process Data Image
0x98110004 EC_E_CANCEL	Cancel	APP	master should abort current mbx transfer
0x98110005 EC_E_INVALIDSIZE	Invalid Size	APP	Invalid size - while accessing Process Data Image - while storing data
0x98110006 EC_E_INVALIDDATA	Invalid Data	ISW	Multiple error sources
0x98110007 EC_E_NOTREADY	Not ready	ISW	Multiple error sources
0x98110008 EC_E_BUSY	Busy	APP	Stack is busy currently and not available to process the API request. The function may be called again later.
0x98110009 EC_E_ACYC_FRM_FREEQ_EMPTY	Cannot queue acyclic ecat command	ISW	Acyclic command queue is full. Possible solution: Increase of configuration value *dwMaxQueuedEthFrames*
0x9811000A EC_E_NOMEMORY	No Memory left	CFG	Not enough allocatable memory available (memory full / corrupted).
0x9811000B EC_E_INVALIDPARM	Invalid Parameter	APP	API function called with erroneous parameter set.
0x9811000C EC_E_NOTFOUND	Not Found	APP	Network Information File not found or API called with invalid SlaveID.
0x9811000D EC_E_DUPLICATE	Duplicated fixed address detected	ISW	Internally handled.
0x9811000E EC_E_INVALIDSTATE	Invalid State	ISW	Multiple error sources

Code / Define	Text	Group	Possible error cause
0x9811000F EC_E_TIMER_LIST_FULL	Cannot add slave to timer list	ISW	Slave timer list full.
0x98110010 EC_E_TIMEOUT	Timeout		Multiple error sources.
0x98110011 EC_E_OPENFAILED	Open Failed	ISW	Multiple error sources.
0x98110012 EC_E_SENDFAILED	Send Failed	LLA	Transmit of frame failed.
0x98110013 EC_E_INSERTMAILBOX	Insert Mailbox error	CFG	Mailbox command couldn't be stored to internal command queue. Internal limit `MAX_QUEUED_COE_CMDS: 20.`
0x98110014 EC_E_INVALIDCMD	Invalid Command	ISW	Unknown mailbox command code.
0x98110015 EC_E_UNKNOWN_MBX_PROTOCOL	Unknown Mailbox Protocol Command	ISW	Unknown Mailbox protocol or mailbox command with unknown protocol association.
0x98110016 EC_E_ACCESSDENIED	Access Denied	ISW	Master internal software error:
0x9811001A EC_E_PRODKEY_INVALID	Product Key Invalid	CFG	Application is using protected version of stack, which stops operation after 60 minutes if license not provided.
0x9811001B EC_E_WRONG_FORMAT	Wrong configuration format	ENI	Network information file is empty or malformed.
0x9811001C EC_E_FEATURE_DISABLED	Feature disabled	APP	Application tried to perform a missing or disabled API function.
0x9811001E EC_E_BUSCONFIG_MISMATCH	Bus Config Mismatch	ENI	Network information file and currently connected bus topology does not match.
0x9811001F EC_E_CONFIGDATAREAD	Error reading config file	ENI	Network information file could not be read.
0x98110021 EC_E_XML_CYCCMDS_MISSING	Cyclic commands are missing	ENI	Network information file does not contain cyclic commands.
0x98110022 EC_E_XML_ALSTATUS_READ_MISSING	AL_STATUS register read missing in XML file for at least one state	ENI	Read of AL Status register is missing in cyclic part of given network information file.
0x98110023 EC_E_MCSM_FATAL_ERROR	Fatal internal McSm	ISW	Master control state machine is in an undefined state.
0x98110024 EC_E_SLAVE_ERROR	Slave error	SLV	A slave error was detected. See also EC_NOTIFY_STATUS_SLAVE_ERROR and EC_NOTIFY_SLAVE_ERROR_STATUS_INFO

Code / Define	Text	Group	Possible error cause
0x98110025 EC_E_FRAME_LOST	Frame lost, IDX mismatch	SLV	An EtherCAT frame was lost on bus segment, means the response was not received. In case this error shows frequently a problem with the wiring could be the cause.
0x98110026 EC_E_CMD_MISSING	At least one EtherCAT command is missing in the received frame	SLV	Received EtherCAT frame incomplete.
0x98110028 EC_E_INVALID_DCL_MODE	IOCTL EC_IOCTL_DC_LATCH_REQ_LTIMV ALS invalid in DCL auto read mode	APP	This function cannot be used if DC Latching is running in mode „Auto Read“.
0x98110029 EC_E_AI_ADDRESS	Auto increment address increment mismatch	SLV	Network information file and bus topology doesn’t match any more. Error shows only, if a already recognized slave isn’t present any more.
0x9811002A EC_E_INVALID_SLAVE_STATE	Slave in invalid state, e.g. not in OP (API not callable in this state)	APP	Mailbox commands are not allowed in current slave state.
0x9811002B EC_E_SLAVE_NOT_ADDRESSABLE	Station address lost (or slave missing) - FPRD to AL_STATUS failed	SLV	Slave had a powercycle.
0x9811002C EC_E_CYC_CMDS_OVERFLOW	Too many cyclic commands in XML configuration file	ENI	Error while creating network information file within configuration utility.
0x9811002D EC_E_LINK_DISCONNECTED	Ethernet link cable disconnected	SLV	EtherCAT bus segment not connected to network interface.
0x9811002E EC_E_MASTERCORE_INACCESSIBLE	Master core not accessible	RAS	Connection to remote server was terminated or master instance has been stopped on remote side.
0x9811002F EC_E_COE_MBXSND_WKC_ERROR	COE mbox send: working counter	SLV	CoE mailbox couldn’t be read on slave, slave didn’t read out mailbox since last write.
0x98110030 EC_E_COE_MBXRCV_WKC_ERROR	COE mbox receive: working counter	SLV	CoE Mailbox couldn’t be read from slave.
0x98110031 EC_E_NO_MBX_SUPPORT	No mailbox support	APP	Slave does not support mailbox access.
0x98110032 EC_E_NO_COE_SUPPORT	CoE protocol not supported	ENI	Configuration error or slave information file doesn’t match slave firmware.
0x98110033 EC_E_NO_EOE_SUPPORT	EoE protocol not supported	ENI	Configuration error or slave information file doesn’t match slave firmware.
0x98110034 EC_E_NO_FOE_SUPPORT	FoE protocol not supported	ENI	Configuration error or slave information file doesn’t match slave firmware.
0x98110035 EC_E_NO_SOE_SUPPORT	SoE protocol not supported	ENI	Configuration error or slave information file doesn’t match slave firmware.
0x98110036	VoE protocol not supported	ENI	Configuration error or slave information file doesn’t match

Code / Define	Text	Group	Possible error cause
EC_E_NO_VOE_SUPPORT			slave firmware.
0x98110037 EC_E_EVAL_VIOLATION	Configuration violates Evaluation limits	ENI	***Obsolete***
0x98110038 EC_E_EVAL_EXPIRED	Evaluation Time limit reached	CFG	Licence not provided and evaluation period (1 hour) of protected version exceeded.
0x98110070 EC_E_CFGFILENOTFOUND	Master configuration not found	CFG	The path to the master configuration file (XML) was wrong or the file is not available.
0x98110071 EC_E_EEPROMREADERROR	Command error while EEPROM upload	SLV	Could not read from slave EEPROM.
0x98110072 EC_E_EEPROMWRITEERROR	Command error while EEPROM download	SLV	Could not write to slave EEPROM.
0x98110073 EC_E_XML_CYCCMDS_SIZEMISMATCH	Cyclic command wrong size (too long)	ENI	Error while creating a new cyclic command. The size which was defined in the master configuration xml does not match to the size of the process data.
0x98110075 EC_E_XML_INVALID_OUT_OFF	Invalid output offset in cyc cmd, please check OutputOffs	ENI	***Obsolete***
0x98110076 EC_E_PORTCLOSE	Port Close failed		
0x98110077 EC_E_PORTOPEN	Port Open failed		
0x9811010e EC_E_SLAVE_NOT_PRESENT	Command not executed (slave not present on bus)	APP / SLV	Slave disappeared or was never present.
0x98110110 EC_E_EEPROMRELOADERROR	Command error while EEPROM reload		
0x98110111 EC_E_SLAVECTRLRESETERROR	Command error while Reset Slave Controller		
0x98110112 EC_E_SYSDRIVERMISSING	Cannot open system driver	SYS	System driver was not loaded.
0x9811011E EC_E_BUSCONFIG_TOPOCHANGE	Bus configuration not detected, Topology changed		Topology changed while scanning bus
0x98110123 EC_E_VOE_MBX_WKC_ERROR	VoE mailbox send: working counter	SLV	VoE mailbox couldn’t be written.
0x98110124 EC_E_EEPROMASSIGNERROR	EEPROM assignment failed	SLV	Assignment of the EEPROM to the slave went wrong.
0x98110125	Error mailbox received	SLV	Unknown mailbox error code received in mailbox

Code / Define	Text	Group	Possible error cause
EC_E_MBX_ERROR_TYPE			
0x98110126 EC_E_REDLINEBREAK	Redundancy line break	SLV	Cable break between slaves or between master and first slave
0x98110127 EC_E_XML_INVALID_CMD_WITH_RED	Invalid EtherCAT cmd in cyclic frame with redundancy	ENI	BRW commands are not allowed with redundancy. LRW commands with an expected WKC>3 are not allowed with redundancy (Workaround: Use LRD/LWR instead of LRW)
0x98110128 EC_E_XML_PREV_PORT_MISSING	<PreviousPort>-tag is missing	ENI	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
0x98110129 EC_E_XML_DC_CYCCMDS_MISSING	DC is enabled and DC cyclic commands are missing (e.g. access to 0x900)	ENI	Error in Configuration Tool.
0x98110130 EC_E_DLSTATUS_IRQ_TOPOCHANGED	Data link (DL) status interrupt because of changed topology	SLV	Handled inside the master
0x98110131 EC_E_PTS_IS_NOT_RUNNING	Pass Through Server is not running	PTS	The Pass-Through-Server was tried to be enabled/disabled or stopped without being started.
0x98110132 EC_E_PTS_IS_RUNNING	Pass Through Server is running	PTS	***Obsolete***. Replaced by EC_E_ADS_IS_RUNNING
0x98110132 EC_E_ADS_IS_RUNNING	ADS adapter (Pass Through Server) is running	PTS	API call conflicts with ADS state (running).
0x98110133 EC_E_PTS_THREAD_CREATE_FAILED	Could not start the Pass Through Server	PTS	The Pass-Through-Server could not be started.
0x98110134 EC_E_PTS_SOCK_BIND_FAILED	The Pass Through Server could not bind the IP address with a socket	PTS	Possibly because the IPaddress (and Port) is already in use or the IP-address does not exist.
0x98110135 EC_E_PTS_NOT_ENABLED	The Pass Through Server is running but not enabled	PTS	-
0x98110136 EC_E_PTS_LL_MODE_NOT_SUPPORTED	The Link Layer mode is not supported by the Pass Through Server	PTS	The Master is running in interrupt mode but the Pass-Through-Server only supports polling mode.
0x98110137 EC_E_VOE_NO_MBX_RECEIVED	No VoE mailbox received	SLV	The master has not yet received a VoE mailbox from a specific slave.
0x98110138 EC_E_DC_REF_CLOCK_SYNC_OUT_UNIT_DISABLED	SYNC out unit of reference clock is disabled	ENI	Slave is selected as Reference clock in ENI file, but slave doesn’t have a SYNC unit. Possible a ESI file bug.

Code / Define	Text	Group	Possible error cause
0x98110139 EC_E_DC_REF_CLOCK_NOT_FOUND	Reference clock not found!	SLV	May happen if reference clock is removed from network.
0x9811013B EC_E_MBX_CMD_WKC_ERROR	Mailbox command working counter error	SLV	Mbx Init Cmd Retry Count exceeded.
0x9811013C EC_E_NO_AOE_SUPPORT	AoE: Protocol not supported	APP / SLV	Application calls AoE-API although not implemented at slave.
0x9811016E EC_E_XML_AOE_NETID_INVALID	AoE: Invalid NetID	ENI	Error from Configuration Tool.
0x9811016F EC_E_MAX_BUS_SLAVES_EXCEEDED	Error: Maximum number of bus slave has been exceeded	CFG	The maximum number of pre-allocated bus slave objects are to small. The maximum number can be adjusted by the master initialization parameter EC_T_INITMASTERPARMS.dwMaxBusSlaves.
0x98110170 EC_E_MBXERR_SYNTAX	Mailbox error: Syntax of 6 octet Mailbox header is wrong	SLV	Slave error mailbox return value: 0x01
0x98110171 EC_E_MBXERR_UNSUPPORTEDPROTOCOL	Mailbox error: The Mailbox protocol is not supported	SLV	Slave error mailbox return value: 0x02
0x98110172 EC_E_MBXERR_INVALIDCHANNEL	Mailbox error: Field contains wrong value	SLV	Slave error mailbox return value: 0x03
0x98110173 EC_E_MBXERR_SERVICENOTSUPPORTED	Mailbox error: The mailbox protocol header of the mailbox protocol is wrong	SLV	Slave error mailbox return value: 0x04
0x98110174 EC_E_MBXERR_INVALIDHEADER	Mailbox error: The mailbox protocol header of the mailbox protocol is wrong	SLV	Slave error mailbox return value: 0x05
0x98110175 EC_E_MBXERR_SIZETOOSHORT	Mailbox error: Length of received mailbox data is too short	SLV	Slave error mailbox return value: 0x06
0x98110176 EC_E_MBXERR_NOMOREMEMORY	Mailbox error: Mailbox protocol can not be processed because of limited ressources	SLV	Slave error mailbox return value: 0x07
0x98110177 EC_E_MBXERR_INVALIDSIZE	Mailbox error: The length of data is inconsistent	SLV	Slave error mailbox return value: 0x08
0x98110178 EC_E_DC_SLAVES_BEFORE_REF_CLOCK	Slaves with DC configured present on bus before reference clock	ENI	The first DC Slave was not configured as potential reference clock.
0x9811017B EC_E_LINE_CROSSED	Line crossed		Cabling wrong.

Code / Define	Text	Group	Possible error cause
0x9811017C EC_E_LINE_CROSSED_SLAVE_INFO	Line crossed at slave …		***Obsolete***
0x98130008 EC_E_OEM_SIGNATURE_MISMATCH	Manufacturer signature mismatch	ENI, OEM	
0x98130009 EC_E_ENI_ENCRYPTION_WRONG_VERSION	ENI encryption algorithm version not supported	ENI, OEM	
0x9813000A EC_E_ENI_ENCRYPTED	Loading encrypted ENI needs OEM key	OEM	
0x9813000B EC_E_OEM_KEY_MISMATCH	OEM key mismatch	RAS, OEM	

5.2.2 CANOpen over EtherCAT (CoE) SDO Error Codes

Code / Define	Text	Group	Possible error cause
0x98110040 EC_E_SDO_ABORTCODE_TOGGLE	SDO: Toggle bit not alternated	SLV	CoE abort code 0x05030000 of slave
0x98110041 EC_E_SDO_ABORTCODE_TIMEOUT	SDO: Protocol timed out	SLV	CoE abort code 0x05040000 of slave
0x98110042 EC_E_SDO_ABORTCODE_CCS_SCS	SDO: Client/server command specifier not valid or unknown	SLV	CoE abort code 0x05040001 of slave
0x98110043 EC_E_SDO_ABORTCODE_BLK_SIZE	SDO: Invalid block size (block mode only)	SLV	CoE abort code 0x05040002 of slave
0x98110044 EC_E_SDO_ABORTCODE_SEQNO	SDO: Invalid sequence number (block mode only)	SLV	CoE abort code 0x05040003 of slave
0x98110045 EC_E_SDO_ABORTCODE_CRC	SDO: CRC error (block mode only)	SLV	CoE abort code 0x05040004 of slave
0x98110046 EC_E_SDO_ABORTCODE_MEMORY	SDO: Out of memory	SLV	CoE abort code 0x05040005 of slave
0x98110047 EC_E_SDO_ABORTCODE_ACCESS	SDO: Unsupported access to an object	SLV	CoE abort code 0x06010000 of slave
0x98110048 EC_E_SDO_ABORTCODE_WRITEONLY	SDO: Attempt to read a write only object	SLV	CoE abort code 0x06010001 of slave
0x98110049 EC_E_SDO_ABORTCODE_READONL	SDO: Attempt to write a read only object	SLV	CoE abort code 0x06010002 of slave

Code / Define	Text	Group	Possible error cause
Y			
0x98130004 EC_E_SDO_ABORTCODE_SI_NOT_WRITTEN	SDO: Sub Index cannot be written, SI0 must be 0 for write access	SLV	CoE abort code 0x06010003 of slave
0x98130005 EC_E_SDO_ABORTCODE_CA_TYPE_MISM	SDO: Complete access not supported for objects of cvariable length suach as ENUM object types	SLV	CoE abort code 0x06010004 of slave
0x98130006 EC_E_SDO_ABORTCODE_OBJ_TOO_BIG	SDO: Object length exceeds mailbox size	SLV	CoE abort code 0x06010005 of slave
0x98130007 EC_E_SDO_ABORTCODE_PDO_MAPPED	SDO: Object mapped to RxPDO, SDO Download blocked	SLV	CoE abort code 0x06010006 of slave
0x9811004A EC_E_SDO_ABORTCODE_INDEX	SDO: Object does not exist in the object dictionary	SLV	CoE abort code 0x06020000 of slave
0x9811004B EC_E_SDO_ABORTCODE_PDO_MAP	SDO: Object cannot be mapped to the PDO	SLV	CoE abort code 0x06040041 of slave
0x9811004C EC_E_SDO_ABORTCODE_PDO_LEN	SDO: The number and length of the objects to be mapped would exceed PDO length	SLV	CoE abort code 0x06040042 of slave
0x9811004D EC_E_SDO_ABORTCODE_P_INCOMP	SDO: General parameter incompatibility reason	SLV	CoE abort code 0x06040043 of slave
0x9811004E EC_E_SDO_ABORTCODE_I_INCOMP	SDO: General internal incompatibility in the device	SLV	CoE abort code 0x06040047 of slave
0x9811004F EC_E_SDO_ABORTCODE_HARDWARE	SDO: Access failed due to an hardware error	SLV	CoE abort code 0x06060000 of slave
0x98110050 EC_E_SDO_ABORTCODE_DATA_SIZE	SDO: Data type does not match, length of service parameter does not match	SLV	CoE abort code 0x06070010 of slave
0x98110051 EC_E_SDO_ABORTCODE_DATA_SIZE1	SDO: Data type does not match, length of service parameter too high	SLV	CoE abort code 0x06070012 of slave
0x98110052 EC_E_SDO_ABORTCODE_DATA_SIZ	SDO: Data type does not match, length of service parameter too low	SLV	CoE abort code 0x06070013 of slave

Code / Define	Text	Group	Possible error cause
E2			
0x98110053 EC_E_SDO_ABORTCODE_OFFSET	SDO: Sub-index does not exist	SLV	CoE abort code 0x06090011 of slave
0x98110054 EC_E_SDO_ABORTCODE_DATA_RANGE	SDO: Value range of parameter exceeded (only for write access)	SLV	CoE abort code 0x06090030 of slave
0x98110055 EC_E_SDO_ABORTCODE_DATA_RANGE1	SDO: Value of parameter written too high	SLV	CoE abort code 0x06090031 of slave
0x98110056 EC_E_SDO_ABORTCODE_DATA_RANGE2	SDO: Value of parameter written too low	SLV	CoE abort code 0x06090032 of slave
0x98110057 EC_E_SDO_ABORTCODE_MINMAX	SDO: Maximum value is less than minimum value	SLV	CoE abort code 0x06090036 of slave
0x98110058 EC_E_SDO_ABORTCODE_GENERAL	SDO: General error	SLV	CoE abort code 0x08000000 of slave
0x98110059 EC_E_SDO_ABORTCODE_TRANSFER	SDO: Data cannot be transferred or stored to the application	SLV	CoE abort code 0x08000020 of slave
0x9811005A EC_E_SDO_ABORTCODE_TRANSFER1	SDO: Data cannot be transferred or stored to the application because of local control	SLV	CoE abort code 0x08000021 of slave
0x9811005B EC_E_SDO_ABORTCODE_TRANSFER2	SDO: Data cannot be transferred or stored to the application because of the present device state	SLV	CoE abort code 0x08000022 of slave
0x9811005C EC_E_SDO_ABORTCODE_DICTIONARY	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)	SLV	CoE abort code 0x08000023 of slave
0x9811005D EC_E_SDO_ABORTCODE_UNKNOWN	SDO: Unknown code	SLV	Unknown CoE abort code of slave

5.2.3 File Transfer over EtherCAT (FoE) Error Codes

Code / Define	Text	Group	Possible error cause
0x98110060 EC_E_FOE_ERRCODE_NOTDEFINED	ERROR FoE: not defined	SLV	FoE Error Code 0 (0x8000) of slave
0x98110061 EC_E_FOE_ERRCODE_NOTFOUND	ERROR FoE: not found	SLV	FoE Error Code 1 (0x8001) of slave
0x98110062 EC_E_FOE_ERRCODE_ACCESS	ERROR FoE: access denied	SLV	FoE Error Code 2 (0x8002) of slave
0x98110063 EC_E_FOE_ERRCODE_DISKFULL	ERROR FoE: disk full	SLV	FoE Error Code 3 (0x8003) of slave
0x98110064 EC_E_FOE_ERRCODE_ILLEGAL	ERROR FoE: illegal	SLV	FoE Error Code 4 (0x8004) of slave
0x98110065 EC_E_FOE_ERRCODE_PACKENO	ERROR FoE: packet number wrong	SLV	FoE Error Code 5 (0x8005) of slave
0x98110066 EC_E_FOE_ERRCODE_EXISTS	ERROR FoE: already exists	SLV	FoE Error Code 6 (0x8006) of slave
0x98110067 EC_E_FOE_ERRCODE_NOUSER	ERROR FoE: no user	SLV	FoE Error Code 7 (0x8007) of slave
0x98110068 EC_E_FOE_ERRCODE_BOOTSTRAPONLY	ERROR FoE: bootstrap only	SLV	FoE Error Code 8 (0x8008) of slave
0x98110069 EC_E_FOE_ERRCODE_NOTINBOOTSTRAP	ERROR FoE: Downloaded file name is not valid in Bootstrap state	SLV	FoE Error Code 9 (0x8009) of slave
0x9811006A EC_E_FOE_ERRCODE_INVALIDPASSWORD	ERROR FoE: no rights	SLV	FoE Error Code 10 (0x800A) of slave
0x9811006B EC_E_FOE_ERRCODE_PROGERROR	ERROR FoE: program error	SLV	FoE Error Code 11 (0x800B) of slave
0x9811006C EC_E_FOE_ERRCODE_INVALID_CHECKSUM	ERROR FoE: Wrong checksum	SLV	FoE Error Code 12 (0x800C) of slave
0x9811006D EC_E_FOE_ERRCODE_INVALID_FIRMWARE	ERROR FoE: Firmware does not fit for Hardware	SLV	FoE Error Code 13 (0x800D) of slave

Code / Define	Text	Group	Possible error cause
			FoE Error Code 14 (0x800E) reserved
0x9811006F EC_E_FOE_ERRCODE_NO_FILE	ERROR FoE: No file to read	SLV	FoE Error Code 15 (0x800F) of slave
0x98130001 EC_E_FOE_ERRCODE_FILE_HEAD_MISSING	ERROR FoE: File header does not exist	SLV	FoE Error Code 16 (0x8010) of slave
0x98130002 EC_E_FOE_ERRCODE_FLASH_PROBLEM	ERROR FoE: Flash problem	SLV	FoE Error Code 17 (0x8011) of slave
0x98130003 EC_E_FOE_ERRCODE_FILE_INCOMPATIBLE	ERROR FoE: File incompatible	SLV	FoE Error Code 18 (0x8012) of slave
0x9811010F EC_E_NO_FOE_SUPPORT_BS	ERROR FoE: Protocol not supported in boot strap	APP	Application requested FoE in Bootstrap although slave does not support this.
0x9811017A EC_E_FOE_ERRCODE_MAX_FILE_SIZE	ERROR FoE: File is bigger than max file size	APP	Slave returned more data than the buffer provided by application can store.

5.2.4 Servo Drive Profil over EtherCAT (SoE) Error Codes

Code / Define	Text	Group	Possible error cause
0x98110078 EC_E_SOE_ERRORCODE_INVALID_ACCESS	ERROR SoE: Invalid access to element 0		
0x98110079 EC_E_SOE_ERRORCODE_NOT_EXIST	ERROR SoE: Does not exist		
0x9811007A EC_E_SOE_ERRORCODE_INVL_ACC_ELEM1	ERROR SoE: Invalid access to element 1		
0x9811007B EC_E_SOE_ERRORCODE_NAME_NOT_EXIST	ERROR SoE: Name does not exist		
0x9811007C EC_E_SOE_ERRORCODE_NAME_UNDERSIZE	ERROR SoE: Name undersize in transmission		
0x9811007D EC_E_SOE_ERRORCODE_NAME_OVERSIZE	ERROR SoE: Name oversize in transmission		
0x9811007E EC_E_SOE_ERRORCODE_NAME_UNCHANGE	ERROR SoE: Name unchangeable		
0x9811007F EC_E_SOE_ERRORCODE_NAME_WR_PROT	ERROR SoE: Name currently write-protected		
0x98110080 EC_E_SOE_ERRORCODE_UNDERS_TRANS	ERROR SoE: Attribute undersize in transmission		
0x98110081 EC_E_SOE_ERRORCODE_OVERS_TRANS	ERROR SoE: Attribute oversize in transmission		
0x98110082 EC_E_SOE_ERRORCODE_ATTR_UNCHANGE	ERROR SoE: Attribute unchangeable		

Code / Define	Text	Group	Possible error cause
0x98110083 EC_E_SOE_ERRORCODE_ATTR_WR_PROT	ERROR SoE: Attribute currently write-protected		
0x98110084 EC_E_SOE_ERRORCODE_UNIT_NOT_EXIST	ERROR SoE: Unit does not exist		
0x98110085 EC_E_SOE_ERRORCODE_UNIT_UNDERSIZE	ERROR SoE: Unit undersize in transmission		
0x98110086 EC_E_SOE_ERRORCODE_UNIT_OVERSIZE	ERROR SoE: Unit oversize in transmission		
0x98110087 EC_E_SOE_ERRORCODE_UNIT_UNCHANGE	ERROR SoE: Unit unchangeable		
0x98110088 EC_E_SOE_ERRORCODE_UNIT_WR_PROT	ERROR SoE: Unit currently write-protected		
0x98110089 EC_E_SOE_ERRORCODE_MIN_NOT_EXIST	ERROR SoE: Minimum input value does not exist		
0x9811008A EC_E_SOE_ERRORCODE_MIN_UNDERSIZE	ERROR SoE: Minimum input value undersize in transmission		
0x9811008B EC_E_SOE_ERRORCODE_MIN_OVERSIZE	ERROR SoE: Minimum input value oversize in transmission		
0x9811008C EC_E_SOE_ERRORCODE_MIN_UNCHANGE	ERROR SoE: Minimum input value unchangeable		
0x9811008D EC_E_SOE_ERRORCODE_MIN_WR_PROT	ERROR SoE: Minimum input value currently write-protected		
0x9811008E EC_E_SOE_ERRORCODE_MAX_NOT_EXIST	ERROR SoE: Maximum input value does not exist		

Code / Define	Text	Group	Possible error cause
0x9811008F EC_E_SOE_ERRORCODE_MAX_UNDERSIZE	ERROR SoE: Maximum input value undersize in transmission		
0x98110090 EC_E_SOE_ERRORCODE_MAX_OVERSIZE	ERROR SoE: Maximum input value oversize in transmission		
0x98110091 EC_E_SOE_ERRORCODE_MAX_UNCHANGE	ERROR SoE: Maximum input value unchangeable		
0x98110092 EC_E_SOE_ERRORCODE_MAX_WR_PROT	ERROR SoE: Maximum input value currently write-protected		
0x98110093 EC_E_SOE_ERRORCODE_DATA_NOT_EXIST	ERROR SoE: Data item does not exist		
0x98110094 EC_E_SOE_ERRORCODE_DATA_UNDERSIZE	ERROR SoE: Data item undersize in transmission		
0x98110095 EC_E_SOE_ERRORCODE_DATA_OVERSIZE	ERROR SoE: Data item oversize in transmission		
0x98110096 EC_E_SOE_ERRORCODE_DATA_UNCHANGE	ERROR SoE: Data item unchangeable		
0x98110097 EC_E_SOE_ERRORCODE_DATA_WR_PROT	ERROR SoE: Data item currently write-protected		
0x98110098 EC_E_SOE_ERRORCODE_DATA_MIN_LIMIT	ERROR SoE: Data item less than minimum input value limit		
0x98110099 EC_E_SOE_ERRORCODE_DATA_MAX_LIMIT	ERROR SoE: Data item exceeds maximum input value limit		
0x9811009A EC_E_SOE_ERRORCODE_DATA_INCOR	ERROR SoE: Data item incorrect		

Code / Define	Text	Group	Possible error cause
0x9811009B EC_E_SOE_ERRORCODE_PASWD_PROT	ERROR SoE: Data item protected by password		
0x9811009C EC_E_SOE_ERRORCODE_TEMP_UNCHANGE	ERROR SoE: Data item temporary unchangeable (in AT or MDT)		
0x9811009D EC_E_SOE_ERRORCODE_INVL_INDIRECT	ERROR SoE: Invalid indirect		
0x9811009E EC_E_SOE_ERRORCODE_TEMP_UNCHANGE1	ERROR SoE: Data item temporary unchangeable (parameter or opmode)		
0x9811009F EC_E_SOE_ERRORCODE_ALREADY_ACTIVE	ERROR SoE: Command already active		
0x98110100 EC_E_SOE_ERRORCODE_NOT_INTERRUPT	ERROR SoE: Command not interruptable		
0x98110101 EC_E_SOE_ERRORCODE_CMD_NOT_AVAIL	ERROR SoE: Command not available (in this phase)		
0x98110102 EC_E_SOE_ERRORCODE_CMD_NOT_AVAIL1	ERROR SoE: Command not available (invalid parameter)		
0x98110103 EC_E_SOE_ERRORCODE_DRIVE_NO	ERROR SoE: Response drive number not identical with requested drive number		
0x98110104 EC_E_SOE_ERRORCODE_IDN	ERROR SoE: Response IDN not identical with requested IDN		
0x98110105 EC_E_SOE_ERRORCODE_FRAGMENT_LOST	ERROR SoE: At least one fragment lost		
0x98110106 EC_E_SOE_ERRORCODE_BUFFER_FULL	ERROR SoE: RX buffer full (ecat call with to small data-buffer)		

Code / Define	Text	Group	Possible error cause
0x98110107 EC_E_SOE_ERRORCODE_NO_DATA	ERROR SoE: No data state		
0x98110108 EC_E_SOE_ERRORCODE_NO_DEFAULT_VALUE	ERROR SoE: No default value		
0x98110109 EC_E_SOE_ERRORCODE_DEFAULT_LONG	ERROR SoE: Default value transmission too long		
0x9811010A EC_E_SOE_ERRORCODE_DEFAULT_WP	ERROR SoE: Default value cannot be changed, read only		
0x9811010B EC_E_SOE_ERRORCODE_INVL_DRIVE_NO	ERROR SoE: Invalid drive number		
0x9811010C EC_E_SOE_ERRORCODE_GENERAL_ERROR	ERROR SoE: General error		
0x9811010D EC_E_SOE_ERRCODE_NO_ELEM_ADR	ERROR SoE: No element addressed		

5.2.5 Remote API Error Codes

Code / Define	Text	Group	Possible error cause
0x98110181 EMRAS_E_INVALIDCOOKIE	Invalid Cookie	RAS	Reconnect with old connection cookie failed. Reconnect is performed implicit with new session cookie. Client registrations a mailbox objects have to be recreated.
0x98110182 EMRAS_E_WDEXPIRED	Watchdog expired	RAS	Connection dropped because of missing keep-alive messages. Either remote API client or server doesn't respond anymore:
0x98110183 EMRAS_E_MULSRVDISMULCON	Connect 2nd server denied because Multi Server support is disabled	RAS	Connection attempt to additional remote API server rejected because an existing connection was established without using multi connection API.
0x98110184 EMRAS_E_LOGONCANCELLED	Logon canceled	RAS	Serverside connection reject while opening a client connection.
0x98110186 EMRAS_E_INVALIDVERSION	Invalid Version	RAS	Connection reject because of using mismatching protocol versions on client and server side.
0x98110187 EMRAS_E_INVALIDACCESSCONFIG	Access configuration is invalid	RAS	The SPoC access configuration is invalid.
0x98110188 EMRAS_E_ACCESSLESS	No access to this call at this accesslevel	RAS	A higher SPoC accesslevel is needed to use the called Remote API function.
0x98110191 EMRAS_EVT_SERVERSTOPPED	Server stopped	RAS	Closer description if connection dropped because of Remote API Server stop using local API call to stop Remote API server.
0x98110192 EMRAS_EVT_WDEXPIRED	Watchdog expired	RAS	Closer description on server side when connection is dropped because of missing keep-alive messages. Session is still reconnectable.
0x98110193 EMRAS_EVT_RECONEXPIRED	Reconnect expired	RAS	Client tries to reconnect old session after disconnect and server has cleaned session already. A reconnect requires to re-register client thread registrations and created mailbox objects have to be re-created.
0x98110194 EMRAS_EVT_CLIENTLOGON	Client logged on	RAS	Serverside notification if a new client establishes a connection.
0x98110195 EMRAS_EVT_RECONNECT	Client reconnect	RAS	Serverside notification if a client tries to reconnect to an existing session successfully.
0x98110196 EMRAS_EVT_SOCKCHANGE	Socket exchanged after Reconnect	RAS	Closer description on a connection change, if newly spawned client socket is transferred to an existing session on a reconnection attempt (only serverside)

Code / Define	Text	Group	Possible error cause
0x98110197 EMRAS_EVT_CLNTDISC	Client disconnect	RAS	The remote client was disconnected from server.
0x9811017D EC_E_SOCKET_DISCONNECTED	Socket disconnected	RAS	IP connection died.