



**acontis technologies GmbH**

## **SOFTWARE**

# **EC-Master**

**EtherCAT® Master Stack Class B**

**Version 3.2**

**Edition: September 24, 2025**

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

© Copyright **acontis technologies GmbH**

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

# Contents

<b>1</b>	<b>Introduction</b>	<b>13</b>
1.1	What is EtherCAT? . . . . .	13
1.2	The EC-Master - Features . . . . .	13
1.3	Protected version . . . . .	15
1.3.1	Licensing procedure for Development Licenses . . . . .	15
1.3.2	Licensing procedure for Runtime Licenses . . . . .	15
1.4	License . . . . .	16
1.4.1	EC-Master license . . . . .	16
1.4.2	Free Open Source Software contained in EC-Master . . . . .	16
1.4.3	Free Open Source Software supported by EC-Master . . . . .	17
1.5	Versioning . . . . .	17
<b>2</b>	<b>Getting Started</b>	<b>18</b>
2.1	EC-Master Architecture . . . . .	18
2.2	EtherCAT Network Configuration (ENI) . . . . .	18
2.3	Operating system configuration . . . . .	19
2.4	Running EcMasterDemo . . . . .	19
2.4.1	Command line parameters . . . . .	19
2.5	Compiling the EcMasterDemo . . . . .	30
2.5.1	EtherCAT Master Software Development Kit (SDK) . . . . .	30
2.5.2	Include search path . . . . .	31
2.5.3	Libraries . . . . .	31
<b>3</b>	<b>Software Integration</b>	<b>32</b>
3.1	Network Timing . . . . .	32
3.1.1	Standard Timing: Short output dead time . . . . .	34
3.1.2	Alternative Timing: Short Input dead time . . . . .	36
3.2	Example application . . . . .	38
3.2.1	File reference . . . . .	39
3.2.2	Master lifecycle . . . . .	41
3.2.3	Synchronization . . . . .	44
3.2.4	Event notification . . . . .	45
3.2.5	Logging . . . . .	47
3.3	Master startup . . . . .	47
3.3.1	Asynchronous (deferred) startup . . . . .	48
3.3.2	Synchronous startup . . . . .	49
3.4	EtherCAT Network Configuration ENI . . . . .	51
3.4.1	Single cyclic entry configuration . . . . .	51
3.4.2	Multiple cyclic entries configuration . . . . .	53
3.4.3	Copy Information for Slave-to-Slave communication . . . . .	54
3.4.4	Swap bytes of variables according to ENI . . . . .	55
3.5	Process Data Access . . . . .	57
3.5.1	Process data access using hard coded offsets . . . . .	59
3.5.2	Process data access using variable names from ENI . . . . .	60
3.5.3	Process data access using variable object index from ENI . . . . .	61
3.5.4	Process data access using slave station address . . . . .	63
3.5.5	Process data access using generated PD layout C-header file . . . . .	64
3.5.6	Process Data Access Functions . . . . .	67
3.6	Process Data Memory . . . . .	68
3.6.1	EC-Master as process data memory provider . . . . .	68
3.6.2	Application as process data memory provider with fixed buffers . . . . .	70
3.6.3	Application as process data memory provider with dynamic buffers . . . . .	72
3.7	Error detection and diagnosis . . . . .	74
3.7.1	Cyclic cmd WKC validation . . . . .	74

3.7.2	Working Counter (WKC) State in Diagnosis Image . . . . .	74
3.7.3	Master Sync Units (MSU) . . . . .	75
3.8	EtherCAT traffic logging in application . . . . .	76
3.9	Trace Data . . . . .	76
3.9.1	Trace Data configuration via EC-Engineer . . . . .	77
3.9.2	Trace Data configuration via API . . . . .	79
3.10	EtherCAT Master Stack Source Code . . . . .	80
3.10.1	Components . . . . .	80
3.10.2	Excluding features . . . . .	81
3.11	Reduced Feature Set . . . . .	85
3.11.1	Rfs1: Convenience functionality excluded . . . . .	86
3.11.2	Rfs2: Rare functionalities excluded . . . . .	86
3.11.3	Rfs3: Common functionalities excluded . . . . .	86
3.11.4	Rfs4: Error detection and diagnosis excluded . . . . .	87
3.11.5	Rfs5: Only CoE slaves supported . . . . .	87
<b>4</b>	<b>Platform and Operating Systems (OS)</b> . . . . .	<b>88</b>
4.1	CMSIS-RTOS for STM32 . . . . .	88
4.1.1	Setting up and running EcMasterDemo in Keil µVision IDE . . . . .	88
4.1.2	OS Compiler settings . . . . .	88
4.1.3	Setting up and running EcMasterDemo in STM32CubeIDE for STM32H747I-DISCO . . . . .	89
4.2	eCos . . . . .	90
4.2.1	Setting up and running EcMasterDemo . . . . .	90
4.2.2	OS Compiler settings . . . . .	92
4.3	FreeRTOS . . . . .	92
4.3.1	Setting up and running EcMasterDemo on Xilinx Zynq UltraScale+ (ZCU104) and Xilinx Zynq-7000 (ZC702 Evaluation Kit) . . . . .	92
4.3.2	Setting up and running EcMasterDemo on TI AM64x EVM for R5 Core . . . . .	94
4.3.3	Setting up and running EcMasterDemo on TI AM243x LP and TI AM243x EVM . . . . .	94
4.3.4	Setting up and running EcMasterDemo on TI J784s4 EVM for R5 Core . . . . .	95
4.4	tenAsys INtime . . . . .	96
4.4.1	Setting up and running EcMasterDemo . . . . .	96
4.4.2	OS Compiler settings . . . . .	96
4.5	Linux . . . . .	97
4.5.1	OS optimizations . . . . .	97
4.5.2	atemsy kernel module . . . . .	98
4.5.3	Unbind Ethernet Driver instance . . . . .	99
4.5.4	Docker . . . . .	100
4.5.5	Setting up and running EcMasterDemo . . . . .	101
4.5.6	OS Compiler settings . . . . .	102
4.5.7	Build using cmake on Linux . . . . .	103
4.5.8	Cross-platform development under Windows . . . . .	103
4.6	PC / BIOS . . . . .	104
4.7	QNX Neutrino . . . . .	104
4.7.1	Thread priority . . . . .	104
4.7.2	Unbind Ethernet Driver instance . . . . .	104
4.7.3	IOMMU/SMMU support . . . . .	105
4.7.4	Setting up and running EcMasterDemo . . . . .	105
4.7.5	OS Compiler settings . . . . .	105
4.8	Renesas . . . . .	106
4.8.1	R-IN32M3 . . . . .	106
4.8.2	R-IN32M4 . . . . .	107
4.9	IntervalZero RTX . . . . .	107
4.9.1	Unbind Ethernet Driver instance . . . . .	108
4.9.2	Setting up and running EcMasterDemo . . . . .	108
4.9.3	OS Compiler settings . . . . .	108
4.10	SylixOS . . . . .	108
4.10.1	Setting up and running EcMasterDemo on SylixOS . . . . .	108

4.11	TI-RTOS . . . . .	109
4.11.1	Setting up and running EcMasterDemo . . . . .	109
4.11.2	OS Compiler settings . . . . .	110
4.12	µC3 for STM32 . . . . .	111
4.12.1	Setting up and running EcMasterDemo in IAR for ARM IDE . . . . .	111
4.12.2	OS Compiler settings . . . . .	112
4.13	µC3 for i.MX8 . . . . .	112
4.13.1	Setting up and running EcMasterDemo on NXP 8MPLUSLPD4-EVK board . . . . .	112
4.14	µC3 for Renesas RZ/N2H . . . . .	114
4.14.1	Setting up and running EcMasterDemo for RZ/N2H . . . . .	114
4.15	Windriver VxWorks . . . . .	115
4.15.1	VxWorks native . . . . .	115
4.15.2	SNARF Ethernet Driver . . . . .	116
4.15.3	Setting up and running EcMasterDemo . . . . .	116
4.15.4	OS Compiler settings . . . . .	118
4.16	Microsoft Windows . . . . .	119
4.16.1	EcMasterDemo . . . . .	119
4.16.2	EcMasterDemoDotNet (.NET) - Microsoft Windows . . . . .	120
4.16.3	EcMasterDemoGuiDotNet (.NET) - Microsoft Windows . . . . .	120
4.16.4	OS Compiler settings . . . . .	121
4.16.5	RtaccDevice for Real-time Ethernet Driver . . . . .	121
4.17	Microsoft Windows CE . . . . .	127
4.17.1	Identification of the Real-time Ethernet Driver . . . . .	128
4.17.2	KUKA CeWin . . . . .	129
4.17.3	Windows CE 5.0 . . . . .	130
4.17.4	Windows CE 6.0 . . . . .	130
4.17.5	Windows CE 2013 . . . . .	130
4.17.6	Setting up and running EcMasterDemo . . . . .	131
4.17.7	OS Compiler settings . . . . .	132
4.18	Xenomai . . . . .	133
4.18.1	Setting up and running EcMasterDemo . . . . .	134
4.18.2	OS compiler settings . . . . .	134
4.19	Zephyr . . . . .	135
4.19.1	Setting up and running EcMasterDemo . . . . .	135
4.19.2	OS Compiler settings . . . . .	136
<b>5</b>	<b>Real-time Ethernet Driver</b>	<b>137</b>
5.1	Real-time Ethernet Driver initialization . . . . .	138
5.1.1	Real-time Ethernet Driver instance selection via PCI location . . . . .	140
5.2	Broadcom Genet - emllBcmGenet . . . . .	141
5.3	Broadcom NetXtreme - emllBcmNetXtreme . . . . .	142
5.3.1	Supported PCI devices . . . . .	142
5.4	Berkeley Packet Filter - emllBPF . . . . .	142
5.5	Beckhoff CCAT - emllCCAT . . . . .	143
5.5.1	Supported PCI devices . . . . .	144
5.6	CMSIS - emllCmsisEth . . . . .	144
5.7	Texas Instruments CPSW - emllCPSW . . . . .	144
5.7.1	CPSW usage under Linux . . . . .	145
5.8	Texas Instruments CPSWG for AM6x and Jacinto 7 - emllCPSWG . . . . .	146
5.8.1	CPSWG usage under Linux . . . . .	147
5.9	Linux DPDK - emllDpdk . . . . .	148
5.9.1	DPDK for PCI device . . . . .	148
5.9.2	DPDK for DPAA . . . . .	149
5.9.3	Linux System Requirements . . . . .	151
5.9.4	Huge pages setup . . . . .	152
5.9.5	Limitations . . . . .	152
5.10	DW3504 - emllDW3504 . . . . .	152
5.10.1	Supported PCI devices . . . . .	154

5.11	Freescale TSEC / eTSEC - emllETSEC . . . . .	154
5.11.1	ETSEC supported MAC's . . . . .	156
5.11.2	Shared MII bus . . . . .	156
5.11.3	Locking . . . . .	156
5.11.4	Link check . . . . .	157
5.11.5	Fixed Link . . . . .	157
5.12	Freescale FslFec - emllFslFec . . . . .	157
5.13	Cadence GEM/MACB - emllGEM . . . . .	159
5.14	INtime HPE - emllHPE . . . . .	160
5.15	Intel Pro/100 - emllI8255x . . . . .	161
5.15.1	Supported PCI devices . . . . .	161
5.16	Texas Instruments ICSS - emllICSS . . . . .	161
5.16.1	TTS Feature . . . . .	163
5.16.2	TI AM335x ICEV2 . . . . .	163
5.16.3	TI AM57xx IDK . . . . .	163
5.16.4	AM5728 IDK and AM5718 IDK boards and Technical Limitations . . . . .	163
5.17	Texas Instruments ICSSG - emllICSSG on AM654x . . . . .	164
5.17.1	TI AM654x IDK . . . . .	164
5.18	Intel Pro/1000 - emllIntelGbe . . . . .	164
5.18.1	TTS Feature . . . . .	165
5.18.2	Supported PCI devices . . . . .	166
5.19	Microchip LAN743x - emlllan743x . . . . .	167
5.19.1	Supported PCI devices . . . . .	167
5.20	Beckhoff CUxxxx Multiplier - emllMultiplier . . . . .	168
5.20.1	Configuration with EC-Engineer . . . . .	168
5.21	Windows NDIS - emllNdis . . . . .	170
5.22	Windows WinPcap - emllPcap . . . . .	171
5.22.1	WinPcap, Npcap support . . . . .	171
5.23	RDC R6040 - emllR6040 . . . . .	172
5.23.1	Supported PCI devices . . . . .	173
5.24	emllRemote . . . . .	173
5.25	Realtek RTL8169 - emllRTL8169 . . . . .	174
5.25.1	RTL8169 usage under Linux . . . . .	174
5.25.2	Supported PCI devices . . . . .	174
5.26	Renesas RZ/T1 - emllRZT1 . . . . .	175
5.27	Renesas SHEth - emllSHEth . . . . .	175
5.27.1	SHEth link status update . . . . .	177
5.27.2	SHEth usage under Linux . . . . .	177
5.28	VxWorks SNARF - emllSNARF . . . . .	177
5.29	Linux SockRaw - emllSockRaw . . . . .	178
5.30	Linux SockXdp - emllSockXdp . . . . .	179
5.30.1	Linux System Requirements . . . . .	180
5.30.2	Getting started . . . . .	180
5.31	Texas Instruments CPSWG for AM6x and Jacinto 7 based on Enet LLD - emllTiEnetCpswg . . . . .	180
5.31.1	TI J784S4X EVM . . . . .	181
5.32	Texas Instruments ICSSG for AM6x and Jacinto 7 based on Enet LLD - emllTiEnetIcssg . . . . .	181
5.32.1	TI AM64x EVM . . . . .	181
5.32.2	TI AM243x LP and TI AM243x EVM . . . . .	181
5.33	Virtual Local Area Network - emllVlan . . . . .	181
<b>6</b>	<b>Application programming interface, reference</b> . . . . .	<b>183</b>
6.1	Generic API return status values . . . . .	183
6.2	Multiple EtherCAT Bus Support . . . . .	184
6.2.1	Licensing . . . . .	184
6.2.2	Overview . . . . .	184
6.2.3	Example application . . . . .	184
6.3	General functions . . . . .	185
6.3.1	emInitMaster . . . . .	185

6.3.2	emDeinitMaster . . . . .	189
6.3.3	emGetMasterParms . . . . .	190
6.3.4	emSetMasterParms . . . . .	190
6.3.5	emScanBus . . . . .	191
6.3.6	emRescueScan . . . . .	192
6.3.7	emConfigureNetwork . . . . .	193
6.3.8	emConfigGet . . . . .	196
6.3.9	emConfigExtend . . . . .	198
6.3.10	emRegisterClient . . . . .	198
6.3.11	emUnregisterClient . . . . .	200
6.3.12	emGetSrcMacAddress . . . . .	200
6.3.13	emSetMasterState . . . . .	201
6.3.14	emSetMasterStateReq . . . . .	202
6.3.15	emGetMasterState . . . . .	204
6.3.16	emGetMasterStateEx . . . . .	204
6.3.17	emStart . . . . .	205
6.3.18	emStop . . . . .	205
6.3.19	emExecJob . . . . .	206
6.3.20	emGetVersion . . . . .	210
6.3.21	emSetLicenseKey . . . . .	211
6.3.22	emSetOemKey . . . . .	212
6.3.23	emIoControl . . . . .	212
6.3.24	emIoControl - EC_IOCTL_GET_PDMEMORYSIZE . . . . .	213
6.3.25	emIoControl - EC_IOCTL_REGISTER_PDMEMORYPROVIDER . . . . .	214
6.3.26	emIoControl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB . . . . .	216
6.3.27	emIoControl - EC_IOCTL_ISLINK_CONNECTED . . . . .	217
6.3.28	emIoControl - EC_IOCTL_GET_LINKLAYER_MODE . . . . .	217
6.3.29	emIoControl - EC_IOCTL_GET_CYCLE_CONFIG_INFO . . . . .	218
6.3.30	emIoControl - EC_IOCTL_IS_SLAVETOSLAVE_COMM_CONFIGURED . . . . .	219
6.3.31	emIoControl - EC_LINKIOCTL . . . . .	219
6.3.32	emIoControl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS . . . . .	219
6.3.33	emIoControl - EC_LINKIOCTL_GET_SPEED . . . . .	220
6.3.34	emIoControl - EC_LINKIOCTL_GET_PCI_INFO . . . . .	220
6.3.35	emIoControl - EC_IOCTL_SET_CYCFRAME_LAYOUT . . . . .	222
6.3.36	emIoControl - EC_IOCTL_SET_MASTER_DEFAULT_TIMEOUTS . . . . .	223
6.3.37	emIoControl - EC_IOCTL_SET_COPYINFO_IN_SENDCYCFRAMES . . . . .	224
6.3.38	emIoControl - EC_IOCTL_SET_BUS_CYCLE_TIME . . . . .	224
6.3.39	emIoControl - EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES	225
6.3.40	emIoControl - EC_IOCTL_SLV_ALIAS_ENABLE . . . . .	225
6.3.41	emIoControl - EC_IOCTL_SET_IGNORE_INPUTS_ON_WKC_ERROR . . . . .	226
6.3.42	emIoControl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ERROR . . . . .	226
6.3.43	emIoControl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ZERO . . . . .	227
6.3.44	emIoControl - EC_IOCTL_SET_GENENI_ASSIGN_EEPROM_BACK_TO_EM . . . . .	227
6.3.45	emIoControl - EC_IOCTL_SET_EOE_DEFERRED_SWITCHING_ENABLED . . . . .	227
6.3.46	emIoControl - EC_IOCTL_SET_MAILBOX_POLLING_CYCLES . . . . .	228
6.3.47	emIoControl - EC_IOCTL_SET_MASTER_MAX_STATE . . . . .	228
6.3.48	emIoControl - EC_IOCTL_ACTIVATE_VOE_RECV_FIFO . . . . .	229
6.3.49	emIoControl - EC_IOCTL_SET_GEN_ENI_PARM . . . . .	229
6.3.50	emIoControl - EC_IOCTL_REALLOC_MBX_QUEUE . . . . .	230
6.4	Process Data Access . . . . .	231
6.4.1	emGetProcessData . . . . .	231
6.4.2	emGetProcessDataBits . . . . .	232
6.4.3	emSetProcessData . . . . .	232
6.4.4	emSetProcessDataBits . . . . .	233
6.4.5	emForceProcessDataBits . . . . .	234
6.4.6	emReleaseProcessDataBits . . . . .	235
6.4.7	emReleaseAllProcessDataBits . . . . .	236
6.4.8	emGetProcessImageInputPtr . . . . .	237

6.4.9	emGetProcessImageOutputPtr . . . . .	237
6.4.10	emGetDiagnosisImagePtr . . . . .	238
6.4.11	emGetDiagnosisImageSize . . . . .	238
6.4.12	emGetSlaveInpVarInfoNumOf . . . . .	238
6.4.13	emGetSlaveInpVarInfo . . . . .	239
6.4.14	emGetSlaveInpVarInfoEx . . . . .	241
6.4.15	emGetSlaveOutpVarInfoNumOf . . . . .	242
6.4.16	emGetSlaveOutpVarInfo . . . . .	243
6.4.17	emGetSlaveOutpVarInfoEx . . . . .	244
6.4.18	emGetSlaveInpVarByObjectEx . . . . .	245
6.4.19	emGetSlaveOutpVarByObjectEx . . . . .	246
6.4.20	emFindInpVarByName . . . . .	247
6.4.21	emFindInpVarByNameEx . . . . .	248
6.4.22	emFindOutpVarByName . . . . .	248
6.4.23	emFindOutpVarByNameEx . . . . .	249
6.4.24	emTraceDataConfig . . . . .	250
6.4.25	emTraceDataGetInfo . . . . .	250
6.4.26	EC_COPYBITS . . . . .	251
6.4.27	EC_COMPAREBITS . . . . .	252
6.4.28	EC_GET_FRM_WORD . . . . .	253
6.4.29	EC_GET_FRM_DWORD . . . . .	253
6.4.30	EC_GET_FRM_QWORD . . . . .	254
6.4.31	EC_SET_FRM_WORD . . . . .	254
6.4.32	EC_SET_FRM_DWORD . . . . .	254
6.4.33	EC_SET_FRM_QWORD . . . . .	255
6.4.34	EC_GETBITS . . . . .	255
6.4.35	EC_SETBITS . . . . .	256
6.5	Generic notification interface . . . . .	256
6.5.1	Notification callback: emNotify . . . . .	256
6.5.2	emNotify - EC_NOTIFY_STATECHANGED . . . . .	257
6.5.3	emNotify - EC_NOTIFY_XXXX . . . . .	258
6.5.4	Feature Pack Master Redundancy Notifications . . . . .	258
6.5.5	emNotifyApp . . . . .	258
6.5.6	emIoControl - EC_IOCTL_SET_NOTIFICATION_ENABLED . . . . .	259
6.5.7	emIoControl - EC_IOCTL_GET_NOTIFICATION_ENABLED . . . . .	260
6.6	Slave control and status functions . . . . .	261
6.6.1	emGetNumConfiguredSlaves . . . . .	261
6.6.2	emGetNumConnectedSlaves . . . . .	261
6.6.3	emGetSlaveId . . . . .	262
6.6.4	emGetSlaveIdAtPosition . . . . .	262
6.6.5	emSetSlaveState . . . . .	263
6.6.6	emSetSlaveStateReq . . . . .	264
6.6.7	emGetSlaveState . . . . .	265
6.6.8	emIsSlavePresent . . . . .	266
6.6.9	emGetSlaveProp . . . . .	267
6.6.10	emSlaveSerializeMbxTfers . . . . .	268
6.6.11	emSlaveParallelMbxTfers . . . . .	268
6.6.12	emIoControl - EC_IOCTL_SET_MBX_RETRYACCESS_PERIOD . . . . .	269
6.6.13	emNotify - EC_NOTIFY_SLAVE_STATECHANGED . . . . .	269
6.6.14	emNotify - EC_NOTIFY_SLAVES_STATECHANGED . . . . .	270
6.6.15	emWriteSlaveRegister . . . . .	271
6.6.16	emWriteSlaveRegisterReq . . . . .	272
6.6.17	emReadSlaveRegister . . . . .	273
6.6.18	emReadSlaveRegisterReq . . . . .	274
6.6.19	emNotify - EC_NOTIFY_SLAVE_REGISTER_TRANSFER . . . . .	276
6.6.20	emReadSlaveEEPROM . . . . .	277
6.6.21	emReadSlaveEEPROMReq . . . . .	278
6.6.22	emWriteSlaveEEPROM . . . . .	279

6.6.23	emWriteSlaveEEPROMReq . . . . .	280
6.6.24	emAssignSlaveEEPROM . . . . .	281
6.6.25	emAssignSlaveEEPROMReq . . . . .	282
6.6.26	emActiveSlaveEEPROM . . . . .	283
6.6.27	emActiveSlaveEEPROMReq . . . . .	284
6.6.28	emReloadSlaveEEPROM . . . . .	285
6.6.29	emReloadSlaveEEPROMReq . . . . .	286
6.6.30	emNotify - EC_NOTIFY_EEPROM_OPERATION . . . . .	287
6.6.31	emResetSlaveController . . . . .	289
6.6.32	emIoControl - EC_IOCTL_ALL_SLAVES_MUST_REACH_MASTER_STATE . . . . .	290
6.6.33	emGetCfgSlaveInfo . . . . .	290
6.6.34	emGetCfgSlaveEoeInfo . . . . .	294
6.6.35	emGetCfgSlaveSmInfo . . . . .	296
6.6.36	emGetBusSlaveInfo . . . . .	297
6.6.37	emReadSlaveIdentification . . . . .	300
6.6.38	emReadSlaveIdentificationReq . . . . .	301
6.6.39	emNotify - EC_NOTIFY_SLAVE_IDENTIFICATION . . . . .	303
6.6.40	emIoControl - EC_IOCTL_SET_AUTO_ACK_AL_STATUS_ERROR_ENABLED . . . . .	304
6.6.41	emIoControl - EC_IOCTL_SET_AUTO_ADJUST_CYCCMD_WKC_ENABLED . . . . .	304
6.6.42	emSetSlaveDisabled . . . . .	305
6.6.43	emIoControl - EC_IOCTL_SET_SLAVE_MAX_STATE . . . . .	305
6.6.44	emSetSlaveDisconnected . . . . .	306
6.6.45	emSetSlavesDisconnected . . . . .	307
6.6.46	emGetSlavePortState . . . . .	308
6.6.47	emSetSlavePortState . . . . .	309
6.6.48	emSetSlavePortStateReq . . . . .	310
6.6.49	emNotify - EC_NOTIFY_PORT_OPERATION . . . . .	311
6.6.50	emIoControl - EC_IOCTL_SET_NEW_BUSSLAVES_TO_INIT . . . . .	312
6.7	Diagnosis, error detection, error notifications . . . . .	312
6.7.1	emSetLogParms . . . . .	313
6.7.2	emEthDbgMsg . . . . .	314
6.7.3	emIoControl - EC_IOCTL_GET_SLVSTATISTICS . . . . .	314
6.7.4	emGetSlaveStatistics . . . . .	315
6.7.5	emIoControl - EC_IOCTL_CLR_SLVSTATISTICS . . . . .	316
6.7.6	emClearSlaveStatistics . . . . .	316
6.7.7	emIoControl - EC_IOCTL_GET_SLVSTAT_PERIOD . . . . .	317
6.7.8	emIoControl - EC_IOCTL_SET_SLVSTAT_PERIOD . . . . .	317
6.7.9	emIoControl - EC_IOCTL_FORCE_SLVSTAT_COLLECTION . . . . .	318
6.7.10	emIoControl - EC_IOCTL_CLEAR_MASTER_INFO_COUNTERS . . . . .	318
6.7.11	emIoControl - EC_IOCTL_SET_FRAME_RESPONSE_ERROR_NOTIFY_MASK . . . . .	319
6.7.12	emIoControl - EC_IOCTL_SET_FRAME_LOSS_SIMULATION . . . . .	320
6.7.13	emIoControl - EC_IOCTL_SET_RXFRAME_LOSS_SIMULATION . . . . .	321
6.7.14	emIoControl - EC_IOCTL_SET_TXFRAME_LOSS_SIMULATION . . . . .	321
6.7.15	Error notifications - general information . . . . .	322
6.7.16	emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR . . . . .	323
6.7.17	emNotify - EC_NOTIFY_MASTER_INITCMD_WKC_ERROR . . . . .	325
6.7.18	emNotify - EC_NOTIFY_SLAVE_INITCMD_WKC_ERROR . . . . .	325
6.7.19	emNotify - EC_NOTIFY_FOE_MBSLAVE_ERROR . . . . .	325
6.7.20	emNotify - EC_NOTIFY_EOE_MBXSND_WKC_ERROR . . . . .	325
6.7.21	emNotify - EC_NOTIFY_COE_MBXSND_WKC_ERROR . . . . .	325
6.7.22	emNotify - EC_NOTIFY_FOE_MBXSND_WKC_ERROR . . . . .	325
6.7.23	emNotify - EC_NOTIFY_VOE_MBXSND_WKC_ERROR . . . . .	326
6.7.24	emNotify - EC_NOTIFY_S2SMBX_ERROR . . . . .	326
6.7.25	emNotify - EC_NOTIFY_FRAME_RESPONSE_ERROR . . . . .	326
6.7.26	emNotify - EC_NOTIFY_SLAVE_INITCMD_RESPONSE_ERROR . . . . .	328
6.7.27	emNotify - EC_NOTIFY_MBSLAVE_INITCMD_TIMEOUT . . . . .	329
6.7.28	emNotify - EC_NOTIFY_MASTER_INITCMD_RESPONSE_ERROR . . . . .	329
6.7.29	emNotify - EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL . . . . .	330

6.7.30	emNotify - EC_NOTIFY_ALL_DEVICES_OPERATIONAL . . . . .	330
6.7.31	emNotify - EC_NOTIFY_STATUS_SLAVE_ERROR . . . . .	330
6.7.32	emNotify - EC_NOTIFY_SLAVE_ERROR_STATUS_INFO . . . . .	330
6.7.33	emNotify - EC_NOTIFY_SLAVES_ERROR_STATUS . . . . .	331
6.7.34	emNotify - EC_NOTIFY_SLAVE_UNEXPECTED_STATE . . . . .	331
6.7.35	emNotify - EC_NOTIFY_SLAVES_UNEXPECTED_STATE . . . . .	332
6.7.36	emNotify - EC_NOTIFY_ETH_LINK_NOT_CONNECTED . . . . .	332
6.7.37	emNotify - EC_NOTIFY_ETH_LINK_CONNECTED . . . . .	333
6.7.38	emNotify - EC_NOTIFY_CLIENTREGISTRATION_DROPPED . . . . .	333
6.7.39	emNotify - EC_NOTIFY_EEPROM_CHECKSUM_ERROR . . . . .	333
6.7.40	emNotify - EC_NOTIFY_MBXRcv_INVALID_DATA . . . . .	333
6.7.41	emNotify - EC_NOTIFY_PDIWATCHDOG . . . . .	333
6.7.42	ecatGetText . . . . .	334
6.7.43	emLogFrameEnable . . . . .	334
6.7.44	emLogFrameDisable . . . . .	335
6.7.45	emGetMasterInfo . . . . .	336
6.7.46	emGetMemoryUsage . . . . .	339
6.7.47	emGetMasterDump . . . . .	340
6.7.48	emGetMasterSyncUnitInfoNumOf . . . . .	341
6.7.49	emGetMasterSyncUnitInfo . . . . .	341
6.7.50	emBadConnectionsDetect . . . . .	342
6.7.51	emBadConnectionsReset . . . . .	343
6.7.52	emNotify - EC_NOTIFY_BAD_CONNECTION . . . . .	343
6.7.53	emSelfTestScan . . . . .	344
6.8	Performance Measurement . . . . .	346
6.8.1	Enabling performance measurements . . . . .	346
6.8.2	Retrieving overall performance statistics (min/avg/max) . . . . .	346
6.8.3	Recording performance histograms . . . . .	347
6.8.4	Special benchmark types . . . . .	348
6.8.5	Application benchmarks . . . . .	348
6.8.6	API . . . . .	348
6.9	EtherCAT Mailbox Transfer . . . . .	356
6.9.1	Mailbox transfer object states . . . . .	357
6.9.2	emMbxTferCreate . . . . .	358
6.9.3	emMbxTferAbort . . . . .	362
6.9.4	emMbxTferDelete . . . . .	362
6.9.5	emNotify - EC_NOTIFY_MBOXRCV . . . . .	362
6.10	Automation Device Specification over EtherCAT (AoE) . . . . .	363
6.10.1	emAoeGetSlaveNetId . . . . .	363
6.10.2	emAoeRead . . . . .	364
6.10.3	emAoeReadReq . . . . .	365
6.10.4	emNotify - eMbxTferType_AOE_READ . . . . .	366
6.10.5	emAoeWrite . . . . .	367
6.10.6	emAoeWriteReq . . . . .	368
6.10.7	emNotify - eMbxTferType_AOE_WRITE . . . . .	369
6.10.8	emAoeReadWrite . . . . .	369
6.10.9	emAoeWriteControl . . . . .	370
6.10.10	emConvertEcErrorToAdsError . . . . .	372
6.11	CAN application protocol over EtherCAT (CoE) . . . . .	372
6.11.1	emCoeSdoDownload . . . . .	372
6.11.2	emCoeSdoDownloadReq . . . . .	374
6.11.3	emNotify - eMbxTferType_COE_SDO_DOWNLOAD . . . . .	376
6.11.4	emCoeSdoUpload . . . . .	377
6.11.5	emCoeSdoUploadReq . . . . .	379
6.11.6	emNotify - eMbxTferType_COE_SDO_UPLOAD . . . . .	381
6.11.7	emCoeGetODListReq . . . . .	382
6.11.8	emNotify - eMbxTferType_COE_GETODLIST . . . . .	384
6.11.9	emCoeGetObjectDescReq . . . . .	385

6.11.10 emNotify - eMbxTferType_COE_GETOBDESC . . . . .	387
6.11.11 emCoeGetEntryDescReq . . . . .	388
6.11.12 emNotify - eMbxTferType_COE_GETENTRYDESC . . . . .	391
6.11.13 emCoeProfileGetChannelInfo . . . . .	392
6.11.14 emNotify - EC_NOTIFY_COE_INIT_CMD . . . . .	393
6.11.15 CoE Emergency (emNotify - eMbxTferType_COE_EMERGENCY)	395
6.11.16 CoE Abort (emNotify - EC_NOTIFY_MBSLAVE_COE_SDO_ABORT)	396
6.11.17 emConvertEcErrorToCoeError . . . . .	397
<b>6.12 File access over EtherCAT (FoE) . . . . .</b>	<b>397</b>
6.12.1 Specification . . . . .	397
6.12.2 emFoeFileDownload . . . . .	400
6.12.3 emFoeFileUpload . . . . .	402
6.12.4 emFoeDownloadReq . . . . .	404
6.12.5 emFoeSegmentedDownloadReq . . . . .	405
6.12.6 emFoeUploadReq . . . . .	406
6.12.7 emFoeSegmentedUploadReq . . . . .	407
6.12.8 emConvertEcErrorToFoeError . . . . .	410
6.12.9 emNotify - EC_NOTIFY_FOE_MBXSND_WKC_ERROR . . . . .	411
6.12.10 emNotify - EC_NOTIFY_FOE_MBSLAVE_ERROR . . . . .	411
6.12.11 Extending EC_T_MBX_DATA . . . . .	411
<b>6.13 Servo Drive Profil according to IEC61491 over EtherCAT (SoE) . . . . .</b>	<b>412</b>
6.13.1 SoE ElementFlags . . . . .	413
6.13.2 SoE IDN coding . . . . .	413
6.13.3 emSoeWrite . . . . .	414
6.13.4 emSoeWriteReq . . . . .	415
6.13.5 emSoeRead . . . . .	416
6.13.6 emSoeReadReq . . . . .	417
6.13.7 emSoeAbortProcCmd . . . . .	418
6.13.8 emConvertEcErrorToSoeError . . . . .	419
6.13.9 emNotify - EC_NOTIFY_SOE_MBXSND_WKC_ERROR . . . . .	419
6.13.10 emNotify - EC_NOTIFY_SOE_WRITE_ERROR . . . . .	419
<b>6.14 Vendor specific protocol over EtherCAT (VoE) . . . . .</b>	<b>419</b>
6.14.1 emVoeWrite . . . . .	420
6.14.2 emVoeWriteReq . . . . .	421
6.14.3 emVoeRead . . . . .	421
6.14.4 emNotify - eMbxTferType_VOE_READ . . . . .	423
6.14.5 emNotify - eMbxTferType_VOE_WRITE . . . . .	423
<b>6.15 Raw command transfer . . . . .</b>	<b>424</b>
6.15.1 emTferSingleRawCmd . . . . .	424
6.15.2 emClnSendRawMbx . . . . .	425
6.15.3 emClnQueueRawCmd . . . . .	426
6.15.4 emQueueRawCmd . . . . .	428
6.15.5 emNotify - EC_NOTIFY_RAWCMD_DONE . . . . .	428
<b>6.16 EtherCAT Bus Scan . . . . .</b>	<b>429</b>
6.16.1 emIoControl - EC_IOCTL_SB_ENABLE . . . . .	429
6.16.2 emIoControl - EC_IOCTL_SB_RESTART . . . . .	430
6.16.3 emIoControl - EC_IOCTL_SB_STATUS_GET . . . . .	430
6.16.4 emIoControl - EC_IOCTL_SB_SET_TOPOLOGY_CHANGED_DELAY . . . . .	431
6.16.5 emIoControl - EC_IOCTL_SB_SET_TOPOLOGY_CHANGED_DELAYS . . . . .	431
6.16.6 emIoControl - EC_IOCTL_SB_SET_ERROR_ON_CROSSED_LINES . . . . .	432
6.16.7 emIoControl - EC_IOCTL_SB_SET_ERROR_ON_LINEBREAK . . . . .	432
6.16.8 emIoControl - EC_IOCTL_SB_SET_TOPOLOGY_CHANGE_AUTO_MODE . . . . .	433
6.16.9 emIoControl - EC_IOCTL_SB_ACCEPT_TOPOLOGY_CHANGE . . . . .	433
6.16.10 emNotify - EC_NOTIFY_SB_STATUS . . . . .	434
6.16.11 emNotify - EC_NOTIFY_SB_MISMATCH . . . . .	434
6.16.12 emNotify - EC_NOTIFY_SB_DUPLICATE_HC_NODE . . . . .	436
6.16.13 emNotify - EC_NOTIFY_SLAVE_PRESENCE . . . . .	436
6.16.14 emNotify - EC_NOTIFY_SLAVES_PRESENCE . . . . .	437

6.16.15 emNotify - EC_NOTIFY_LINE_CROSSED . . . . .	438
6.16.16 emNotify - EC_NOTIFY_SLAVE_NOTSUPPORTED . . . . .	438
6.16.17 emNotify - EC_NOTIFY_FRAMELOSS_AFTER_SLAVE . . . . .	439
6.16.18 emNotify - Bus Scan notifications for Feature Packs . . . . .	439
6.16.19 emIoControl - EC_IOCTL_SB_NOTIFY_UNEXPECTED_BUS_SLAVES . . . . .	439
6.16.20 emIsTopologyChangeDetected . . . . .	440
6.16.21 emNotify - EC_NOTIFY_HC_TOPOCHGDONE . . . . .	440
6.16.22 emIoControl - EC_IOCTL_SB_SET_NO_DC_SLAVES_AFTER_JUNCTION . . . . .	441
<b>7 RAS-Server for EC-Lyser and EC-Engineer</b>	<b>442</b>
7.1 Integration Requirements . . . . .	442
7.2 Application programming interface, reference . . . . .	442
7.2.1 emRasSrvStart . . . . .	442
7.2.2 emRasSrvStop . . . . .	444
7.2.3 emRasNotify - xxx . . . . .	445
7.2.4 emRasNotify - ECMASTERRAS_NOTIFY_CONNECTION . . . . .	445
7.2.5 emRasNotify - ECMASTERRAS_NOTIFY_REGISTER . . . . .	445
7.2.6 emRasNotify - ECMASTERRAS_NOTIFY_UNREGISTER . . . . .	446
7.2.7 emRasNotify - ECMASTERRAS_NOTIFY_MARSHALERROR . . . . .	446
7.2.8 emRasNotify - ECMASTERRAS_NOTIFY_ACKERROR . . . . .	447
7.2.9 emRasNotify - ECMASTERRAS_NOTIFY_NONOTIFYMEMORY . . . . .	447
7.2.10 emRasNotify - ECMASTERRAS_NOTIFY_STDNOTIFYMEMORYSMALL . . . . .	448
7.2.11 emRasNotify - ECMASTERRAS_NOTIFY_MBXNOTIFYMEMORYSMALL . . . . .	448
<b>8 Error Codes</b>	<b>449</b>
8.1 Groups . . . . .	449
8.2 Generic Error Codes . . . . .	449
8.3 DCM Error Codes . . . . .	460
8.4 ADS over EtherCAT (AoE) Error Codes . . . . .	462
8.5 CAN application protocol over EtherCAT (CoE) SDO Error Codes . . . . .	464
8.6 File Transfer over EtherCAT (FoE) Error Codes . . . . .	467
8.7 Servo Drive Profil over EtherCAT (SoE) Error Codes . . . . .	469
8.8 Remote API Error Codes . . . . .	473

## 1 Introduction

### 1.1 What is EtherCAT?

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

**See also:**

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

### 1.2 The EC-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	Class A	Class B	Feature ID
<b>Basic Features</b>				
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
<b>Process Data Exchange</b>				
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
<b>Network Configuration</b>				
Online scanning	Network configuration functionality included in EtherCAT Master	✓	✓	301
Reading ENI	Network Configuration taken from ENI file	✓	✓	301
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

continues on next page

Table 1 – continued from previous page

Feature name	Short description	Class A	Class B	Feature ID
Access to EEPROM	Support routines to access EEPROM via ESC register	✓	✓	305
<b>Mailbox Support</b>				
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
<b>CAN application layer over EtherCAT (CoE)</b>				
SDO Up/Download	Normal and expedited transfer	✓	✓	501
Segmented Transfer	Segmented transfer	✓	✓	502
Complete Access	Transfer the entire object (with all subindices) 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
<b>EoE</b>				
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
<b>FoE</b>				
FoE Protocol	Support FoE Protocol	✓	✓	701
Firmware Up-/Download	Password, FileName should be given by the application	✓	✓	702
Boot State	Support Boot-State for Firmware Up/Download	✓	✓	703
<b>SoE</b>				
SoE Protocol	Support SoE Services	✓	✓	801
<b>AoE</b>				
AoE Protocol	Support AoE Protocol	✓	✓	901
<b>VoE</b>				
VoE Protocol	External Connectivity supported	✓	✓	1001
<b>Synchronization with Distributed Clock (DC)</b>				
DC support	Support of Distributed Clocks	✓	-(*)2	1101
Continuous Propagation Delay compensation	Continuous Calculation of the propagation delay	✓	-(*)2	1102
Sync window monitoring	Continuous monitoring of the Synchronization difference in the slaves	✓	-(*)2	1103
<b>Slave-to-Slave Communication</b>				
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
<b>Master information</b>				
Master Object Dictionary		FP	-(*)2	1301

## 1.3 Protected version

The EC-Master software can be delivered in 3 different versions:

### Protected

Binary with MAC protection

### Unrestricted

Binary without MAC protection

### Source

Source code

The protected version will automatically stop after about 1 hour of continuous operation. In order to remove this restriction a valid runtime license key is required. The runtime license protection is based on the MAC address of the Ethernet controller used for the EtherCAT protocol. With a valid License Key the protected version of EC-Master will automatically become an unrestricted version.

### 1.3.1 Licensing procedure for Development Licenses

1. Installation of EC-Master protected version
2. Determine the MAC Address by calling `emGetSrcMacAddress()` or from a sticker applied on the hardware near the Ethernet controller
3. Send an Email with the subject “**Development License Key Request, Commission your commission number**” with the MAC address to [sales@acontis.com](mailto:sales@acontis.com)
4. Acontis will create the license keys and return them in a License Key Text File (CSV format).

```
Number;MAC Address;License Key
1;00-00-5A-11-77-FE;DA1099F2-15C249E9-54327FBC
2;64-31-50-80-20-4E;1B7C1F86-D08E40A8-4F96F2BA
3;2C-F0-5D-03-CB-2B;10005078-DFD9A2C3-5FD4B1CD-35041597-F8094AA4-6C7CCE7E
```

5. Activate the License Key by calling `emSetLicenseKey()` with the license key that corresponds to the MAC address on the hardware and check the return code. The license key is 26 or 53 characters long.

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

### 1.3.2 Licensing procedure for Runtime Licenses

1. Installation of EC-Master protected version
2. Determine the MAC Address by calling `emGetSrcMacAddress()` or from a sticker applied on the hardware near the Ethernet controller
3. Provide the MAC Addresses and numbers from previously ordered and unused runtime license stickers in a text file to acontis as described in the example below. Please use a separate line for each runtime license sticker number and MAC Address.

```
S/N; MAC Address
100-105-1-1/1603310001;00-00-5A-11-77-FE
100-105-1-1/1603310002;64-31-50-80-20-4E
100-105-1-1/1603310003;2C-F0-5D-03-CB-2B
```

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

```
Number;MAC Address;License Key
1;00-00-5A-11-77-FE;DA1099F2-15C249E9-54327FBC
2;64-31-50-80-20-4E;1B7C1F86-D08E40A8-4F96F2BA
3;2C-F0-5D-03-CB-2B;10005078-DFD9A2C3-5FD4B1CD-35041597-F8094AA4-6C7CCE7E
```

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

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

## 1.4 License

### 1.4.1 EC-Master license

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

### 1.4.2 Free Open Source Software contained in EC-Master

#### Expat XML parser license V2.6.0

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

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

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

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

### 1.4.3 Free Open Source Software supported by EC-Master

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

#### acontis atemsys Linux kernel module

The acontis atemsys is licensed under the GPL:

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

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

#### WinPCap

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

#### Npcap

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

## 1.5 Versioning

EC-Master follows the following versioning scheme: **VMAJOR. MINOR. SERVICEPACK. BUILD** (e.g. V3.2.1.04).

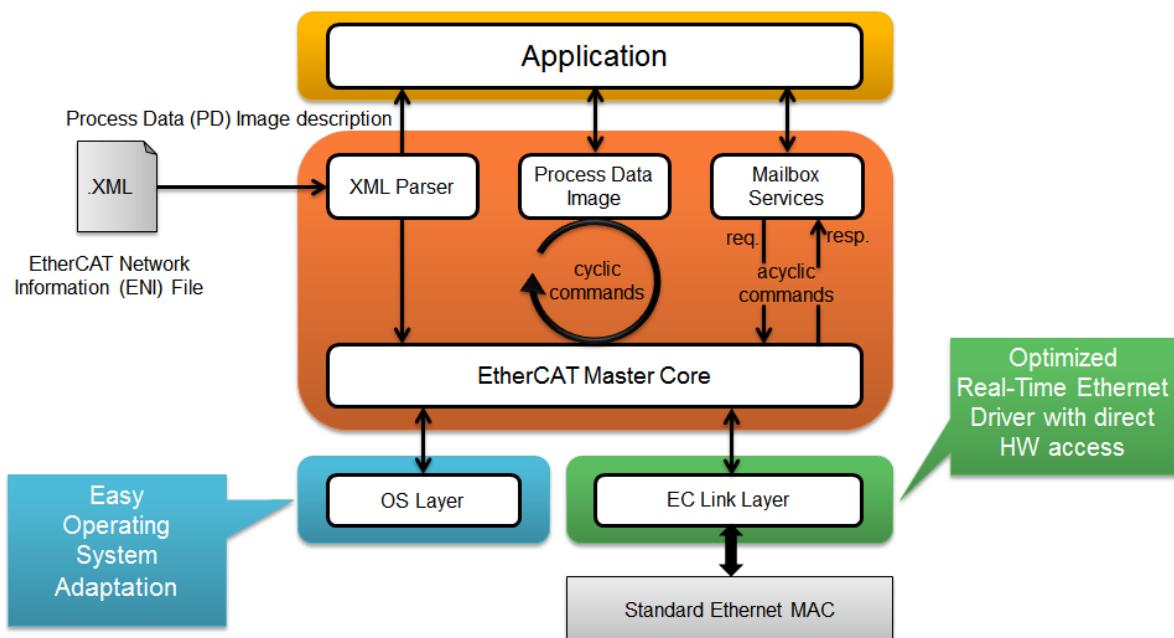
The libraries are binary compatible by unchanged *MAJOR* and *MINOR* digits. If *SERVICEPACK* increments, *BUILD* restarts with 01. *BUILD* 99 is reserved for test builds that have not been officially released for productive usage.

## 2 Getting Started

### 2.1 EC-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.
- Real-time Ethernet Driver 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 interoperability 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

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

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

## 2.4 Running EcMasterDemo

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

Start the EcMasterDemo from the command line to put the EtherCAT network into operation. At least a Real-time Ethernet Driver must be specified.

```
> EcMasterDemo -ndis 192.168.157.2 1 -f eni.xml -t 0 -v 3
```

### See also:

- *Example application* for detailed explanation

### 2.4.1 Command line parameters

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

The parameters are as follows:

**-f** <ENI-FileName>  
Path to ENI file

**-t** <time>  
Running duration in msec. When the time expires the demo application exits completely.

**<time>**  
Time in msec, 0 = forever (default = 120000)

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

**<cycle time>**  
Bus cycle time in µsec

**-a** <affinity>  
The CPU affinity specifies which CPU the demo application ought to use.

**<affinity>**  
0 = first CPU, 1 = second, ...

**-v <level>**

The verbosity level specifies how much console output messages will be generated by the demo application. A high verbosity level leads to more messages.

**<level>**

Verbosity level: 0=off (default), 1..n=more messages

**-perf [<level>]**

Enable max. and average time measurement in  $\mu$ s for all EtherCAT jobs (e.g. ProcessAllRxFrames).

**<level>**

Depending on level the performance histogram can be activated as well.

**-log <prefix> [<msg cnt>]**

Use given file name prefix for log files.

**<prefix>****<msg cnt>**

Messages count for log buffer allocation

**-lic <key>**

Set License key.

**<key>**

License key string

**-oem <key>**

Use OEM key

**<key>**

64 bit OEM key.

**-junctionred**

Enable junction redundancy (automatic mode)

**-flash <address>**

Flash outputs

**<address>**

0=all, >0 = slave station address

**-sp [<port>]**

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

**<port>**

RAS server port

**-rec [<prefix> [<frame cnt>]]**

Packet capture file recording

**<prefix>**

File name prefix

**<frame cnt>**

Frame count for log buffer allocation

## Link Layer

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

---

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

---

**-alteratse** <instance> <mode>

<instance>

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

<mode>

0 = Interrupt mode | 1 = Polling mode

**-antaios**

Device instance fixed to 2

Mode fixed to 1 = Polling mode

**-bcmgenet** <instance> <mode>

**Hardware: Broadcom BcmGenet**

<instance>

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

<mode>

0 = Interrupt mode | 1 = Polling mode

**-bcmnetxtreme** <instance> <mode>

**Hardware: Broadcom NetXtreme**

<instance>

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

<mode>

0 = Interrupt mode | 1 = Polling mode

**-bpf** <instance> <mode> <interface> [<prefix>]

**Hardware: Berkeley Packet Filter, Hardware independent**

<instance>

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

<mode>

0 = Interrupt mode | 1 = Polling mode

<interface>

Name of Ethernet Interface, e.g. wm0

**Optional:**

<prefix>

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

**-ccat** <instance> <mode>

**Hardware: Beckhoff CCAT**

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

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

**-cpsz** <instance> <mode> <portpriority> <masterflag> <refboard>

**Hardware: TI CPSW**

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

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

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

**<masterflag>**

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

**<RefBoard>**  
bone | am3359-icev2 | am437x-idk | am572x-idk | 387X\_evm | custom | osdriver

**If custom:**

**<CpswType>**  
am33XX | am437X | am57X | am387X

**<PhyAddress>**  
0 ... 31

**<PhyInterface>**  
rmii | gmii | rgmii | osdriver

**<NotUseDmaBuffers>**  
0 = FALSE | 1 = TRUE

**-dpdk** <instance> <mode> <port>

**Hardware: Hardware independent, only available for Linux (including the NXP DPAA).**

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

**<mode>**  
0 = Interrupt mode (not implemented), 1 = Polling mode

**<port>**  
DPDK port ID, e.g. 0

This option supports a large variety of NICs. Whether a NIC is supported depends on its network driver. For a list of supported network drivers see <https://www.dpdk.org>.

**-dw3504** <instance> <mode> <RefBoard>

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

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

**<mode>**

0 = Interrupt mode | 1 = Polling mode

**<RefBoard>**

Reference Board: intel\_atom | lces1 | rd55up06 | r12ccpu | rzn1 | socrates | stm32mp157a-dk1 | custom

**If custom:****<DW3504Type>**

intel\_atom | cycloneV | lces1 | stm32mp157a-dk1

**<PhyInterface>**

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

**<PhyAddress>**

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

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

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

**<mode>**

0 = Interrupt mode | 1 = Polling mode

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

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

**<mode>**

0 = Interrupt mode | 1 = Polling mode

**<RefBoard>**

MC2002E | custom

**If custom:****<RegisterBase>**

Register base address as hex value

**<RegisterLength>**

Register length as hex value

**<NotUseDmaBuffers>**

0 = FALSE | 1 = TRUE

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

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

**<mode>**

0 = Interrupt mode | 1 = Polling mode

**<RefBoard>**

p2020rdb | twrp1025 | istmpc8548 | xj\_epu20c | twrls1021a | tqmls\_ls102xa | custom

**If custom:**

**<PhyAddress>**  
0 ... 31

**<RxIrq>**  
Default depending on ETSEC type

**<NotUseDmaBuffers>**  
0 = FALSE | 1 = TRUE

**-fs1fec** <instance> <mode> <refboard> [nopinmuxing] [nomacaddr]

**Hardware: Freescale FEC/ENET**

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

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

**<RefBoard>**  
mars | sabrelite | sabresd | imx28evk | topaz | imxceetul2 | mimxrt1064-evk | imx93evk |  
custom

**If custom:**

**<FecType>**  
imx25 | imx28 | imx53 | imx6 | vf6 | imx7 | imx8 | imx8m | imxrt1064 | imx9

**<PhyInterface>**  
fixed | mii | rmii | rmii50Mhz | gmii | sgmii | rgmii | osdriver

**<PhyAddress>**  
0 ... 31, default 0 (don't use if osdriver)

**Optional:**

**nopinmuxing**  
no pin muxing

**Optional:**

**nomacaddr**  
don't read MAC address

**-gem** <instance> <mode> <refboard> [osdriver] [clkdivtype\_k26] [nopinmuxing]

**Hardware: Cadence GEM/MACB**

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

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

**<RefBoard>**  
zc702 | zedboard | microzed | zcu102 | zcu104 | KR260 | rpi5 | custom

**If custom:**

**<PhyAddress>**  
0 ... 31

**<PhyConnectionMode>**  
MIO (0) or EMIO (1)

**<UseGmiiToRgmii>**  
Use Xilinx GmiiToRgmii converter TRUE (1) or FALSE (0)

**<GmiiToRgmiiPort>**  
GmiiToRgmii converter PHY address 0 ... 31

**<GemType>**  
zynq7000 | ultrascale | bcm2712

**Optional:**

**osdriver**  
PhyInterface osdriver

**Optional:**

**clkdivtype\_k26**  
Clock divisor

**Optional:**

**nopinmuxing**  
Don't use pin muxing

**-intelgbe <instance> <mode> [tts <SendOffset>|tmr] [--nophyctrlonconnect]**

**Hardware: Intel Pro/1000 network adapter card**

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

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

**Optional:**

**tts** Enables Real-time Ethernet Driver Time Triggered Send (TTS)

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

**Optional:**

**tmr**  
Enables Real-time Ethernet Driver Timer

**Optional:**

**--nophyctrlonconnect**  
Disable PHY control (e.g. PHY reset, PHY PM settings, Gbits Ctrl) on link connection detected

**-i8255x <instance> <mode>**

**Hardware: Intel Pro/100 network adapter card**

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

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

-  
**icss** <instance> <mode> <masterflag> <refboard> [<PhyInterface> <PhyAddress>] [no-phyreset] [tts <SendOffset>]

#### Hardware: Texas Instruments Board with PRUSS

**<instance>**  
ICSS Port (100 Mbit/s) 1 ... 4

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

**<MasterFlag>**

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

**<RefBoard>**  
am572x-idk | am571x-idk | am3359-icev2 | am574x

#### Optional:

**<PhyInterface>**  
mii | osdriver

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

#### Optional:

**nophyreset**  
NoPhyReset

#### Optional:

**tts** Enables Real-time Ethernet Driver Time Triggered Send (TTS)

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

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

#### Hardware: Texas Instruments AARCH64 Board with Gigabit PRUSS

**<instance>**  
ICSSG Port 1 ... 6

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

**<MasterFlag>**

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

**<RefBoard>**  
am654x-idk

-**19218i** <instance> <mode>

#### Hardware: SMSC LAN9218i/LAN9221

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

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

**-lan743x <instance> <mode>**

#### **Hardware: Microchip LAN743x**

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

**<mode>**  
1= Polling mode

**-ndis <IpAddress> <mode> [--name <AdapterName>] [DisablePromiscuousMode] [DisableForceBroadcast]**

#### **Hardware: Hardware independent, only available for Windows.**

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

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

#### **Optional:**

**--name**  
Select network adapter by name

**<AdapterName>**  
Service name from HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkCards

#### **Optional:**

**DisablePromiscuousMode**  
Disable promiscuous mode

#### **Optional:**

**DisableForceBroadcast**

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

#### **Hardware: Beckhoff CUxxxx Ethernet-Port-Multiplier**

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

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

**<port>**  
used CU2508 downlink port 0 = X1, 1 = X2, ...

**<link parms>**  
link parms of network adapter connected to the uplink port e.g. -intelge ...

**Optional:**

**<type>**  
cu2508 = CU2508 Ethernet-Port-Multiplier

**-r6040 <instance> <mode>**

**Hardware: RDC R6040**

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

**<mode>**  
1 = Polling mode

**-rin32m3 <instance> <mode>**

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

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

**<mode>**  
1 = Polling mode

**-rt18139 <instance> <mode>**

**Hardware: Realtek RTL8139**

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

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

**-rt18169 <instance> <mode>**

**Hardware: Realtek RTL8168 / RTL8169 / RTL8111**

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

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

**-rztl1 <instance>**

**Hardware: Renesas RZ/T1**

**<instance>**  
Device instance 1 = Port 0 | 2 = Port 1

**-sheth <instance> <mode> <RefBoard>**

**Hardware: Renesas RZG1 or Armadillo-800 EVA**

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

**<mode>**  
1 = Polling mode

**<RefBoard>**  
rzg1e | a800eva

**-snarf** <adapterName>**Hardware:** Hardware independent, only available for VxWorks**<adapterName>**

Adapter name, e.g. fei0

**-sockraw** <device> [<mode>] [--nommaprx] [--promiscuousmode]**Hardware:** Hardware independent, only available for Linux.**<device>**

Network device, e.g. eth1

**Optional:****<mode>**

0 = Interrupt mode | 1 = Polling mode

**Optional:****--nommaprx**

Disable PACKET\_MMAP for receive

**Optional:****--promiscuousmode**

Enable promiscuous mode

**-sockxdp** <instance> <mode> [<queue>] [xdpmode]**Hardware:** Hardware independent, only available for Linux.**<device>**

Network device, e.g. eth1

**<mode>**

0 = Interrupt mode | 1 = Polling mode

**Optional:****<queue>**

Queue Id, e.g. 0

**Optional:****<xdpmode>**

1 = SKB | 2 = DRV | 3 = HW | 4 = DRV\_ZEROCOPY

**-stm32eth** <instance> <mode>**Hardware:** STM32H7 Ethernet**<instance>**

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

**<mode>**

1 = Polling mode

**-tap** <instance> <mode> <adapterName>

### OpenVPN's Windows TAP

```

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

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

<adapterName>
    Adapter name

-winpcap <ipAddress> <mode>

```

**Hardware: Hardware independent, only available for Windows.**

```

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

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

```

## 2.5 Compiling the EcMasterDemo

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

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

### 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:**

```

<InstallPath>/Bin
<InstallPath>/Doc
<InstallPath>/SDK
<InstallPath>/SDK/INC
<InstallPath>/SDK/LIB
<InstallPath>/SDK/FILES
<InstallPath>/Sources/Common

```

#### /Bin

Executables containing the master stack

#### /Doc

Documentation

#### /Examples

One or more example applications using a predefined EtherCAT-configuration. It is easily adaptable to different configurations using an appropriate EtherCAT configuration XML file.

#### /SDK

EtherCAT Software Development Kit containing libraries and header files to build C/C++-applications.

#### /SDK/INC:

Header files to be included with the application

**/SDK/LIB:**

Libraries to be linked with the application

**/SDK/FILES:**

Additional files for platform integration (e.g. Windows CE registry files)

**/Sources/Common:**

Shared .cpp-files

## 2.5.2 Include search path

The header files are located in the following directories:

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

## 2.5.3 Libraries

The libraries located in the following directories:

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

## 3 Software Integration

### 3.1 Network Timing

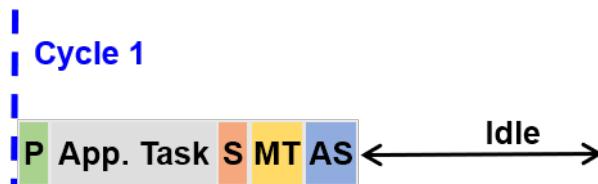
Interaction between application and EtherCAT network

EC-Master has no internal tasks, the operation is fully controlled by the user's application. The benefits of this design are:

- No synchronization issues between application and EC-Master
- Consistent process data without using any locks
- Various network timings driven by the application possible
- Cyclic part may run within Interrupt Service Routine (ISR)
- Easy to integrate

From the application perspective, EC-Master behaves like a driver that is controlled by the `emExecJob()` function with additional parameters, so-called `EC_T_USER_JOB`.

Typical sequence of `EC_T_USER_JOB` for `emExecJob()` to be called cyclically by the application:



**P**

#### Refresh Inputs

`EC_T_USER_JOB::eUsrJob_ProcessAllRxFrames`: Process all received frames

**S**

#### Write Outputs

`EC_T_USER_JOB::eUsrJob_SendAllCycFrames`: Send cyclic frames to update process output data.

**MT**

#### Administration

`EC_T_USER_JOB::eUsrJob_MasterTimer`: Trigger master and slave state machines.

**AS**

#### Send acyclic datagrams/commands

`EC_T_USER_JOB::eUsrJob_SendAcycFrames`: Transmit pending acyclic frame(s).

When a process data update is initiated by calling `emExecJob(eUsrJob_ProcessAllRxFrames)` new input data are read from the received frames and copied into the process data image. After the function returns the application can process the inputs, calculate the outputs and update the values in the process image. With calling `emExecJob(eUsrJob_SendAllCycFrames)` the output data are read from the process data image and stored in Ethernet/EtherCAT frames prior to sending them to the Real-time Ethernet Driver. 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 image and calling `emEx-ecJob(eUsrJob_SendAllCycFrames)` no further output process data synchronization is necessary. The application is responsible to (cyclically) calling the function `emExecJob()` with the appropriate parameters.

EtherCAT frames are divided into two categories:

### 1. Cyclic frames

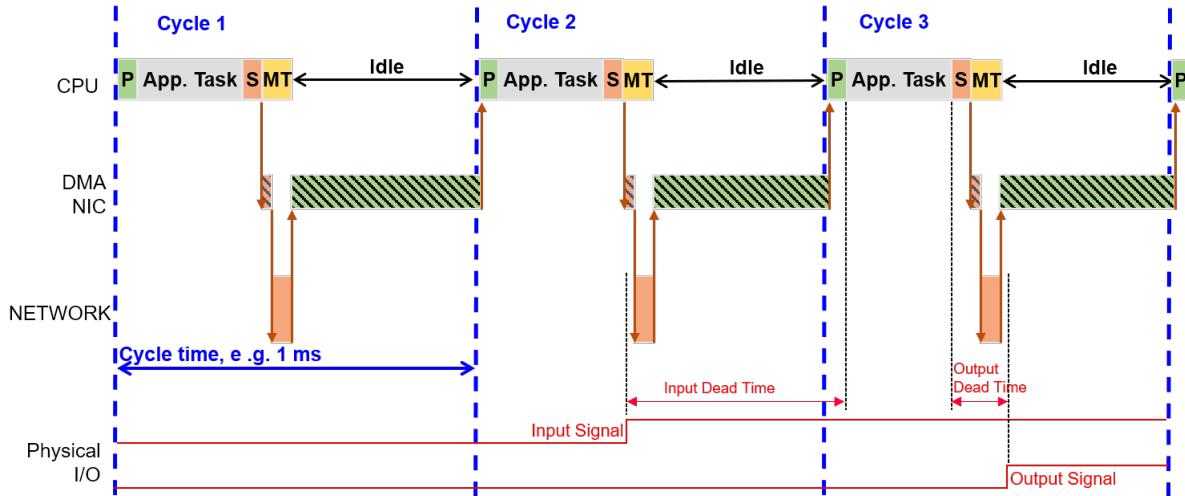
- Contain process output and input data
- Distributed Clocks (DC): Contain datagram to distribute network time
- Typically sent by master in every cycle
- Defined by the configuration tool (which data to read and to write)

### 2. Acyclic frames

- Asynchronous, event triggered communication
- Mailbox communication (CoE, FoE, EoE)
- Status requests (e. g. read slave state information)
- Raw EtherCAT datagrams requested by application

### 3.1.1 Standard Timing: Short output dead time

#### Cyclic frames



Application has to perform:

```

/* Job P: Process data are saved in the process data image */
emExecJob(dwInstanceId, eUsrJob_ProcessAllRxFrames, &oJobParms);

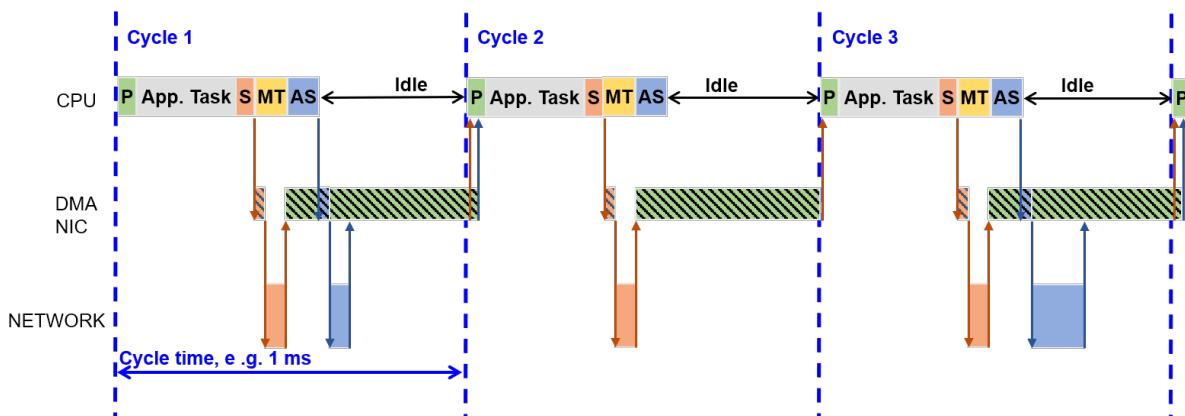
/* App. Task */

/* Job S: Send updated process data.
   Outputs are updated in slaves and input data is collected to be present for the
   next cycle.
   The process data image is saved during eUsrJob_ProcessAllRxFrames */
emExecJob(dwInstanceId, eUsrJob_SendAllCycFrames, EC_NULL);

/* Job MT: Trigger master state machines.
   Required to perform any status changes or internal administration tasks */
emExecJob(dwInstanceId, eUsrJob_MasterTimer, EC_NULL);

```

#### Cyclic and acyclic frames



Application has to perform:

```

/* Job P: Process data are saved in the process data image */
emExecJob(dwInstanceId, eUsrJob_ProcessAllRxFrames, &oJobParms);

```

(continues on next page)

(continued from previous page)

```

/* App. Task */

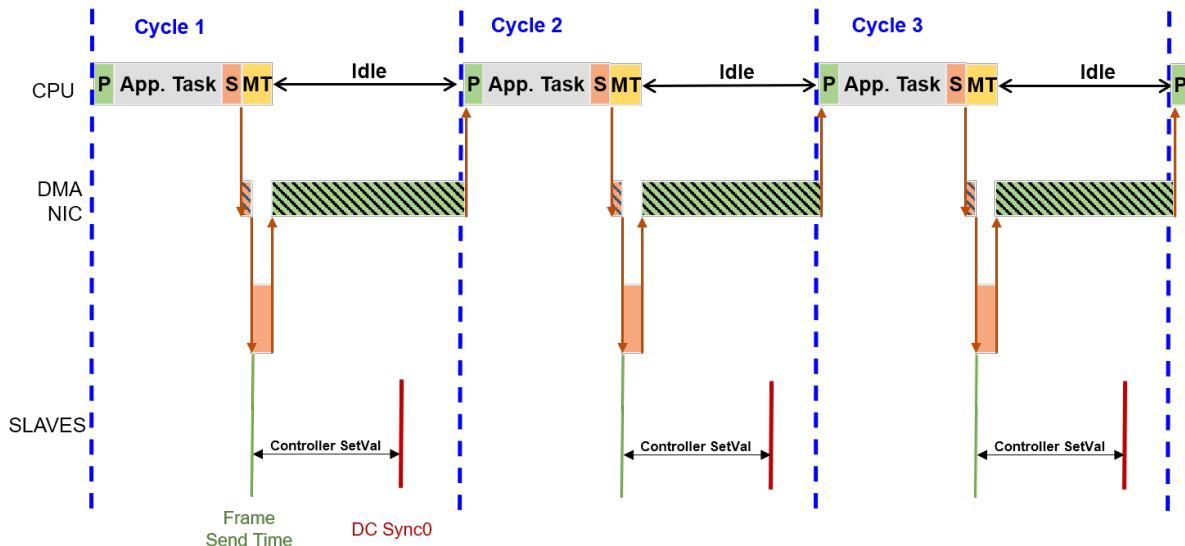
/* Job S: Send updated process data.
   Outputs are updated in slaves and input data is collected to be present for the
   next cycle.
   The process data image is saved during eUsrJob_ProcessAllRxFrames */
emExecJob(dwInstanceId, eUsrJob_SendAllCycFrames, EC_NULL);

/* Job MT: Trigger master state machines.
   Required to perform any status changes or internal administration tasks */
emExecJob(dwInstanceId, eUsrJob_MasterTimer, EC_NULL);

/* Job AS: Transmission of the acyclic commands from the queue.
   These may have been queued by the application or by the internal administration
   task (eUsrJob_MasterTimer) */
emExecJob(dwInstanceId, eUsrJob_SendAcycFrames, EC_NULL);

```

### Cyclic frames with DC



Application has to perform:

```

/* Job P: Process data are saved in the process data image */
emExecJob(dwInstanceId, eUsrJob_ProcessAllRxFrames, &oJobParms);

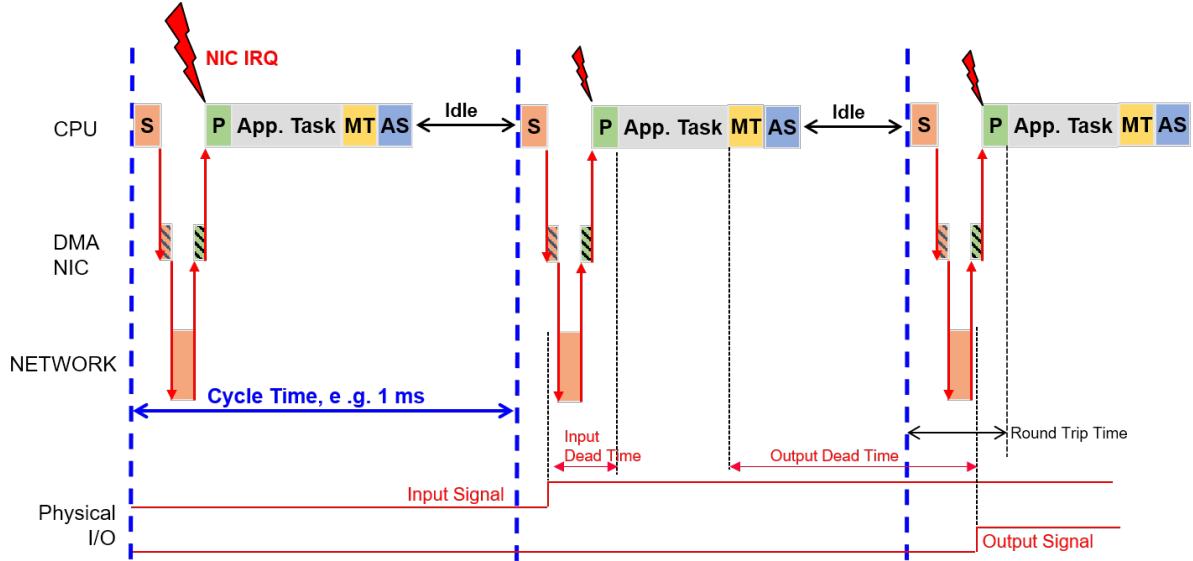
/* App. Task */

/* Job S: Send updated process data.
   Outputs are updated in slaves and input data is collected to be present for the
   next cycle.
   The process data image is saved during eUsrJob_ProcessAllRxFrames */
emExecJob(dwInstanceId, eUsrJob_SendAllCycFrames, EC_NULL);

/* Job MT: Trigger master state machines.
   Required to perform any status changes or internal administration tasks */
emExecJob(dwInstanceId, eUsrJob_MasterTimer, EC_NULL);

```

### 3.1.2 Alternative Timing: Short Input dead time



Application has to perform during startup:

```
emInitMaster(dwInstanceId, &oInitMasterParms);

/* create event for "cyclic frame received" and register RX callback function */
{
    EC_T_CYCFRAME_RX_CBDESC oCycFrameRxCallbackDesc;

    S_pvCycFrameReceivedEvent = OsCreateEvent();

    /* setup callback function which is called after RX */
    OsMemset(&oCycFrameRxCallbackDesc, 0, sizeof(EC_T_CYCFRAME_RX_CBDESC));
    oCycFrameRxCallbackDesc.pfnCallback = CycFrameReceivedCallback;
    oCycFrameRxCallbackDesc.pCallbackContext = S_pvCycFrameReceivedEvent;

    emIoCtl(dwInstanceId, EC_IOCTL_REGISTER_CYCFRAME_RX_CB, &oCycFrameRxCallbackDesc,
    ↪ sizeof(EC_T_CYCFRAME_RX_CBDESC), EC_NULL, 0, EC_NULL);
}

/* create cyclic process data Thread */
S_pvtJobThread = OsCreateThread((EC_T_CHAR*) "EcMasterJobTask", EcMasterJobTask,
    ↪ CpuSet,
    JOBS_THREAD_PRIO, JOBS_THREAD_STACKSIZE, (EC_T_VOID*) pAppContext);
```

Application has to perform inside job task:

```
/* Job S: Send updated process data.
   Outputs are updated in slaves and input data is collected to be present for the
   ↪ current cycle.
   The process data image is saved after receiving the response frame within the
   ↪ interrupt service thread */
emExecJob(dwInstanceId, eUsrJob_SendAllCycFrames, EC_NULL);

/* wait until cyclic frame is received */
OsWaitForEvent(S_pvCycFrameReceivedEvent, dwCycleTime);

/* App. Task */

/* Job MT: Trigger master state machines.
```

(continues on next page)

(continued from previous page)

```
Required to perform any status changes or internal administration tasks */
emExecJob(dwInstanceId, eUsrJob_MasterTimer, EC_NULL);

/* Job AS: Transmission of the acyclic commands from the queue.
   These may have been queued by the application or by the internal administration
   task (eUsrJob_MasterTimer) */
emExecJob(dwInstanceId, eUsrJob_SendAcycFrames, EC_NULL);
```

For closer details find an example project Examples/EcMasterDemoSyncSm

## 3.2 Example application

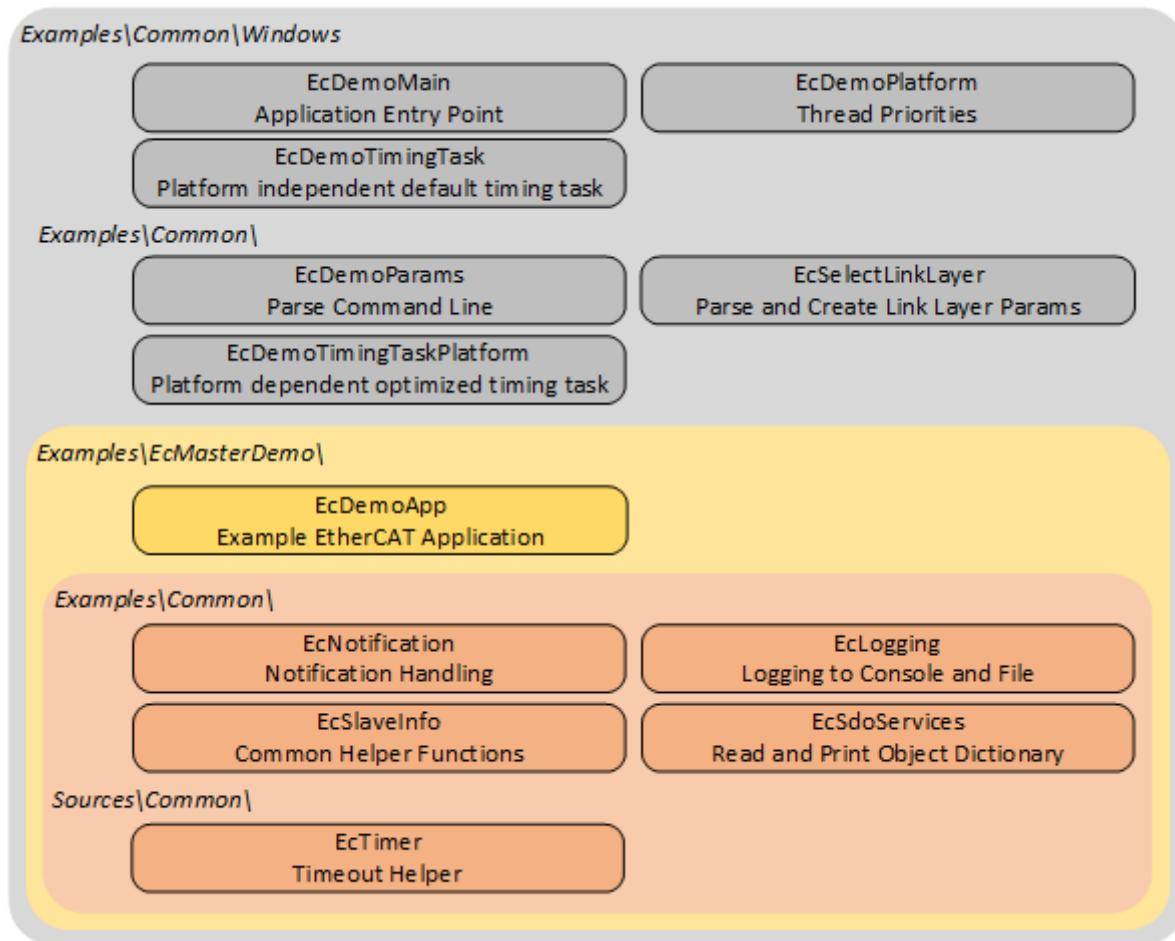
The example application EcMasterDemo handles the following tasks:

- Showing basic EtherCAT communication
- Master stack initialization
- Start (set all slaves into OPERATIONAL state)
- “Out of the box” solution for different operating systems, see *Platform and Operating Systems (OS)*
- 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:**

- ecmaster0.log all messages
- error0.log application error messages (logged via LogError function)

### 3.2.1 File reference

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



#### **EcDemoMain.cpp**

Entry point for the different operating systems

#### **EcDemoPlatform.h**

Operating system specific settings (taskpriorities, timer settings)

#### **EcDemoTimingTask.h/.cpp**

Operating system independent default timing task implementation (base class)

#### **EcDemoTimingTaskPlatform.h/.cpp**

Operating system dependent performance increasing overrides of EcDemoTimingTask

#### **EcDemoApp.cpp**

Initialize, start and terminate the EtherCAT master

#### **EcDemoApp.h**

Application specific settings for EcDemoApp

#### **EcDemoParams.cpp**

Parsing of command line parameters

#### **EcDemoParams.h**

Basic configuration structs and parameters (EtherCAT master parameter)

#### **EcSelectLinkLayer.cpp**

Common Functions which abstract the command line parsing into Real-time Ethernet Driver parameters

**EcNotification.cpp**

Slave monitoring and error detection (function emNotify() )

**EcSdoServices.cpp**

CoE object dictionary example

**EcSlaveInfo.cpp**

Slave information services (bus scan, slave properties, getting information of slaves connected to the EtherCAT bus)

**EcLogging.cpp**

Message logging functions

**EcTimer.cpp**

Start and monitor timeouts

### 3.2.2 Master lifecycle

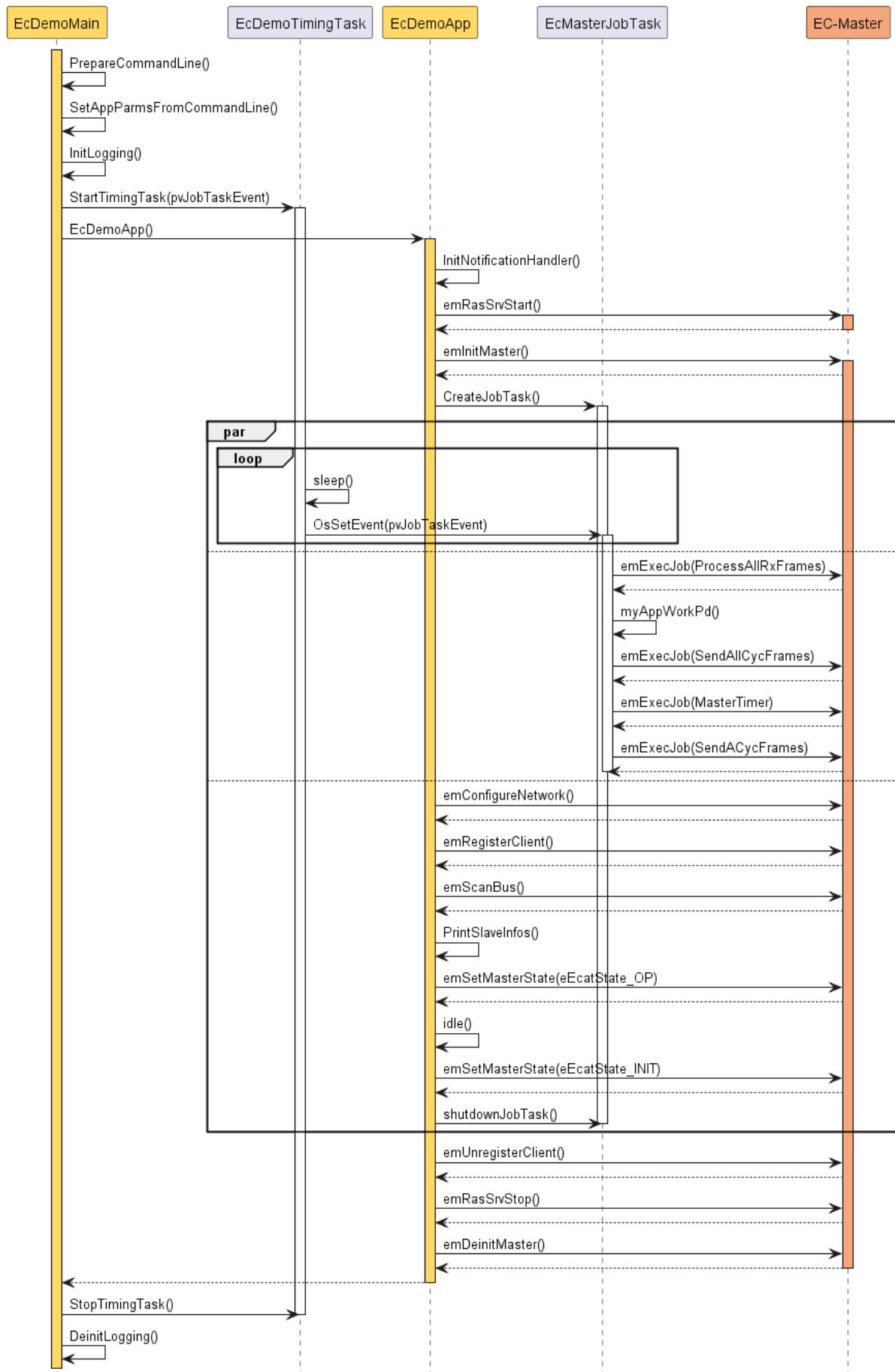
This chapter provides a brief overview of starting and stopping the EC-Master. Basically the operation of the EC-Master is wrapped between the functions:

- *emInitMaster()*
- *emSetMasterState()*

and

- *emDeinitMaster()*

The EC-Master is made ready for operation and started with the first two functions mentioned. During this preparation, a thread is set up and started that handles all the cyclic tasks of the EC-Master. The last function stops the EC-Master and clears the memory. The following sequence diagram gives an overview of the complete life cycle.



A more detailed description of the functions:

#### **EcDemoMain()**

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

#### **EcDemoApp()**

Demo application. The function takes care of starting and stopping the master and all related tasks. In between, the function runs idle, while all relevant work is done by the EcMasterJobTask().

#### **EcMasterJobTask()**

Thread that does the necessary periodic work. Very important here is myApp-Workpd() between `EC_T_USER_JOB::eUsrJob_ProcessAllRxFrames` and `EC_T_USER_JOB::eUsrJob_SendAllCycFrames`. Application-specific working on process data, which must be synchronous with the bus cycle, can be carried out here.

#### **EcDemoTimingTask()**

Timing Thread. This thread sets the timing event that triggers the EcMasterJobTask for the next cycle.

#### **emInitMaster()**

EC-Master API function: Prepare the master for operation and set operational parameters, e.g. used Real-time Ethernet Driver, buffer sizes, maximum number of slaves, ....

#### **emConfigureNetwork()**

EC-Master API function: Loads the configuration from the ENI (XML file).

#### **emRegisterClient()**

EC-Master API function: Register the application as a client at the EC-Master to receive event notifications.

#### **emSetMasterState()**

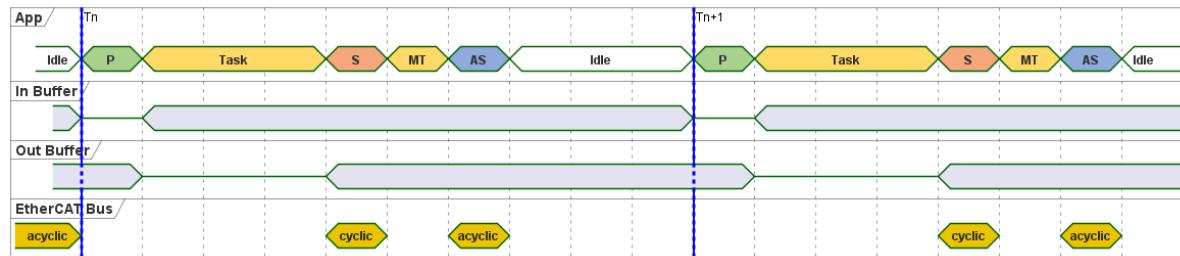
EC-Master API function: Startup the EtherCAT master and switch the bus to the different states from INIT to OPERATIONAL.

#### **emDeinitMaster()**

EC-Master API function: Clean up.

### 3.2.3 Synchronization

This chapter relates the tasks or functions that run in the `EcMasterJobTask()` to the timing and communication on the EtherCAT bus.



**App** Shown are the tasks/jobs P, Task, S, MT and AS which must be done by the application every single cycle. The details of the individual tasks are described below. Once all tasks are finished, the application is idle for the next cycle.

#### 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 P.

#### 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 Task.

#### 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 `EcDemoApp.cpp` the tasks/jobs shown in the timing-diagram are managed and scheduled by `EcMasterJobTask()`.

#### Job P

The job `EC_T_USER_JOB::eUsrJob_ProcessAllRxFrames` handles the frames and data received from previous bus activity, including both cyclic and acyclic frames. These received frames are analyzed for new input data, and the local process image is updated accordingly. During this update, the input data section of the process image is invalid.

#### Task

The function `myAppWorkpd()` allows application-specific working on process data. In this function, the application can use updated input information from Job P, perform calculations and manipulations, and write new data to the output section of the process image.

#### Job S

The job `EC_T_USER_JOB::eUsrJob_SendAllCycFrames` initiates the transmission of all cyclic frames on the EtherCAT bus.

#### Job MT

The job `EC_T_USER_JOB::eUsrJob_MasterTimer` serves an administrative role, primarily managing the timeout timers. During these calls, there is no interaction with the process image, nor do they trigger any bus traffic. It is not essential to run this function with every bus cycle, particularly in systems with cycle times shorter than 1 ms. However, it is recommended to run this function at a 1 ms interval.

#### Job AS

The job `EC_T_USER_JOB::eUsrJob_SendAcycFrames` schedules the transmission of acyclic frames.

#### Idle

Currently implemented to wait for the next cycle, which is triggered by the timing event.

### 3.2.4 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
- ...

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

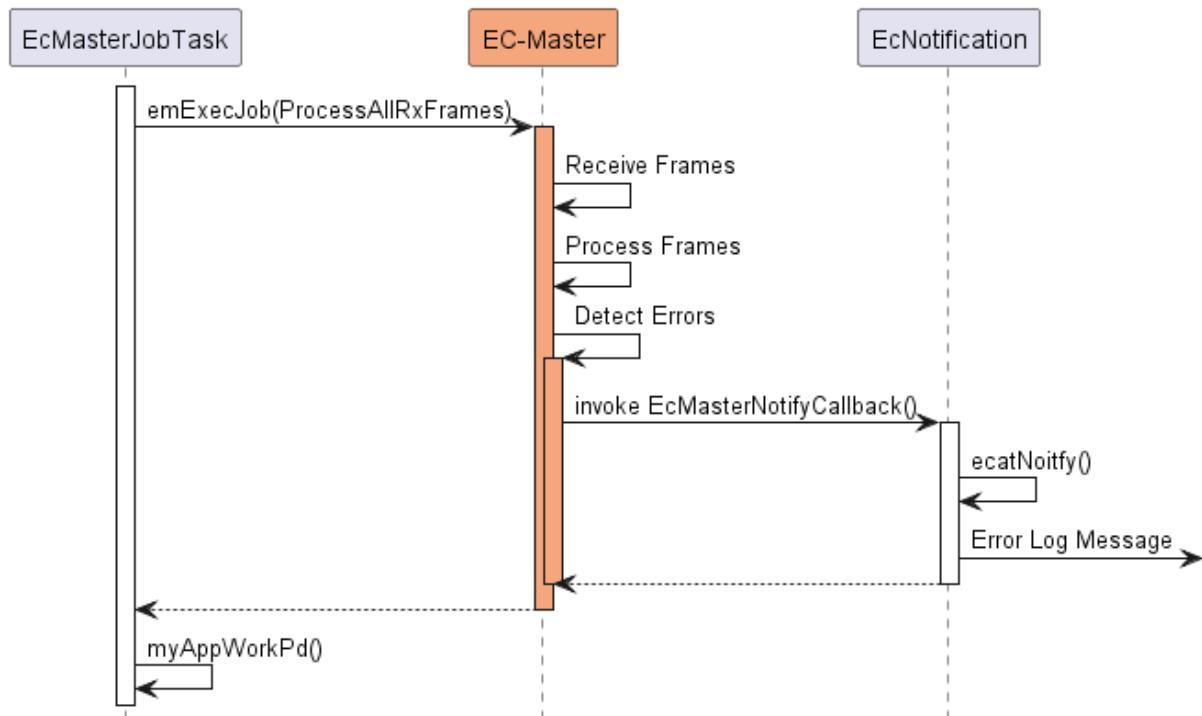
```
EC_T_DWORD emRegisterClient(EC_T_DWORD dwInstanceID, EC_PF_NOTIFY pfnNotify, EC_T_VOID
*pCallerData, EC_T_REGISTERRESULTS *pRegResults)
```

In case of the EcMasterDemo the class CEmNotification is provided. It implements the complete framework to catch and handle the EC-Master notifications. The class is instantiated once and registered at the EC-Master with the call `emRegisterClient()` shown above. The class implements the method `ecatNotify()` as major entry point (or callback function) for events.

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

#### Direct notification handling

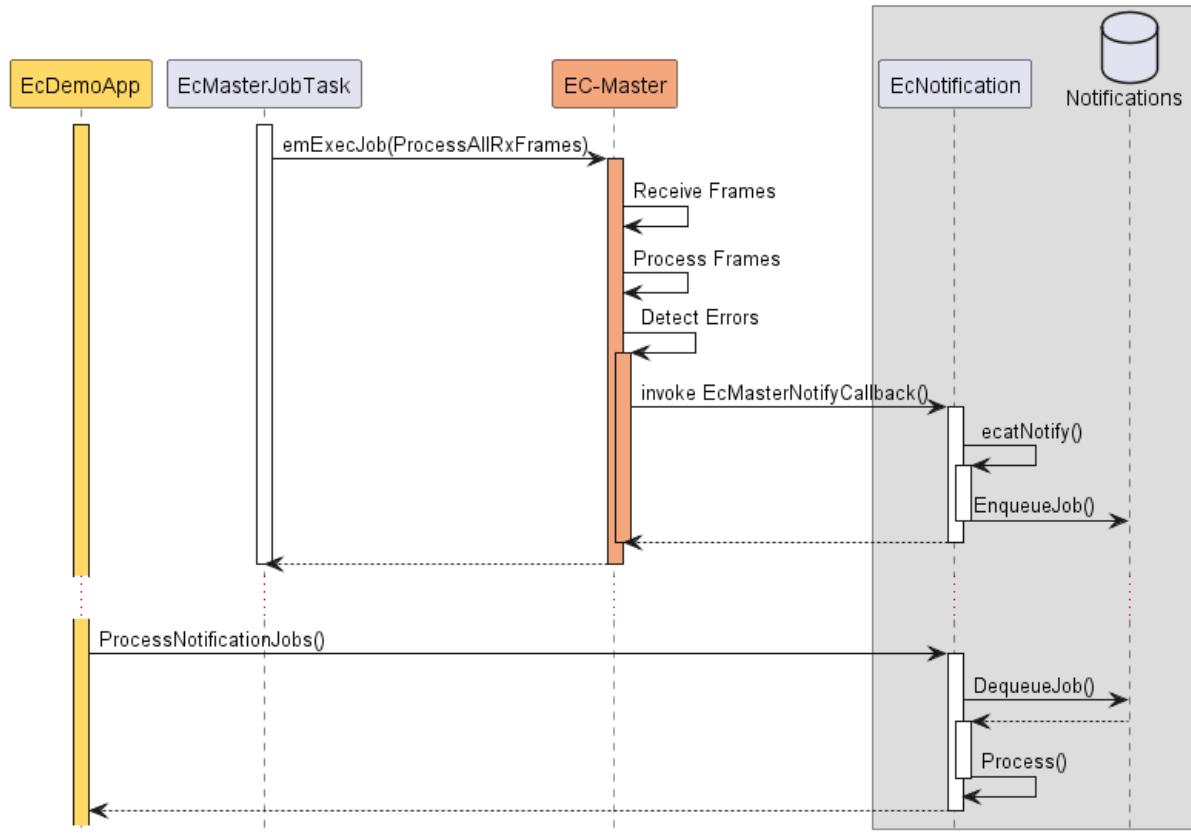
Smaller events can be handled directly in the context in which they are detected. A possible example of such an event is the detection of a false work counter (WKC). The procedure is as follows:



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

## Postponed notification handling

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



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

---

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

---

### 3.2.5 Logging

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

---

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

---

#### Parameters

The verbosity of the EcMasteDemo is specified as a -v command line parameter. It is used to determine the log level of the application, see EcDemoMain.cpp. For performance reasons the EC-Master automatically filters log messages according to `EC_T_LOG_PARMS::dwLogLevel`. EcLogging.cpp has various parameters beside the log level, like Roll Over setting, log task prio and affinity, log buffer size, etc. See EcMasteDemo for reference.

#### Configure EC-Master logging

The EC-Master logging is configured on initialization, see `EC_T_INIT_MASTER_PARMS::LogParms` in `em-InitMaster()`. The application can provide customized log message handlers of type `EC_PF_LOGMSGHK` if the default handler in EcLogging.cpp does not fulfill the needs of the application.

---

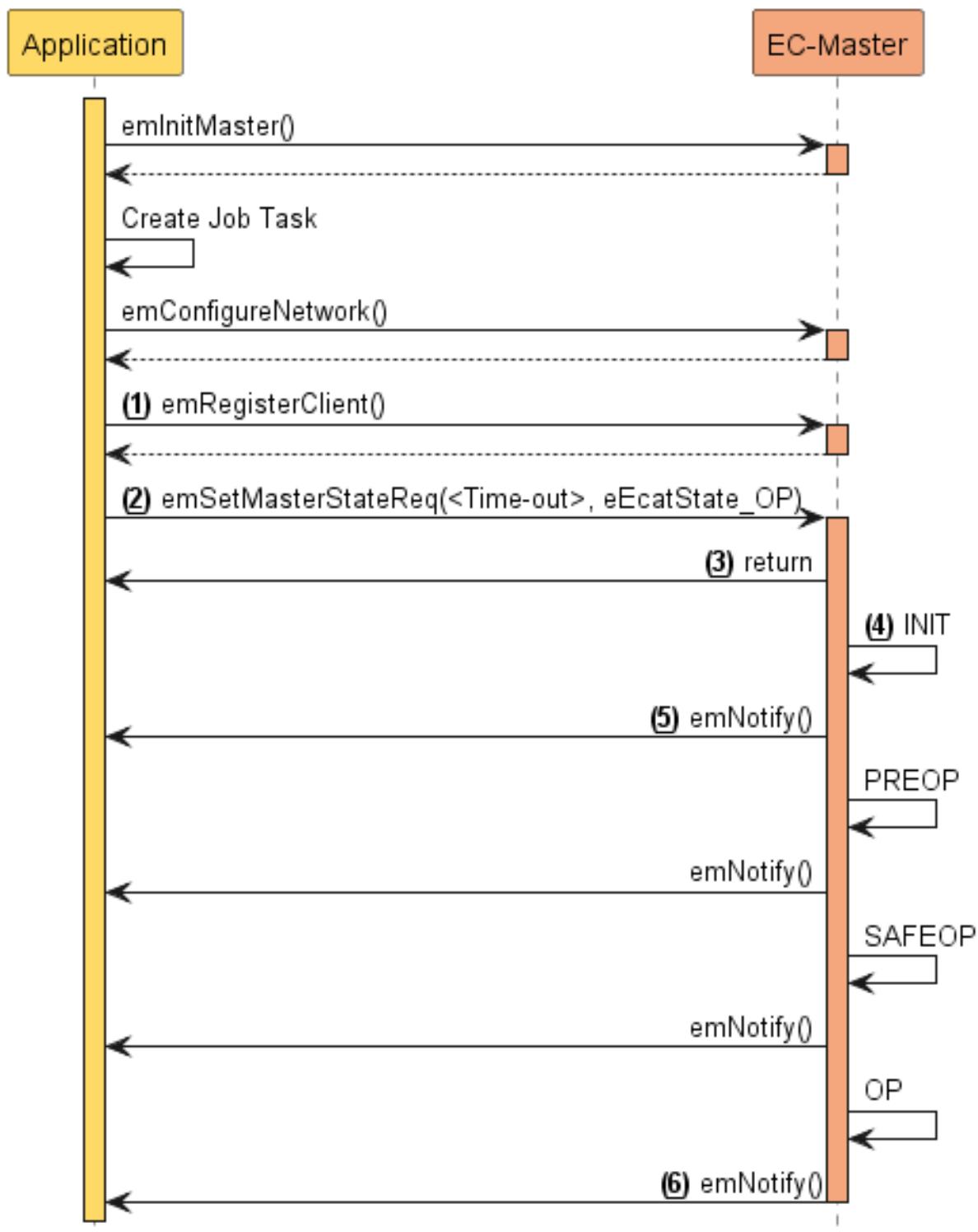
**Note:** The callback is typically called from the context of the EcMasterJobTask and should return as fast as possible.

---

## 3.3 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.

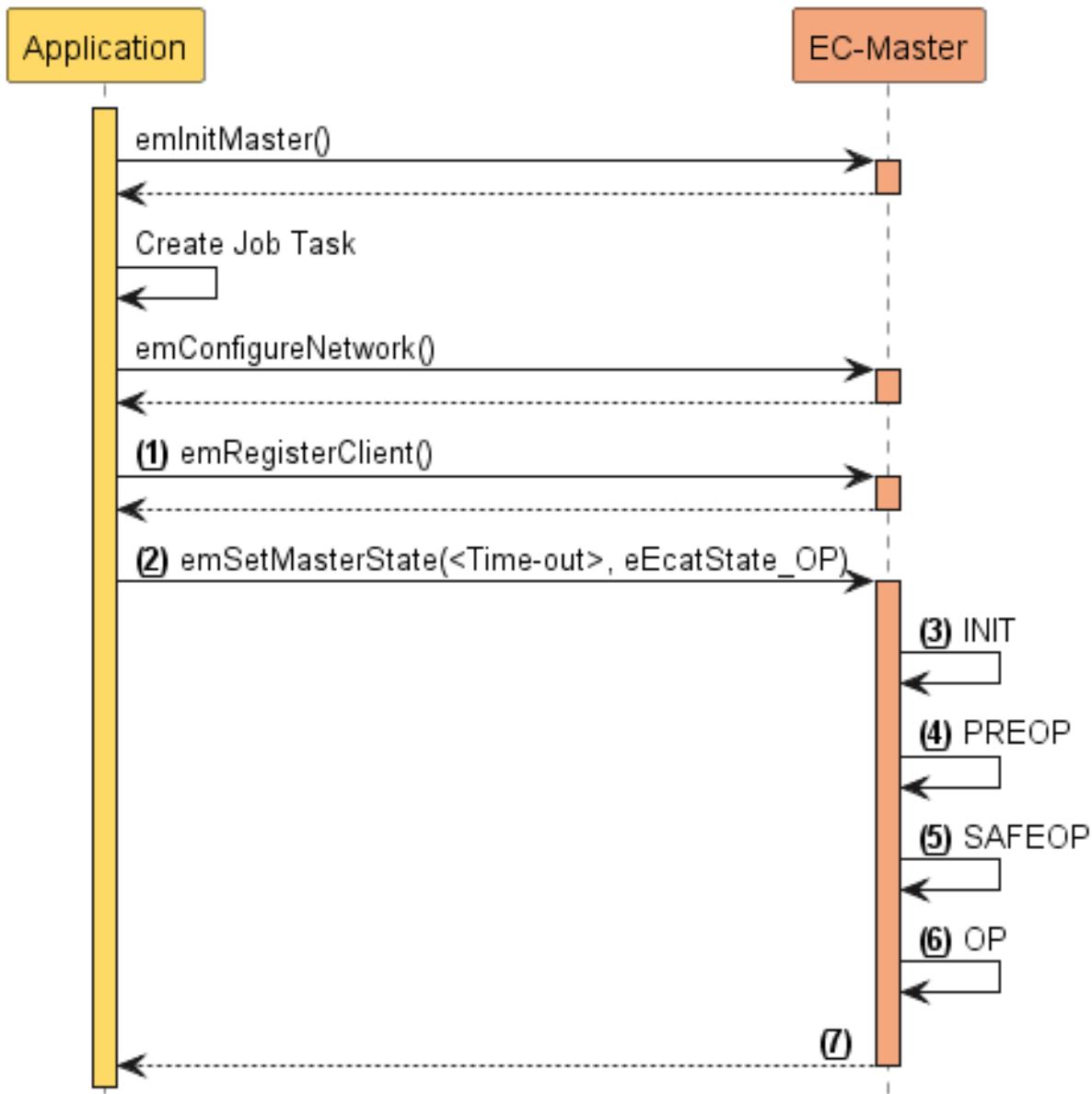
### 3.3.1 Asynchronous (deferred) startup



- Application calls `emInitMaster() (...)`
- Application creates Job Task. See [Master lifecycle](#)
- Application calls `emConfigureMaster() (...)`
- Application calls `emConfigureNetwork() (...)` (See “1” )
- Application calls `emSetMasterStateReq() (...)` with an appropriate timeout value (See “2” )

- Function `emSetMasterStateReq()` (...) returns immediately (See “3”)
- `emSetMasterStateReq()` (...) initiated the master startup procedure (See “4”)
- The master initializes all slaves until all slaves reach OPERATIONAL state
- After every state change the application will be notified (See “5”)
- After reaching the OPERATIONAL state the system is ready (See “6”)

### 3.3.2 Synchronous startup



- Application calls `emInitMaster()` (...)
- Application creates Job Task. See *Master lifecycle*
- Application calls `emConfigureNetwork()` (...)
- Application calls `emRegisterClient()` (...) (See “1”)
- Application calls `emSetMasterState()` (...) with an appropriate timeout value (See “2”)

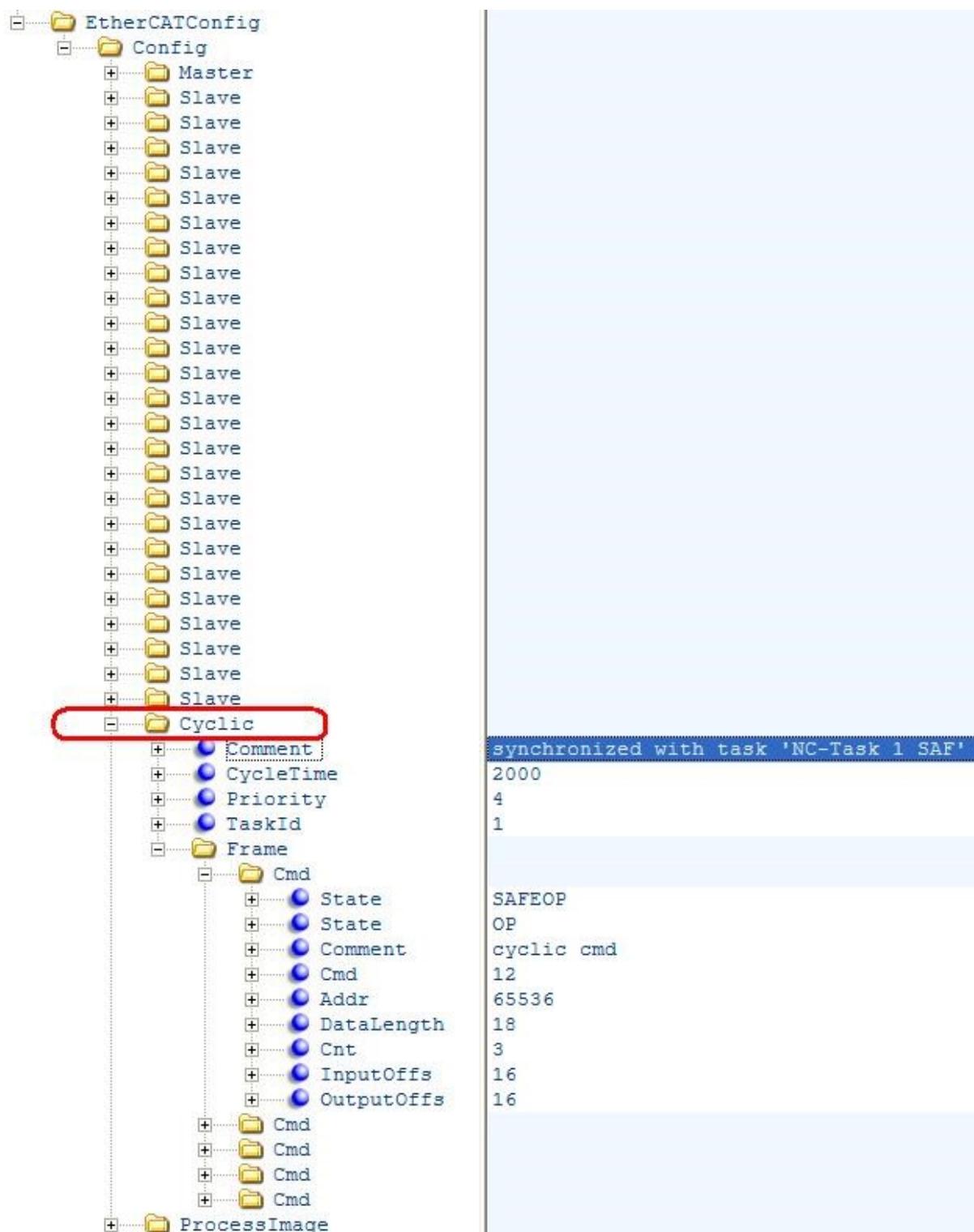
- Inside `emSetMasterState()` (...) the master startup procedure will be initiated (See “3” )
- The application is blocked until the whole startup has finished (See “7” )
- The master initializes all slaves until all slaves reach OPERATIONAL state (See “3-6” )
- After reaching the OPERATIONAL state the system is ready (See “6” )
- `emSetMasterState()` (...) returns (See “7” )

## 3.4 EtherCAT Network Configuration ENI

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.

### 3.4.1 Single cyclic entry configuration

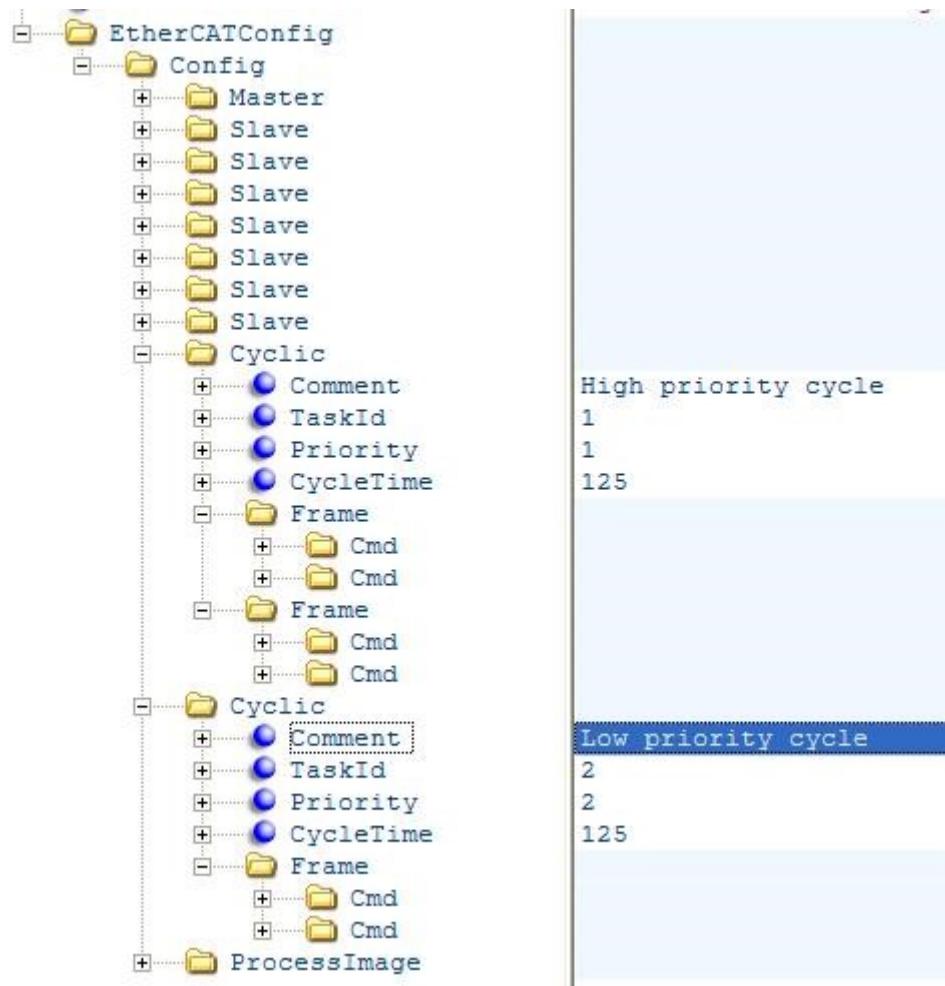
By default there is only a single cyclic entry with one or more cyclic frames:



All process data synchronization modes support this configuration variant.

### 3.4.2 Multiple cyclic entries configuration

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



The application has to use the `EC_T_USER_JOB::eUsrJob_SendCycFramesByTaskId` job call to the master to send the appropriate cyclic frame.

**See also:**

`emExecJob()`

### 3.4.3 Copy Information for Slave-to-Slave communication

It is possible to configure the system to copy input variables to output variables within EC-Master. The copy info declarations of the corresponding received cyclic frame are processed in `emEx-ecJob(eUsrJob_ProcessAllRxFrames)`.

The exchange of process data takes two communication cycles. The duration is necessary if cable redundancy is used or if the WKC of INPUT needs to be checked before changing OUTPUT.

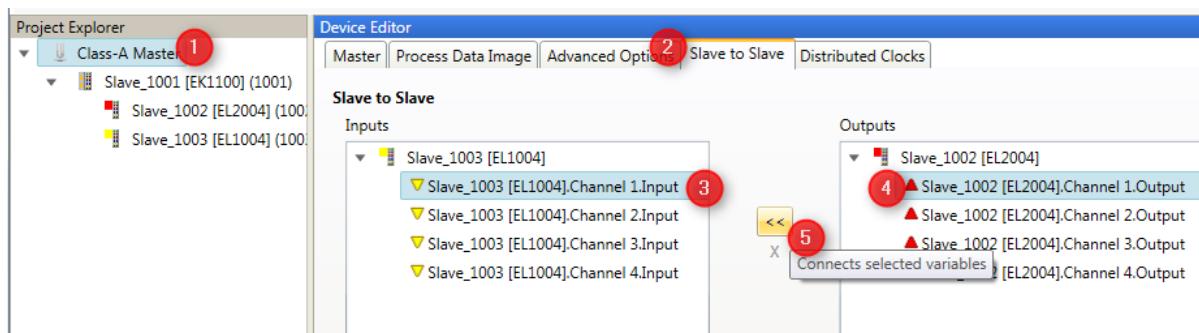
The copy info declarations are located at /EtherCATConfig/Config/Cyclic/Frame/Cmd/CopyInfos in the ENI file.

**See also:**

- *Cyclic cmd WKC validation*
- CopyInfosType in ETG.2100

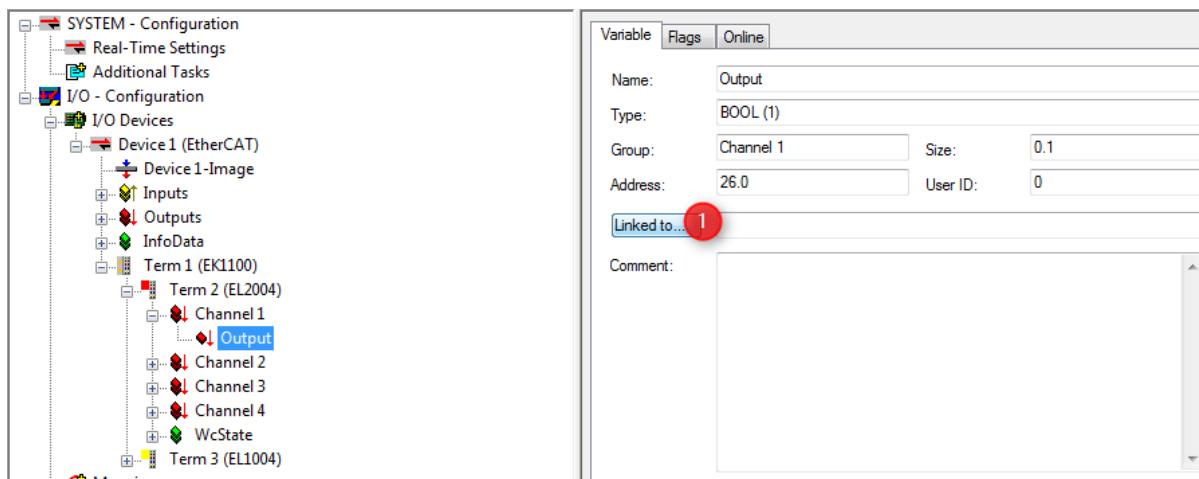
#### Configuration with EC-Engineer

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

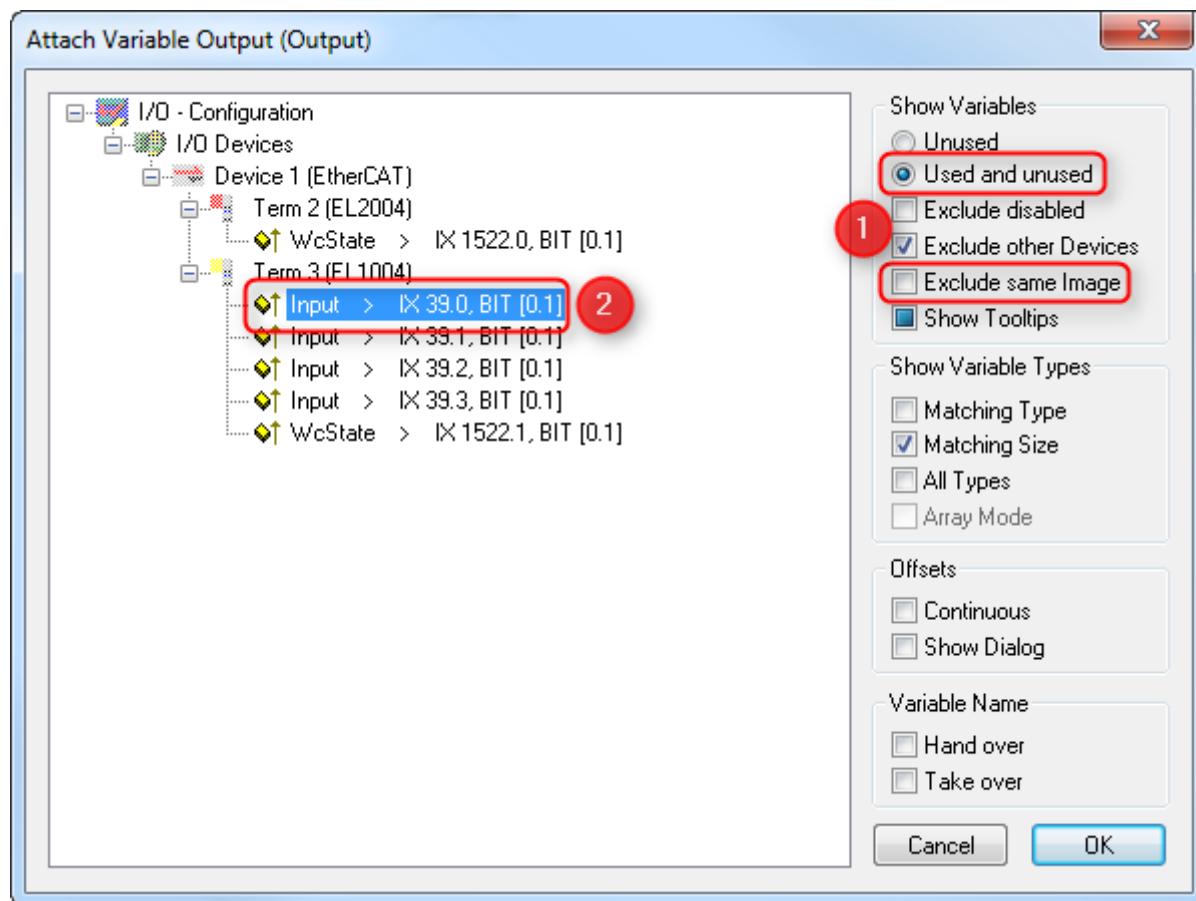


#### Configuration with ET9000

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



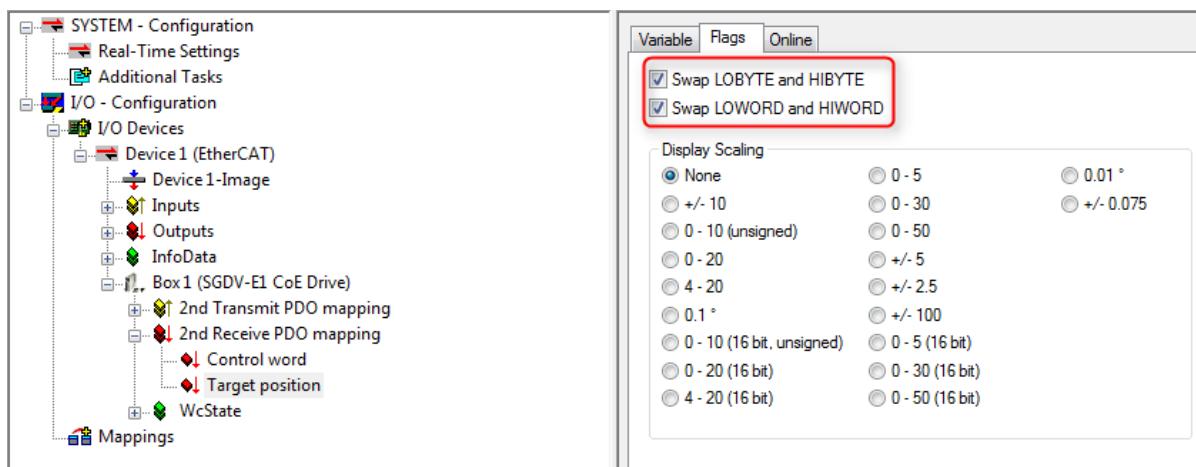
2. Select Input Variable to be attached to the Output Variable:



**Hint:** Copy info declaration processing is independent of WKC values, but updating the INPUT source depends on successful Cyclic cmd WKC validation.

### 3.4.4 Swap bytes of variables 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.

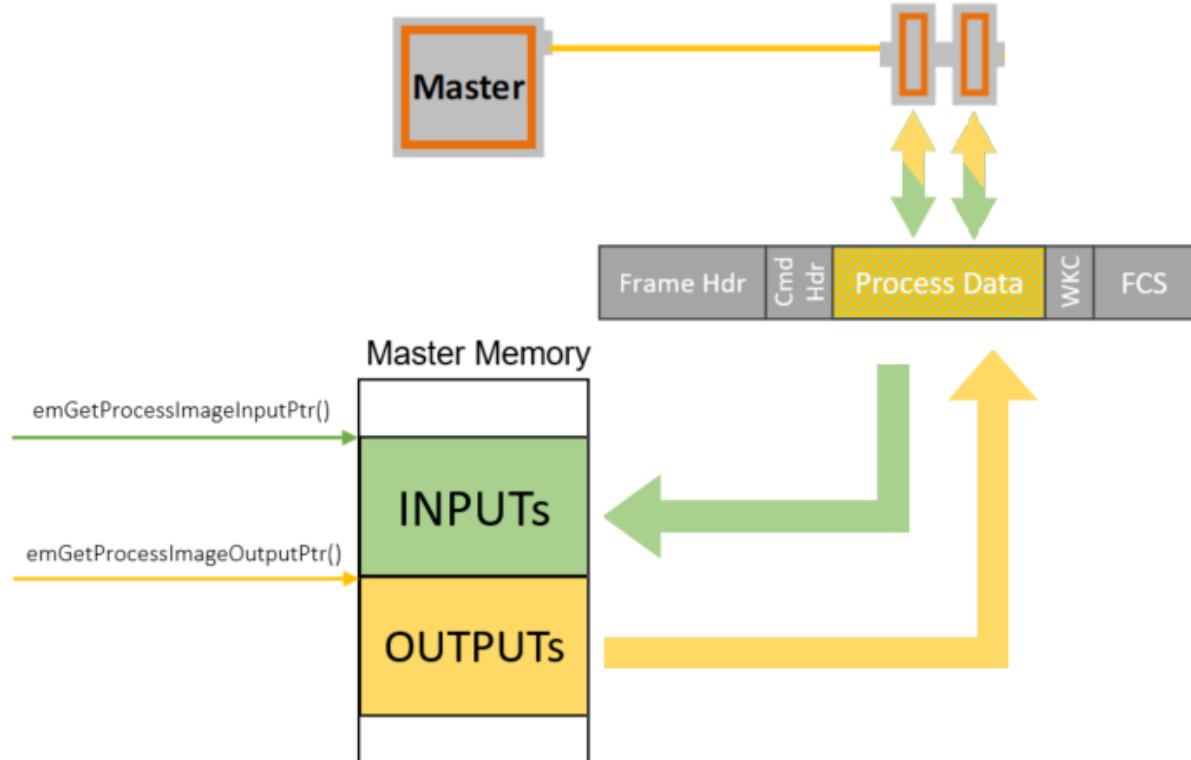
---

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.

### 3.5 Process Data Access

The process data is exchanged as variables between the EtherCAT master and the slaves with EtherCAT commands every cycle.

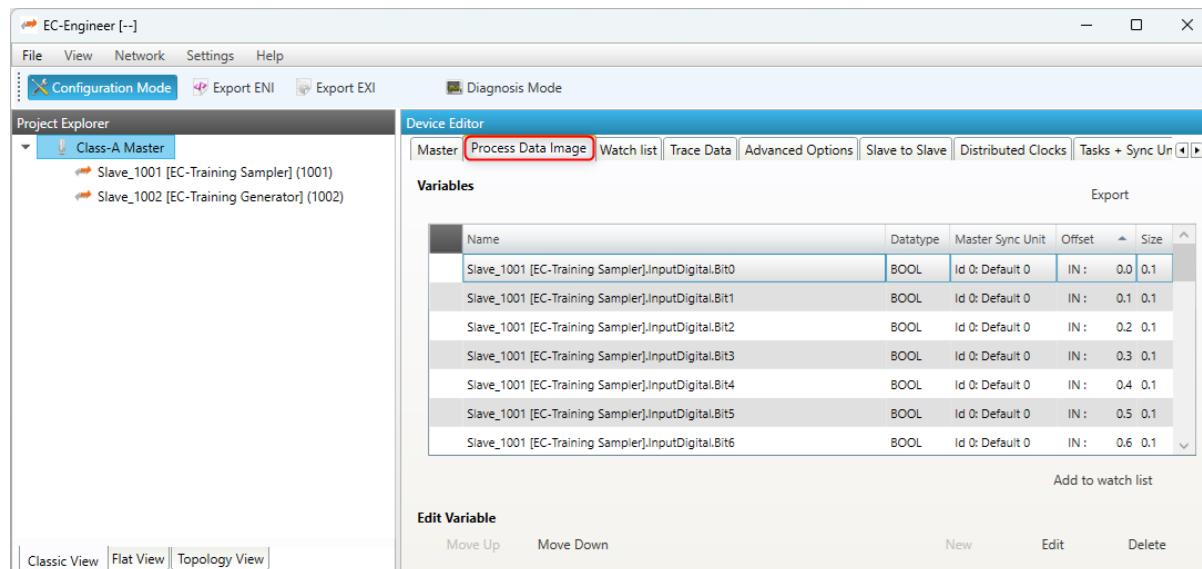
The Master Memory contains the Process Data Image separated in two memory areas, one for input data and another one for output data:



The base addresses of these areas are provided by calling the functions `emGetProcessImageInputPtr()` and `emGetProcessImageOutputPtr()`.

The size of the Process Data Image INPUT/OUTPUT areas as defined in the ENI file under EtherCATConfig/Config/ProcessImage/Inputs/ByteSize and EtherCATConfig/Config/ProcessImage/Outputs/ByteSize is returned by `emRegisterClient()` at `EC_T_REGISTERRESULTS::dwPDOoutSize` and `EC_T_REGISTERRESULTS::dwPDIinSize`.

All INPUT and OUTPUT process data variables are mapped and contained in the Process Data Image:



The information about variables is loaded from the ENI file using `emConfigureNetwork()`:

```
/* load ENI */
const EC_T_CHAR* szFileName = "eni.xml";
dwRes = emConfigureNetwork(dwInstanceId, eCnfType_Filename,
    (EC_T_BYT*)szFileName, (EC_T_DWORD)OsStrlen(szFileName));
```

To get the list of all variables, the example application EcMasterDemo can be started with command line option `-printvars`, see [Running EcMasterDemo](#). It demonstrates the usage of the functions `emGetSlaveInpVarInfoEx()` and `emGetSlaveOutpVarInfoEx()`.

Sizes of EtherCAT variables are given as bit length, not byte length and their offset within the Process Data Image are given as bit offsets, not byte offsets.

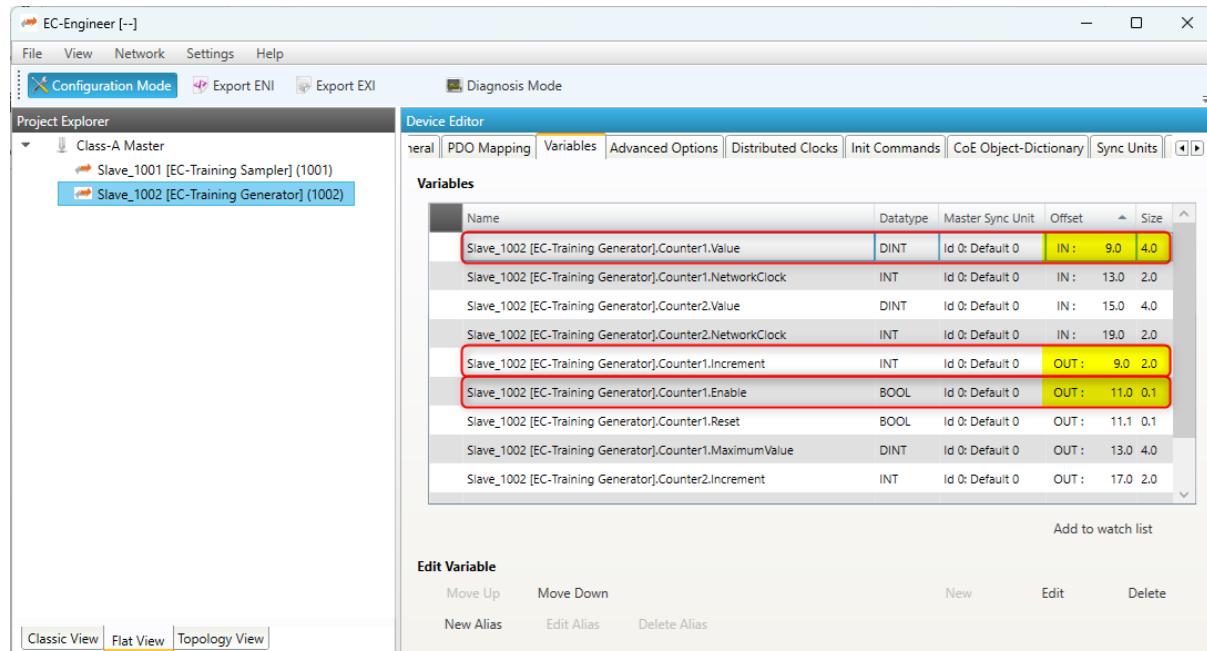
Due to padding bits within the Process Data Objects (PDOs) defined in the EtherCAT slave description (ESI file) and the calculation algorithms within the configuration tool, process data variables are typically starting at a byte boundary.

The structure of the process data image is read from the ENI file. It includes all offsets of all process data variables and does not change until a new configuration is provided. Lookup of variables is therefore only needed once after loading the network configuration (ENI) and is not needed every cycle. In the example program it can be integrated in `myAppPrepare()` at application startup. All variables names are stored in the ENI file under `EtherCATConfig/Config/ProcessImage/ [ Inputs | Outputs ] /Variable`. Application-specific working on process data is supposed to be integrated in `myAppWorkPd()`.

Different ways to lookup informations of variables using the parsed information from the ENI by the EC-Master stack in `myAppPrepare()` and Application-specific working on process data in `myAppWorkPd()` is described in this chapter below.

### 3.5.1 Process data access using hard coded offsets

The configuration tool assigns the offset and size of each variable within the Process Data Image:



The following example demonstrates how to access the variables with hard coded bit offsets in `myAppWorkPd()`.

#### Process data access using hard coded offsets example

```
static EC_T_DWORD myAppWorkpd(T_EC_DEMO_APP_CONTEXT* pAppContext)
{
    EC_T_DWORD dwInstanceId = pAppContext->dwInstanceId;
    EC_T_BYTE* pbyPdIn = emGetProcessImageInputPtr(dwInstanceId);
    EC_T_BYTE* pbyPdOut = emGetProcessImageOutputPtr(dwInstanceId);

    /* Slave_1002 [EC-Training Generator].Counter1.Value (Offset: 9.0) */
    EC_T_SDWORD sdwCounter1_Value = EC_GET_FRM_DWORD(&pbyPdIn[9 /* 9.0 */]);
    EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
        "Counter1.Value: %d\n", sdwCounter1_Value));

    /* Slave_1002 [EC-Training Generator].Counter1.Enable (Offset: 11.0) */
    EC_T_BYTE byCounter1_Enable = 1;
    EC_COPYBIT(&pbyPdOut[11], 0 /* 11.0 */, byCounter1_Enable);

    /* Slave_1002 [EC-Training Generator].Counter1.Increment (Offset: 9.0) */
    EC_T_SWORD swCounter1_Increment = 100;
    EC_SET_FRM_WORD(&pbyPdIn[9 /* 9.0 */], swCounter1_Increment);

    return EC_E_NOERROR;
}
```

**Note:** The offsets are subject to be changed if the ENI file changes. It is strongly recommended to determine the bit offsets of variables on startup as described in the following chapters instead of using hard coded values!

### 3.5.2 Process data access using variable names from ENI

Using the configuration tool, unique names can be assigned to Process Data variables. They are included in the ENI file.

---

**Note:** The variable name parameter to `emFindOutpVarByNameEx()` / `emFindInpVarByNameEx()` must match the variable name in the ENI file. The variable names of each slave are taken from the ESI file and can be changed using the configuration tool.

---

The following example demonstrates how to query the bit offset of a variable by its name from the EC-Master stack using `emFindOutpVarByNameEx()`, `emFindInpVarByNameEx()`.

#### Process data access using variable names from ENI example

```

struct _T_MY_APP_DESC;
typedef struct _T_EC_DEMO_APP_CONTEXT
{
    T_EC_DEMO_APP_PARMS      AppParms;
    EC_T_LOG_PARMS           LogParms;
    EC_T_DWORD                dwInstanceId;
    struct _T_MY_APP_DESC* pMyAppDesc;
} T_EC_DEMO_APP_CONTEXT;

typedef struct _T_MY_APP_DESC
{
    EC_T_PROCESS_VAR_INFO_EX aoProcVarInfo[3];
} T_MY_APP_DESC;

static EC_T_DWORD myAppPrepare(T_EC_DEMO_APP_CONTEXT* pAppContext)
{
    EC_T_DWORD dwRetVal = EC_E_NOERROR;
    EC_T_DWORD dwInstanceId = pAppContext->dwInstanceId;
    T_MY_APP_DESC* pMyAppDesc = pAppContext->pMyAppDesc;
    EC_T_PROCESS_VAR_INFO_EX* aoProcVarInfo = pMyAppDesc->aoProcVarInfo;

    OsMemset(pMyAppDesc->aoProcVarInfo, 0, sizeof(pMyAppDesc->aoProcVarInfo));

    dwRetVal = emFindInpVarByNameEx(dwInstanceId, "Slave_1002 [EC-Training_
→Generator].Counter1.Value", &aoProcVarInfo[0]);
    if (EC_E_NOERROR != dwRetVal)
        return dwRetVal;

    dwRetVal = emFindOutpVarByNameEx(dwInstanceId, "Slave_1002 [EC-Training_
→Generator].Counter1.Enable", &aoProcVarInfo[1]);
    if (EC_E_NOERROR != dwRetVal)
        return dwRetVal;

    dwRetVal = emFindOutpVarByNameEx(dwInstanceId, "Slave_1002 [EC-Training_
→Generator].Counter1.Increment", &aoProcVarInfo[2]);
    if (EC_E_NOERROR != dwRetVal)
        return dwRetVal;

    return EC_E_NOERROR;
}
static EC_T_DWORD myAppWorkpd(T_EC_DEMO_APP_CONTEXT* pAppContext)
{
    EC_T_DWORD dwInstanceId = pAppContext->dwInstanceId;
    T_MY_APP_DESC* pMyAppDesc = pAppContext->pMyAppDesc;
    EC_T_PROCESS_VAR_INFO_EX* aoProcVarInfo = pMyAppDesc->aoProcVarInfo;
}

```

(continues on next page)

(continued from previous page)

```

EC_T_BYTE* pbyPdIn = emGetProcessImageInputPtr(dwInstanceId);
EC_T_BYTE* pbyPdOut = emGetProcessImageOutputPtr(dwInstanceId);

/* Slave_1002 [EC-Training Generator].Counter1.Value (Offset: 9.0) */
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
    "Counter1.Value: %d\n", EC_GET_FRM_DWORD(&pbyPdIn[aoProcVarInfo[0].
    nBitOffs / 8])));

/* Slave_1002 [EC-Training Generator].Counter1.Enable (Offset: 11.0) */
EC_SETBIT(pbyPdOut, aoProcVarInfo[1].nBitOffs);

/* Slave_1002 [EC-Training Generator].Counter1.Increment (Offset: 9.0) */
EC_SET_FRM_WORD(&pbyPdOut[aoProcVarInfo[2].nBitOffs / 8], 100);

return EC_E_NOERROR;
}

```

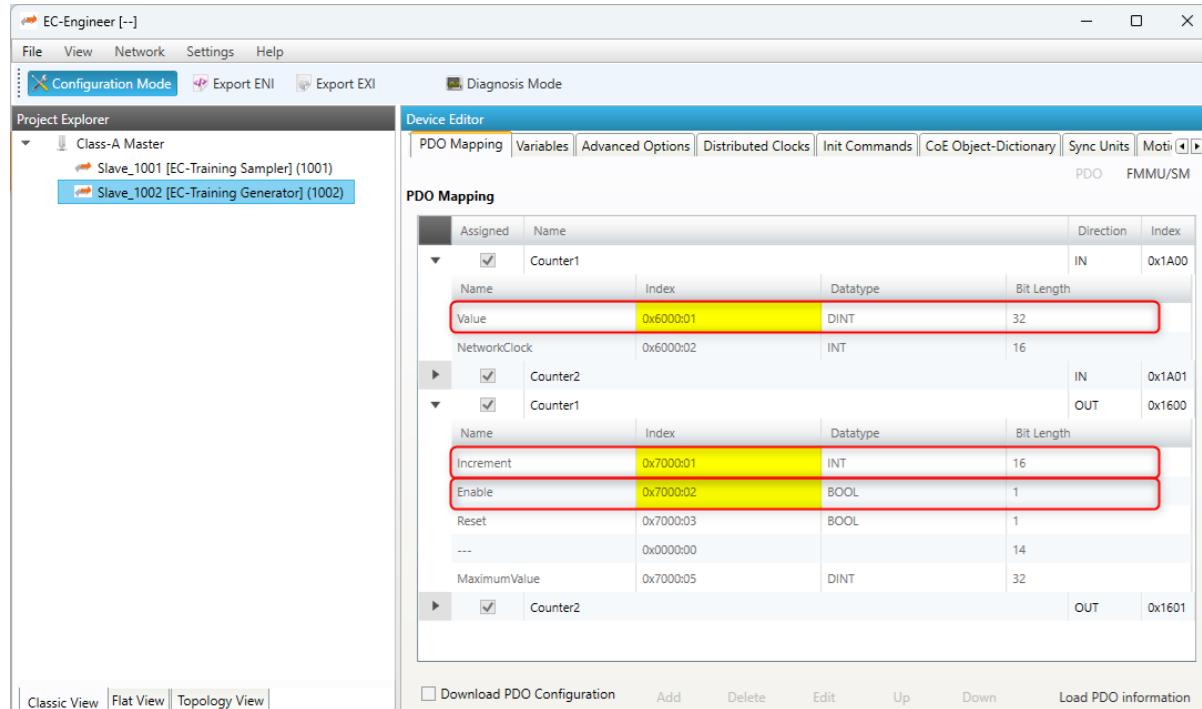
**See also:**

- [emFindOutpVarByNameEx\(\)](#)
- [emFindInpVarByNameEx\(\)](#)

### 3.5.3 Process data access using variable object index from ENI

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

The “PDO Mapping” tab in EC-Engineer shows the Object Index and SubIndex for each variable:



The following example demonstrates how to query the bit offset of a variable by its Process Data variable Object Index and SubIndex from the EC-Master stack using [emGetSlaveOutpVarByObjectEx\(\)](#), [emGetSlaveInpVarByObjectEx\(\)](#).

**Process data access using variable object index from ENI example**

```

struct _T_MY_APP_DESC;
typedef struct _T_EC_DEMO_APP_CONTEXT
{
    T_EC_DEMO_APP_PARMS      AppParms;
    EC_T_LOG_PARMS           LogParms;
    EC_T_DWORD                dwInstanceId;
    struct _T_MY_APP_DESC* pMyAppDesc;
} T_EC_DEMO_APP_CONTEXT;

typedef struct _T_MY_APP_DESC
{
    EC_T_PROCESS_VAR_INFO_EX aoProcVarInfo[3];
} T_MY_APP_DESC;

static EC_T_DWORD myAppPrepare(T_EC_DEMO_APP_CONTEXT* pAppContext)
{
    EC_T_DWORD dwRetVal = EC_E_NOERROR;
    EC_T_DWORD dwInstanceId = pAppContext->dwInstanceId;
    T_MY_APP_DESC* pMyAppDesc = pAppContext->pMyAppDesc;
    EC_T_PROCESS_VAR_INFO_EX* aoProcVarInfo = pMyAppDesc->aoProcVarInfo;

    OsMemset(pMyAppDesc->aoProcVarInfo, 0, sizeof(pMyAppDesc->aoProcVarInfo));

    /* Slave_1002 [EC-Training Generator].Counter1.Value (Object 0x6000:01) */
    dwRetVal = emGetSlaveInpVarByObjectEx(dwInstanceId, EC_TRUE, 1002, 0x6000, 1, &
    ↪aoProcVarInfo[0]);
    if (EC_E_NOERROR != dwRetVal)
        return dwRetVal;

    /* Slave_1002 [EC-Training Generator].Counter1.Enable (Object 0x7000:01) */
    dwRetVal = emGetSlaveOutpVarByObjectEx(dwInstanceId, EC_TRUE, 1002, 0x7000, 1, &
    ↪&aoProcVarInfo[1]);
    if (EC_E_NOERROR != dwRetVal)
        return dwRetVal;

    /* Slave_1002 [EC-Training Generator].Counter1.Increment (Object 0x7000:02) */
    dwRetVal = emGetSlaveOutpVarByObjectEx(dwInstanceId, EC_TRUE, 1002, 0x7000, 2, &
    ↪&aoProcVarInfo[2]);
    if (EC_E_NOERROR != dwRetVal)
        return dwRetVal;

    return EC_E_NOERROR;
}

static EC_T_DWORD myAppWorkpd(T_EC_DEMO_APP_CONTEXT* pAppContext)
{
    EC_T_DWORD dwInstanceId = pAppContext->dwInstanceId;
    T_MY_APP_DESC* pMyAppDesc = pAppContext->pMyAppDesc;
    EC_T_PROCESS_VAR_INFO_EX* aoProcVarInfo = pMyAppDesc->aoProcVarInfo;

    EC_T_BYTE* pbyPdIn = emGetProcessImageInputPtr(dwInstanceId);
    EC_T_BYTE* pbyPdOut = emGetProcessImageOutputPtr(dwInstanceId);

    /* Slave_1002 [EC-Training Generator].Counter1.Value (Object 0x6000:01) */
    EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
        "Counter1.Value: %d\n", EC_GET_FRM_DWORD(&pbyPdIn[aoProcVarInfo[0].
    ↪nBitOffs / 8])));

    /* Slave_1002 [EC-Training Generator].Counter1.Enable (Object 0x7000:01) */
    EC_SETBIT(pbyPdOut, aoProcVarInfo[1].nBitOffs);
}

```

(continues on next page)

(continued from previous page)

```

/* Slave_1002 [EC-Training Generator].Counter1.Increment (Object 0x7000:02) */
EC_SET_FRM_WORD(&pbyPdOut[aoProcVarInfo[2].nBitOffs / 8], 100);

return EC_E_NOERROR;
}

```

**See also:**

- `emGetSlaveOutpVarByObjectEx()`
- `emGetSlaveInpVarByObjectEx()`

### 3.5.4 Process data access using slave station address

Based on the unique station address of a specific slave the base offset of INPUTs and OUTPUTs can be determined using `emGetCfgSlaveInfo()`. The offsets are stored in `EC_T_CFG_SLAVE_INFO::dwPdOffsIn` and `EC_T_CFG_SLAVE_INFO::dwPdOffsOut`:

The following example demonstrates how to query the bit offset of a slave's process data by its station address from the EC-Master stack using `emGetCfgSlaveInfo()`.

#### Process data access using slave station address example

```

static EC_T_DWORD myAppPrepare(T_EC_DEMO_APP_CONTEXT* pAppContext)
{
    EC_T_DWORD dwRetVal      = EC_E_NOERROR;
    EC_T_DWORD dwInstanceId   = pAppContext->dwInstanceId;
    T_MY_APP_DESC* pMyAppDesc = pAppContext->pMyAppDesc;
    OsMemset(pMyAppDesc->aSlaveList, 0, sizeof(pMyAppDesc->aSlaveList));

    dwRetVal = emGetCfgSlaveInfo(dwInstanceId, EC_TRUE, 1001, &pMyAppDesc->
→aSlaveList[0]);
    if (EC_E_NOERROR != dwRetVal)
        return dwRetVal;

    dwRetVal = emGetCfgSlaveInfo(dwInstanceId, EC_TRUE, 1002, &pMyAppDesc->
→aSlaveList[1]);
    if (EC_E_NOERROR != dwRetVal)
        return dwRetVal;

    return EC_E_NOERROR;
}
static EC_T_DWORD myAppWorkpd(T_EC_DEMO_APP_CONTEXT* pAppContext)
{
    EC_T_DWORD dwInstanceId = pAppContext->dwInstanceId;
    T_MY_APP_DESC* pMyAppDesc = pAppContext->pMyAppDesc;

    EC_T_BYTEx pbyPdIn = emGetProcessImageInputPtr(dwInstanceId);
    EC_T_BYTEx pbyPdOut = emGetProcessImageOutputPtr(dwInstanceId);

    /* Slave_1002 [EC-Training Generator].Counter1.Value (first INPUT variable) */
    EC_T_SDWORD* psdwCounter1Value = (EC_T_SDWORD*)&(pbyPdIn[pMyAppDesc->
→aSlaveList[1].dwPdOffsIn / 8]);

    EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
        "Counter1.Value: %d\n", EC_GET_FRM_DWORD(psdwCounter1Value)));

    /* Slave_1002 [EC-Training Generator].Counter1.Increment (first OUTPUT_
variable) */
}

```

(continues on next page)

(continued from previous page)

```

    EC_T_SWORd* pswCounter1Increment = (EC_T_SWORd*) & (pbyPdOut [pMyAppDesc->
    ↪aSlaveList [1].dwPdOffsOut / 8]);
    EC_SET_FRM_WORD (pswCounter1Increment, 100);

    return EC_E_NOERROR;
}

```

**See also:**

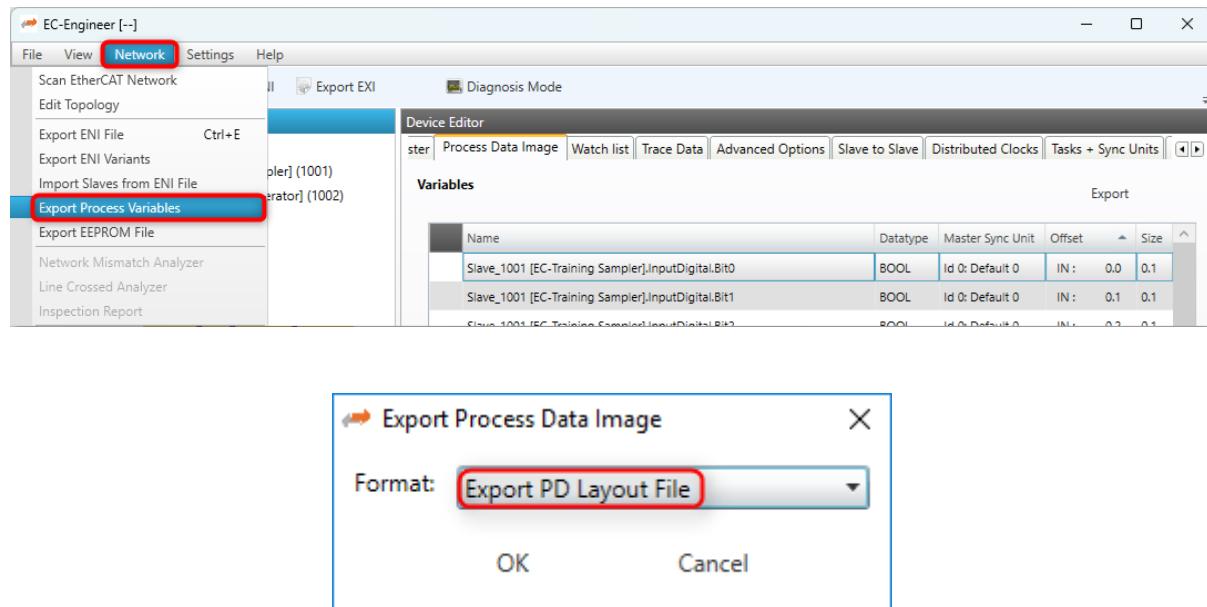
- [emGetCfgSlaveInfo \(\)](#)

**Note:** A slave may have multiple sync units with individual offsets and sizes, see [EC\\_T\\_CFG\\_SLAVE\\_INFO::dwPdOffsIn2](#) [EC\\_T\\_CFG\\_SLAVE\\_INFO::dwPdOffsOut2](#), ... .

**Note:** The example application EcMasterDemo demonstrates the usage of [emGetCfgSlaveInfo \(\)](#) with its process data OUPPUT flashing. See command line option [-flash](#) in [Running EcMasterDemo](#).

### 3.5.5 Process data access using generated PD layout C-header file

The EC-Engineer can export the process variables to a PD layout C-header file via the menu item *Network ▶ Export Process Variables* as shown in the following screenshots:



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

```

#define PDLAYOUT_IN_OFFSET_SLAVE_1002 9
typedef struct _T_PDLAYOUT_IN_SLAVE_1002
{
    EC_T_SDWORD sdwCounter1_Value;           // Slave_1002 ... Counter1.Value
    EC_T_SWORd swCounter1_NetworkClock;     // Slave_1002 ... Counter1.NetworkClock
    EC_T_SDWORD sdwCounter2_Value;           // Slave_1002 ... Counter2.Value
    EC_T_SWORd swCounter2_NetworkClock;     // Slave_1002 ... Counter2.NetworkClock
} EC_PACKED (1) T_PDLAYOUT_IN_SLAVE_1002;
#include EC_PACKED_INCLUDESTOP

```

(continues on next page)

(continued from previous page)

```
#include EC_PACKED_INCLUDESTART(1)
#define PDLAYOUT_OUT_OFFSET_SLAVE_1002 9
typedef struct _T_PDLAYOUT_OUT_SLAVE_1002
{
    EC_T_SWORD    swCounter1_Increment;      // Slave_1002 ... Counter1.Increment
    EC_T_BYTE     byCounter1_Enable : 1;       // Slave_1002 ... Counter1.Enable
    //...
} EC_PACKED(1) T_PDLAYOUT_OUT_SLAVE_1002;
#include EC_PACKED_INCLUDESTOP
```

The following example demonstrates how to access process data variables using a generated PD layout C-header file in `myAppWorkPd()`.

### Process data access using generated PD layout C-header file example

```
#include "pdlayout.h"
static EC_T_DWORD myAppWorkPd(T_EC_DEMO_APP_CONTEXT* pAppContext)
{
    EC_T_DWORD dwInstanceId = pAppContext->dwInstanceId;
    T_PDLAYOUT_IN* poPdIn = (T_PDLAYOUT_IN*)emGetProcessImageInputPtr(dwInstanceId);
    ↵
    T_PDLAYOUT_OUT* poPdOut = (T_PDLAYOUT_OUT*)emGetProcessImageOutputPtr(dwInstanceId);
    ↵

    /* Slave_1002 [EC-Training Generator].Counter1.Value (Offset: 9.0) */
    EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
        "Counter1.Value: %d\n", EC_GET_FRM_DWORD(&poPdIn->sdwSlave_1002_Counter1_
    ↵Value)));

    /* Slave_1002 [EC-Training Generator].Counter1.Enable (Offset: 11.0) */
    poPdOut->bySlave_1002_Counter1_Enable = 1;

    /* Slave_1002 [EC-Training Generator].Counter1.Increment (Offset: 9.0) */
    EC_SET_FRM_WORD(&poPdOut->swSlave_1002_Counter1_Increment, 100);

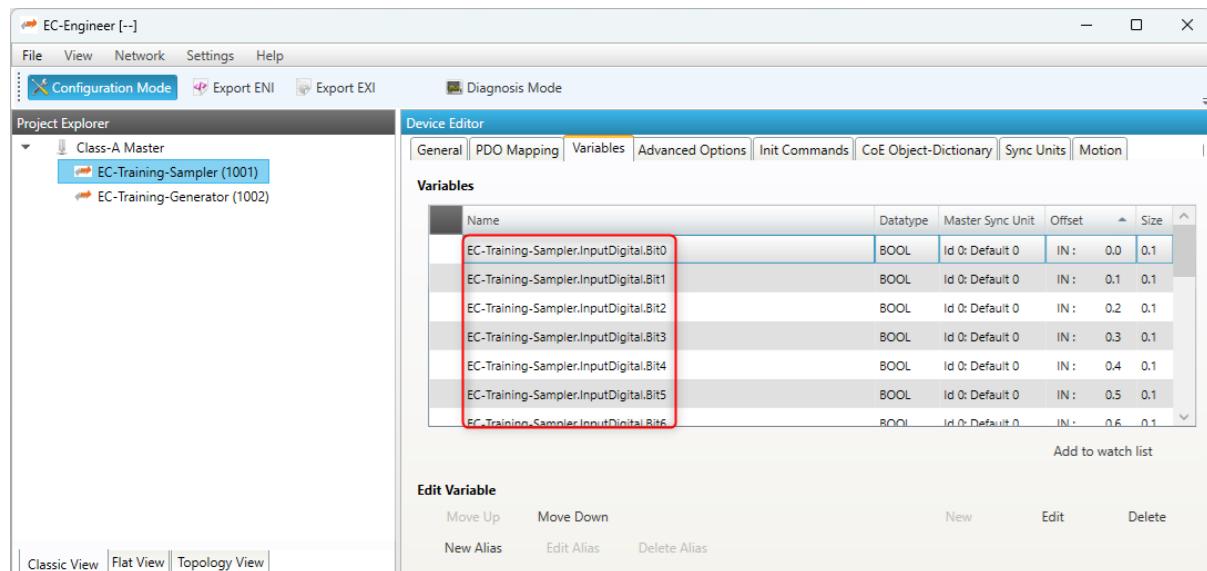
    return EC_E_NOERROR;
}
```

---

**Note:** The offsets from the PD layout C-header file (`PdLayout.h`) are byte offsets, *not* bit offsets!

---

It is possible to change the variable names of slaves before generating the PD layout C-header file to give them custom names:



This will generate a header file containing the customized variable names of the slaves as follows:

```
#include EC_PACKED_INCLUDESTART(1)
#define PDLAYOUT_OUT_OFFSET_EC_TRAINING_SAMPLER 0
typedef struct _T_PDLAYOUT_OUT_EC_TRAINING_SAMPLER
{
    EC_T_BYTE    byOutputDigital_Bit0 : 1;    // ...
    EC_T_BYTE    byOutputDigital_Bit1 : 1;    // ...
    EC_T_BYTE    byOutputDigital_Bit2 : 1;    // ...
    EC_T_BYTE    byOutputDigital_Bit3 : 1;    // ...
    EC_T_BYTE    byOutputDigital_Bit4 : 1;    // ...
    EC_T_BYTE    byOutputDigital_Bit5 : 1;    // ...
    EC_T_BYTE    byOutputDigital_Bit6 : 1;    // ...
    EC_T_BYTE    byOutputDigital_Bit7 : 1;    // ...
    EC_T_SWORD   swOutputAnalog_SpeedFactor; // ...
    EC_T_SWORD   swOutputAnalog_Reserved_2; // ...
    EC_T_SWORD   swOutputAnalog_Reserved_3; // ...
    EC_T_SWORD   swOutputAnalog_Reserved_4; // ...
} EC_PACKED (1) T_PDLAYOUT_OUT_EC_TRAINING_SAMPLER;
#include EC_PACKED_INCLUDESTOP
```

### Process data access example using generated PD layout C-header file with customized variable names of the slaves

```
#include "pdlayoutcustom.h"
static EC_T_DWORD myAppWorkpd(T_ECDemo_APP_CONTEXT* pAppContext)
{
    EC_T_DWORD dwInstanceId = pAppContext->dwInstanceId;
    T_PDLAYOUT_IN* poPdIn = (T_PDLAYOUT_IN*)emGetProcessImageInputPtr(dwInstanceId);
    ↪
    T_PDLAYOUT_OUT* poPdOut = (T_PDLAYOUT_OUT*)emGetProcessImageOutputPtr(dwInstanceId);
    ↪

    /* EC-Training-Generator.Counter1.Value (Offset: 9.0, Size: 4.0, Datatype:DINT) */
    EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
        "Counter1.Value: %d\n", EC_GET_FRM_DWORD(&poPdIn->sdwEC_Training_Generator_
    ↪Counter1_Value_Counter1_Value)));
}
```

(continues on next page)

(continued from previous page)

```

/* EC-Training-Generator.Counter1.Enable (Offset: 11.0, Size: 0.1, Datatype: →BOOL) */
    poPdOut->byEC_Training_Generator_Counter1_Enable_Counter1_Enable = 1;

/* EC-Training-Generator.Counter1.Increment (Offset: 9.0, Size: 2.0, Datatype: →INT) */
    EC_SET_FRM_WORD (&poPdOut->swEC_Training_Generator_Counter1_Increment_Counter1_Increment, 100);

    return EC_E_NOERROR;
}

```

**Note:** The offsets are subject to be changed if the ENI file changes. It is strongly recommended to determine the bit offsets of the variables on startup e.g. by querying the bit offset of a slave's process data by its station address from the EC-Master stack using [emGetCfgSlaveInfo\(\)](#) as described in the previous chapters instead of using hard coded values!

### 3.5.6 Process Data Access Functions

Process data variables that are packed as array of bits are bit aligned and not byte aligned in process data. See [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:** These functions do not initiate any transmission on the line. Process data is typically transmitted as little endian and must therefore be swapped on big endian systems such as PPC in order to be correctly interpreted, see e.g. [EC\\_SET\\_FRM\\_WORD](#), [EC\\_GET\\_FRM\\_WORD](#).

Variable Type	Bit Size	EC Type	Macro
INTEGER8, UNSIGNED8, BIT8	8	EC_T_BYTE	N/A
INTEGER16, UNSIGNED16	16	EC_T_WORD	<a href="#">EC_SET_FRM_WORD</a> , <a href="#">EC_GET_FRM_WORD</a>
INTEGER32, UNSIGNED32, REAL32	32	EC_T_DWORD	<a href="#">EC_SET_FRM_DWORD</a> , <a href="#">EC_GET_FRM_DWORD</a>
INTEGER64, UNSIGNED64, REAL64	64	EC_T_UINT64	<a href="#">EC_SET_FRM_QWORD</a> , <a href="#">EC_GET_FRM_QWORD</a>
BOOLEAN, BIT1...BIT7	1	EC_T_BOOL	<a href="#">EC_SETBITS</a> , <a href="#">EC_GETBITS</a>

## 3.6 Process Data Memory

All mapped process data objects of the slaves are copied by the master into a process data memory image. New input values received from the slaves are written to the input process data image. New output values to be sent to the slaves are read from the output process data image.

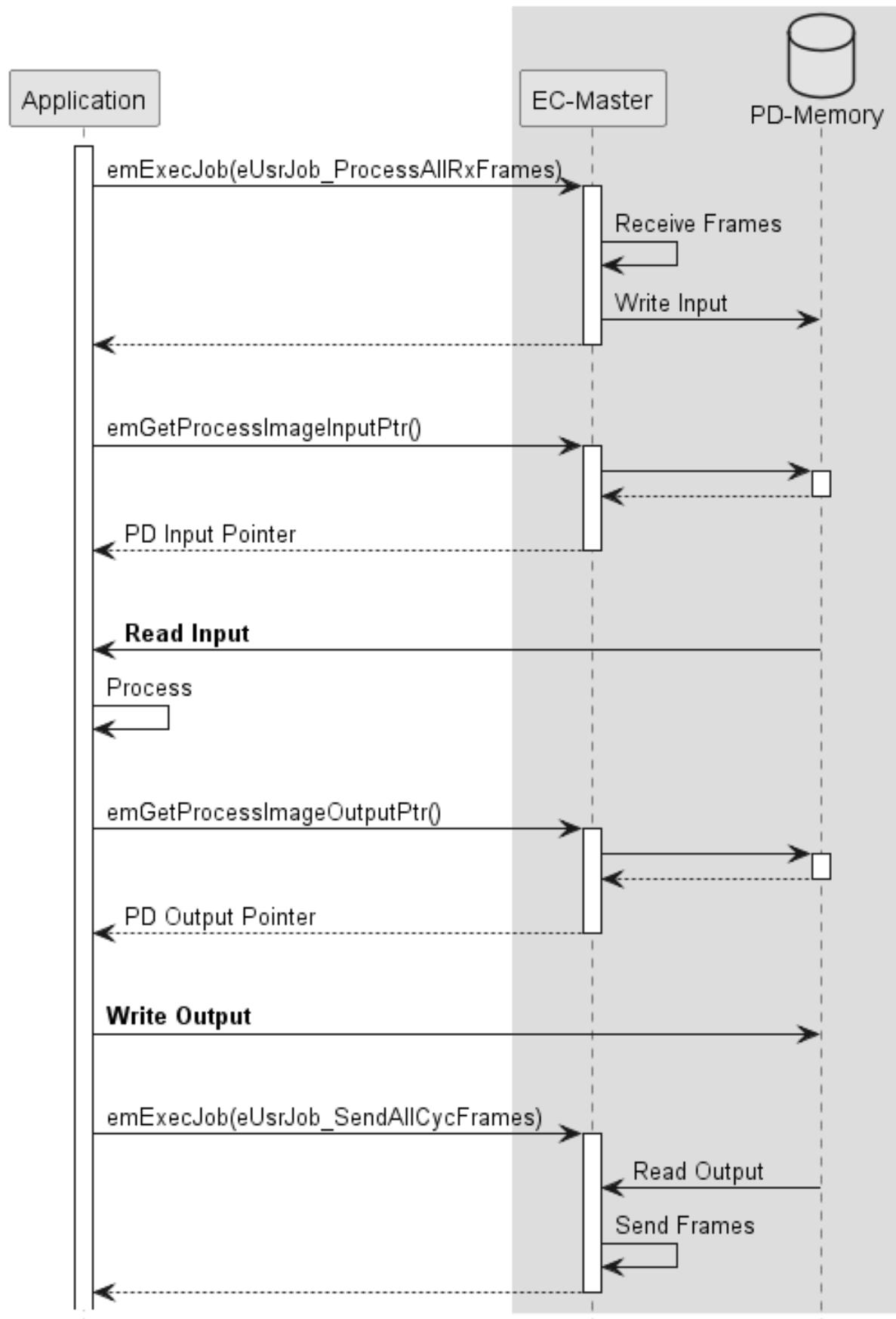
The EC-Master uses two separate buffers where process data input values and process data output values are stored. The buffers used may either be always the same (fixed buffers) or be changed on every process data transfer cycle (dynamic buffers).

The EC-Master has different options for how the process data memory is provided.

1. EC-Master provides process data memory (fixed buffers)
2. User application registers an external memory provider with fixed buffers
3. User application registers an external memory provider with dynamic buffers

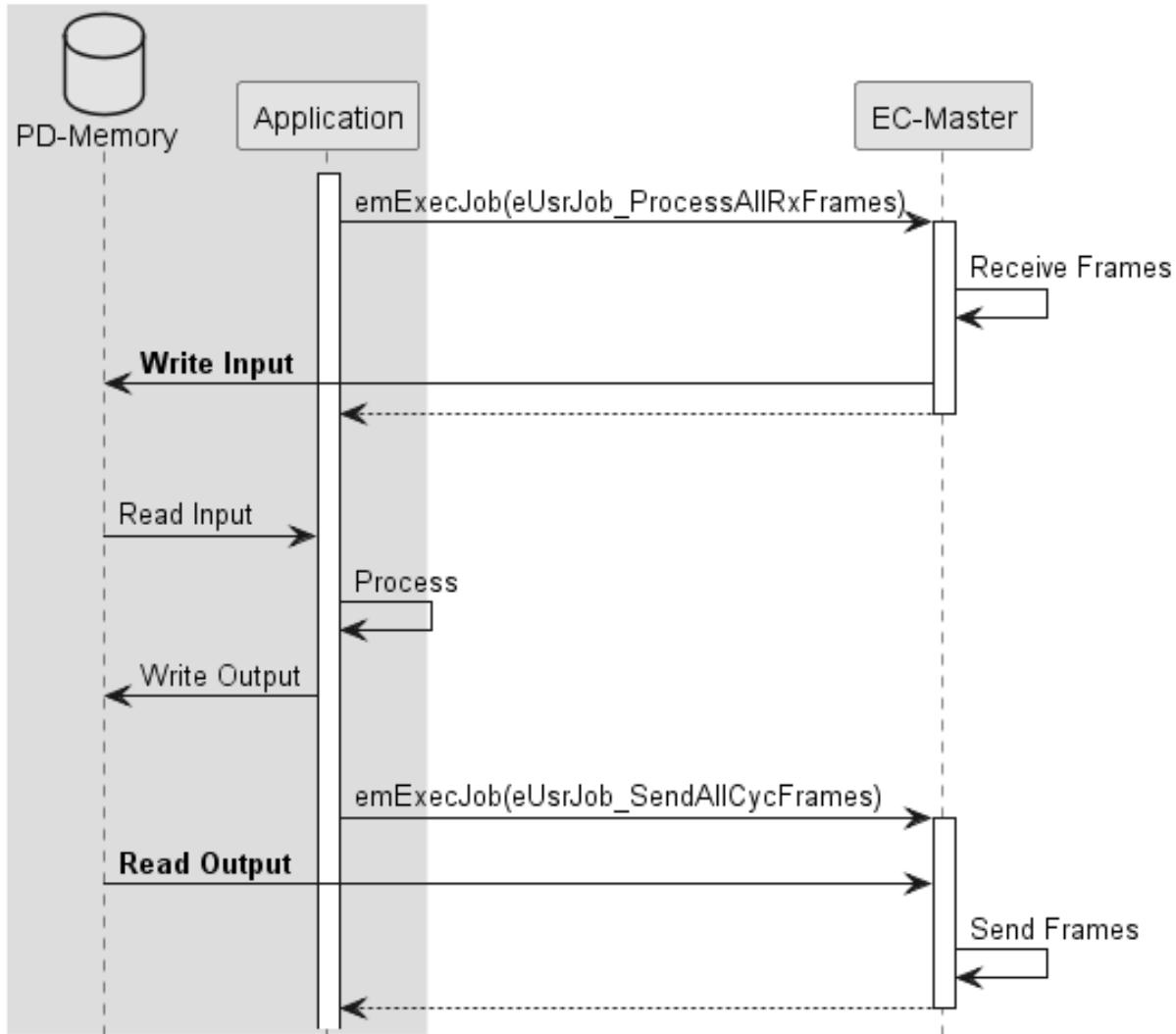
### 3.6.1 EC-Master as process data memory provider

If the application does not register a memory provider, the EC-Master internally allocates the required memory needed to store input and output process data values during `emConfigureNetwork()`. The EC-Master always uses the same buffers for reading/writing process data.



### 3.6.2 Application as process data memory provider with fixed buffers

The application may register a memory provider with [emIoControl - EC\\_IOCTL\\_REGISTER\\_PDMEMORYPROVIDER](#) in case the master shall use externally allocated memory to store input and output process data values.



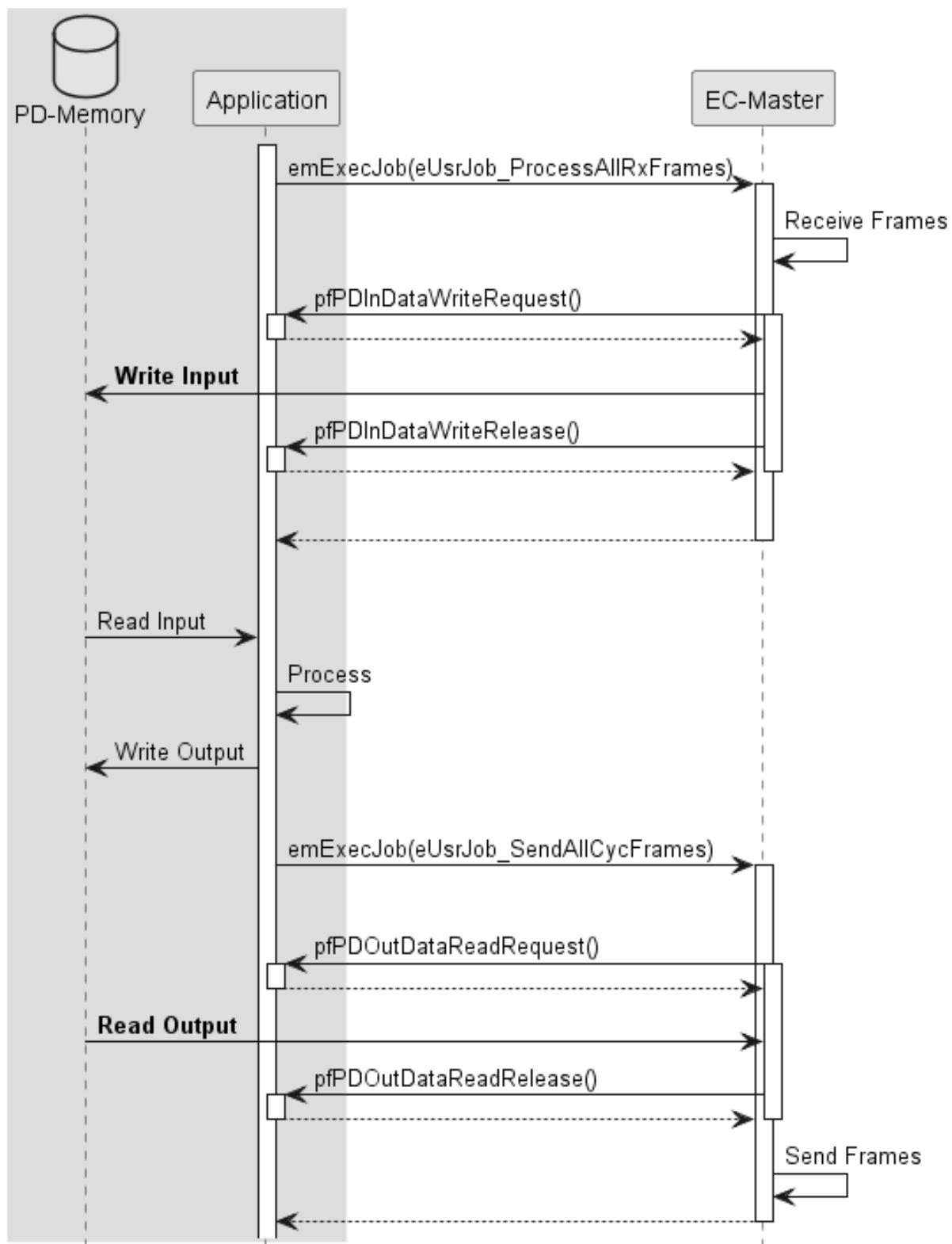
The memory provider may optionally supply callback functions to synchronize memory access between the application and the EC-Master.

#### Receiving new input process data:

- `EC_T_MEMPROV_DESC::pfPDIInDataWriteRequest`
- `EC_T_MEMPROV_DESC::pfPDIInDataWriteRelease`

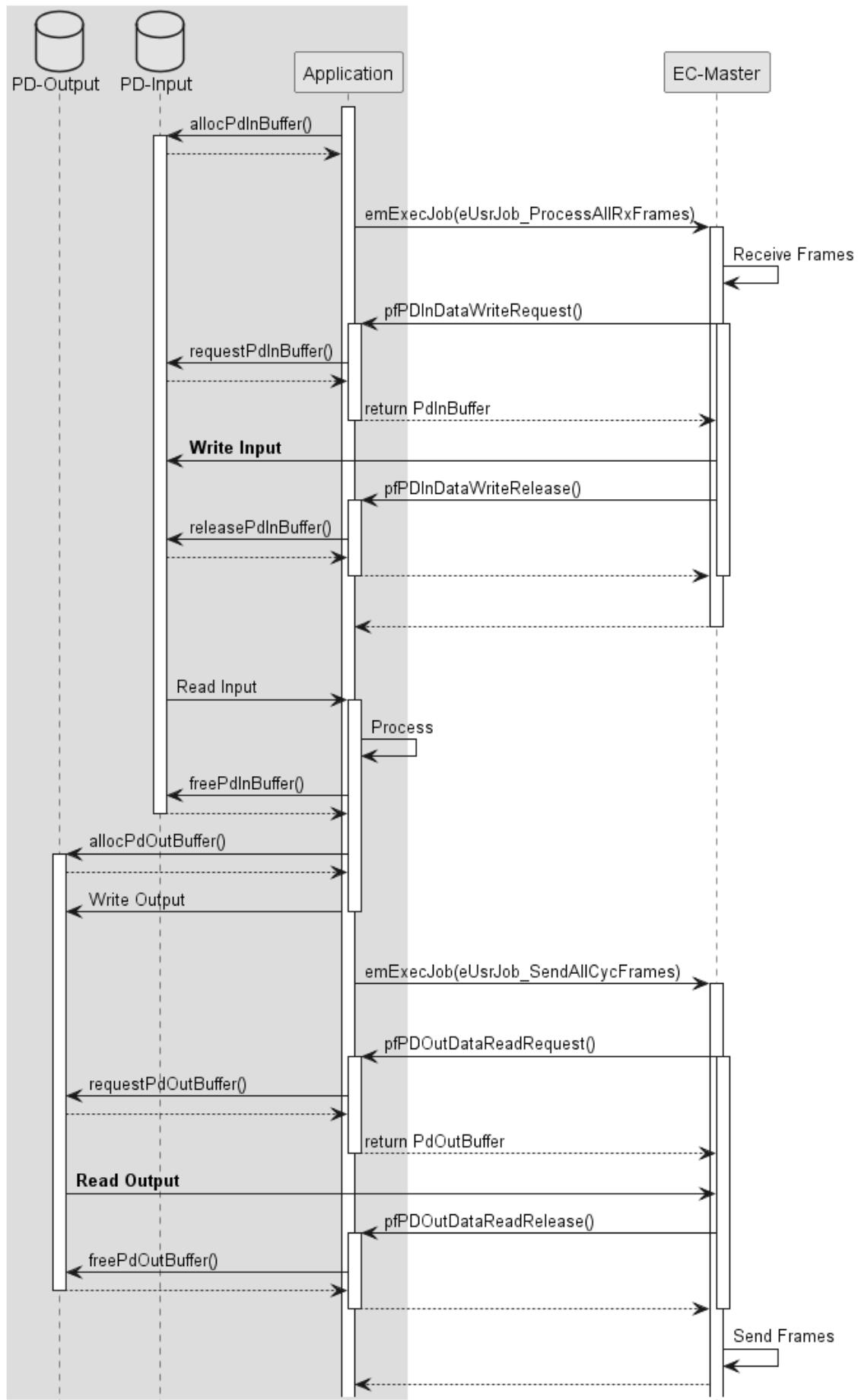
#### Sending new output process data:

- `EC_T_MEMPROV_DESC::pfPDOOutDataReadRequest`
- `EC_T_MEMPROV_DESC::pfPDOOutDataReadRelease`



### 3.6.3 Application as process data memory provider with dynamic buffers

The application registers an external memory provider without fixed buffers via *emIoControl - EC\_IOCTL\_REGISTER\_PDMEMORYPROVIDER* with the parameters *EC\_T\_MEMPROV\_DESC::pbyPDInData* and *EC\_T\_MEMPROV\_DESC::pbyPDOOutData* set to EC\_NULL. In this case, the EC-Master requests via the callback functions the buffer addresses cyclically when reading or writing process data. This mode can be used to implement dynamic buffering mechanisms between the application and the EC-Master, e.g. double buffering, triple buffering.



## 3.7 Error detection and diagnosis

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

One of the parameters that the client must set when registering with the EC-Master is a generic notification callback function (`emNotify()`). If an error is detected, the master calls this function.

The EC-Master log messages are enabled if `EC_T_LOG_PARMS` is configured as described in `emInitMaster()` or configured using `emSetLogParms()`.

### 3.7.1 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 configured WKC value.

**See also:**

`emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR`

### 3.7.2 Working Counter (WKC) State in Diagnosis Image

Each cyclic Process Data cmd has its own WKC State bit in the diagnosis image. The state is updated on frame receiving, frame loss detection or link disconnection. All process data variables within a datagram have the same WKC State value. If the WKC value of the received datagram is not as expected, the WKC State bit is set to 1 for this datagram (error). In case of Master Sync Units (MSU) if all the commands related to the MSU return WKC 0, the WKC State will be set to 1.

The WKC State offset within the Diagnosis Image is available at `EC_T_CFG_SLAVE_INFO` and `EC_T_PROCESS_VAR_INFO_EX`, `EC_T_MSU_INFO` see `emGetDiagnosisImagePtr()`, `emGetCfgSlaveInfo()`, `emGetSlaveInpVarInfoEx()`, `emGetSlaveOutpVarInfoEx()`, `emGetMasterSyncUnitInfo()`.

The application can check the WKC State of a variable e.g. as follows:

```
EC_T_CFG_SLAVE_INFO oSlaveInfo;
EC_T_BYTE* pbyDiagnosisImage = emGetDiagnosisImagePtr();
EC_T_BYTE byWkcState = 1;

if (EC_NULL != pbyDiagnosisImage)
{
    if (EC_NOERROR == emGetCfgSlaveInfo(EC_TRUE, 2302, &oSlaveInfo))
    {
        EC_GETBITS(pbyDiagnosisImage, &byWkcState, oSlaveInfo.wWkcStateDiagOffsOut[0],
        ← 1);
    }
}

if (1 == byWkcState)
{
    /* ... error ... */
}
```

**See also:**

*Cyclic cmd WKC validation*

### Behavior in case of automatically adjusted expected WKC value

Optionally, the expected WKC value can be automatically adjusted according the state and the presence of the slaves. See [emIoControl - EC\\_IOCTL\\_SET\\_AUTO\\_ADJUST\\_CYCCMD\\_WKC\\_ENABLED](#). The WKC State bits change synchronized to the corresponding notification, e.g. on link disconnection all slaves disappear and the behavior is as follows:

- All WKC State bits are set to 1 as missing data is not expected.
- The Master notifies the application about the link disconnection and the disappearing of slaves
- All WKC State bits are set to 0 as it is now expected to have no process data if all slaves are absent.

#### See also:

- [emNotify - EC\\_NOTIFY\\_ETH\\_LINK\\_NOT\\_CONNECTED](#)
- [emNotify - EC\\_NOTIFY\\_SLAVE\\_PRESENCE](#)

### 3.7.3 Master Sync Units (MSU)

MSUs are useful in grouping specific data (into consistency units) - Process Image: Variables are stored together within one memory block - Error checking: Own datagram(s) allow individual WKC state check (consistency unit)

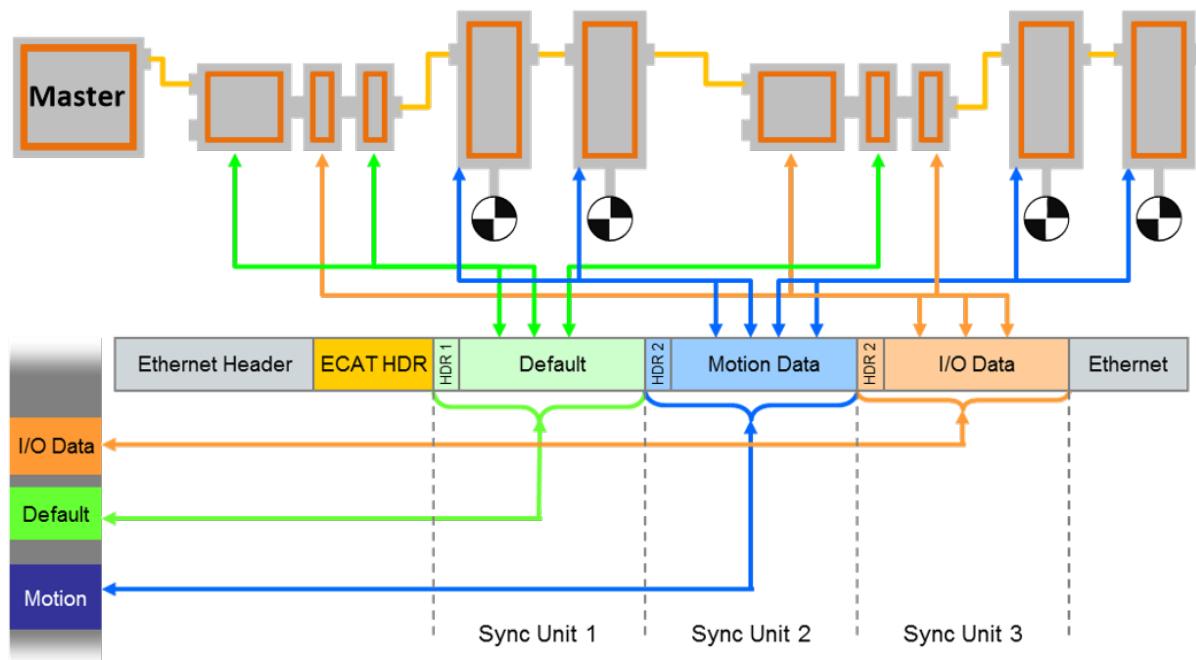


Figure out the MSU offsets by calling the functions [emGetMasterSyncUnitInfoNumOf\(\)](#) and [emGetMasterSyncUnitInfo\(\)](#), as described in the corresponding documentation.

## 3.8 EtherCAT traffic logging in application

```
emLogFrameEnable() emLogFrameDisable()
```

All network traffic can be recorded by starting EcMasterDemo with the parameter `-rec`. For this EcMaster-Demo needs to be compiled with preprocessor definition `INCLUDE_PCAP_RECORDER` defined. See class `CPCapRecorder` in `EcLogging.h/cpp`.

### EtherCAT traffic logging Example

```
EC_T_VOID /* EC_FNCALL */ LogFrameHandler(EC_T_VOID* /* pvContext */,
                                             EC_T_DWORD dwLogFlags, EC_T_DWORD dwFrameSize, EC_T_BYTE* pbyFrame)
{
    EC_T_WORD wFrameType = EC_ETHFRM_GET_FRAMETYPE(pbyFrame);
    EcLogMsg(EC_LOG_LEVEL_VERBOSE_CYC, (pEcLogContext, EC_LOG_LEVEL_VERBOSE_CYC,
                                         "%d: LogFrameHandler(): Type: 0x%04X (%s, %s %s), length: %d bytes\n",
                                         wFrameType,
                                         ((0 != (dwLogFlags & EC_LOG_FRAME_FLAG_RED_FRAME)) ? "RED" : "MAIN"),
                                         ((0 != (dwLogFlags & EC_LOG_FRAME_FLAG_RX_FRAME)) ? "RX" : "TX"),
                                         ((0 != (dwLogFlags & EC_LOG_FRAME_FLAG_ACYC_FRAME)) ? "acyclic" : "cyclic
                                         ")),
                                         dwFrameSize);
}
```

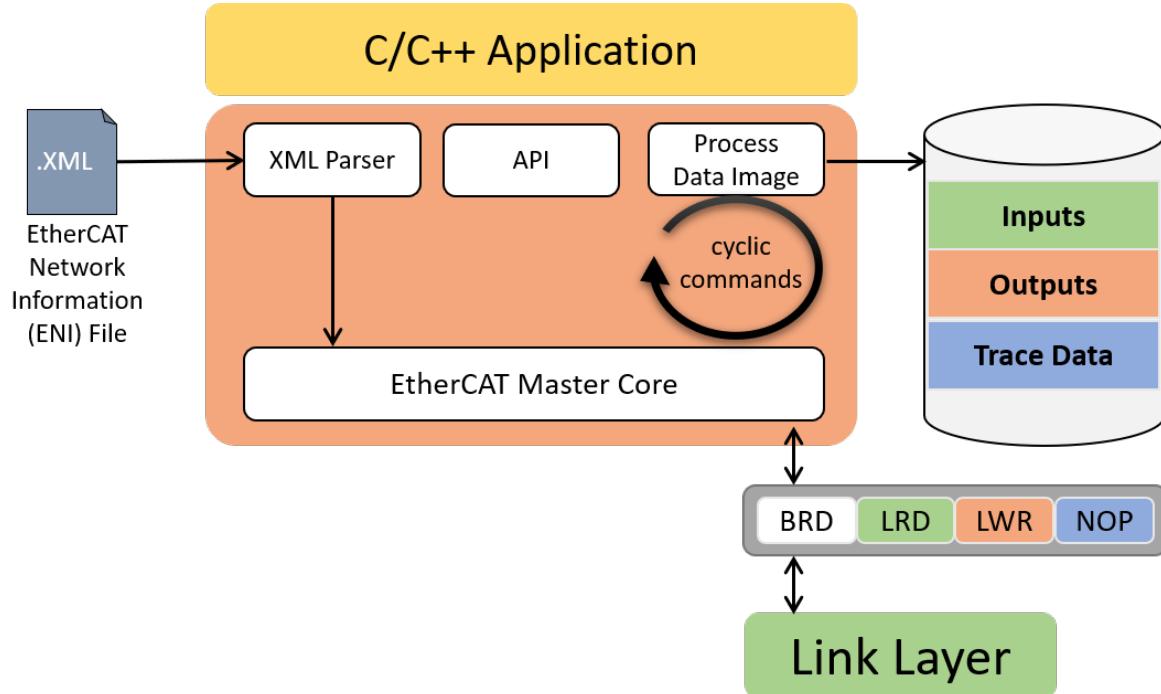
```
/* setup callback function to log EtherCAT network traffic */
EC_T_VOID* pvMyAppContext = this;
dwRes = emLogFrameEnable(dwInstanceId, LogFrameHandler, pvMyAppContext);
```

```
/* disable frame logging callback */
dwRes = emLogFrameDisable(dwInstanceId);
```

## 3.9 Trace Data

Trace Data allows applications to trace data in real time on the network. To ensure real-time transmission, it is implemented as part of the cyclic process data. They are placed behind the slave output data in the output area of the process data image of the EtherCAT application. The trace data area can be configured either via the ENI with the help of the EC-Engineer or without changing the ENI using the API `emTraceDataConfig()`.

Trace Data can be captured with a network monitoring tool like Wireshark.

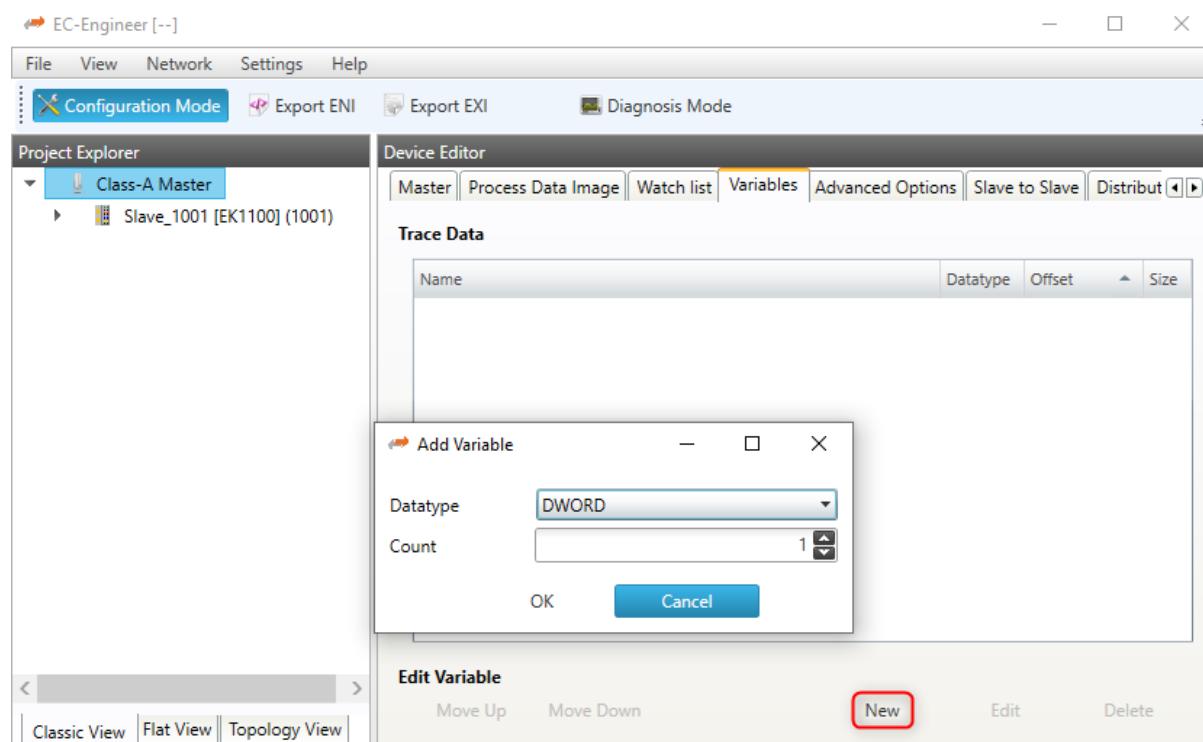


To transfer the data, an additional NOP cmd is appended to the end of the cyclic EtherCAT frame. The NOP cmd has ADP 0 and ADO 0x4154. The EC-Master automatically fills the data area of the NOP Cmd with the current trace data when sending cyclic frames. Since the trace data are transferred to the network as NOP Cmd, they are not evaluated by any ESC. Therefore, the WKC of the trace data remains 0 and the application cannot validate the data.

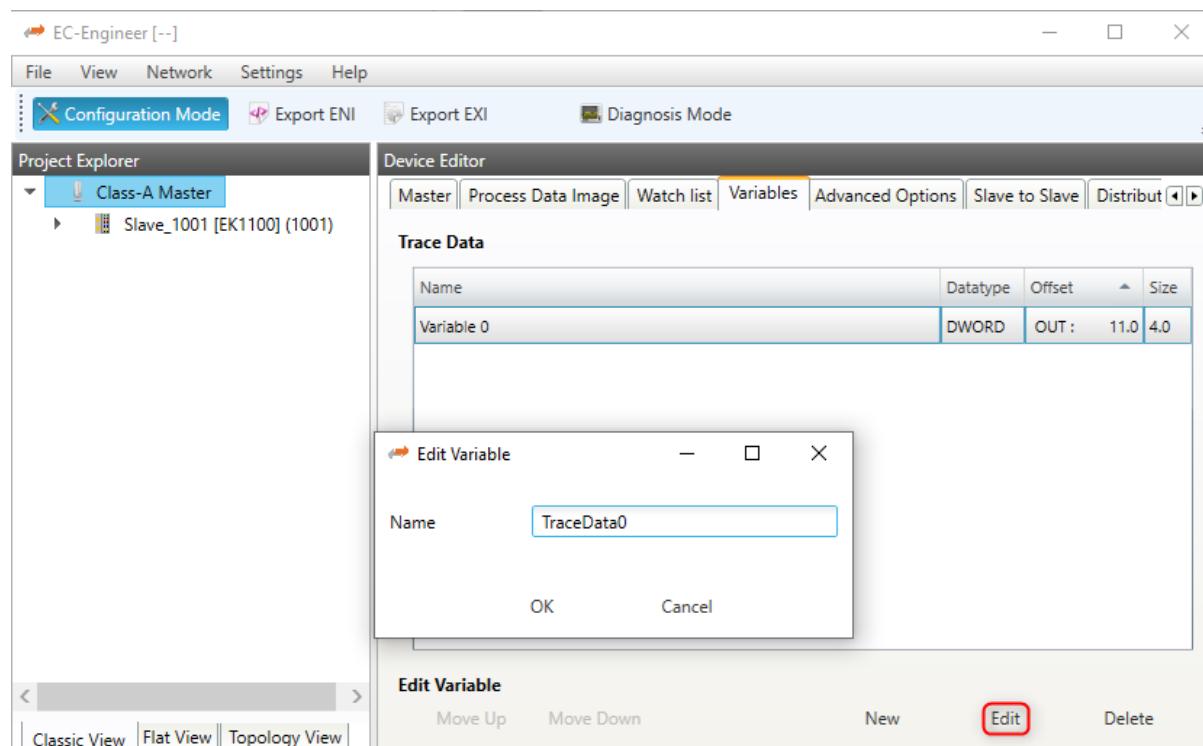
### 3.9.1 Trace Data configuration via EC-Engineer

The easiest and most comfortable way to create trace data variables is with the help of the EC-Engineer. The necessary NOP cmd and the process data variables are automatically created and exported to the ENI. The process variables can be accessed as usual using the `emFindOutpVarByNameEx()` function.

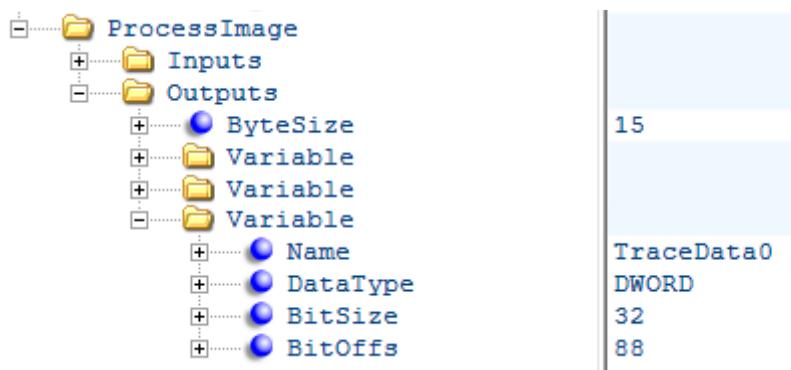
**Trace data variables of any size and number can be created in the Variables tab of the EC-Engineer:**



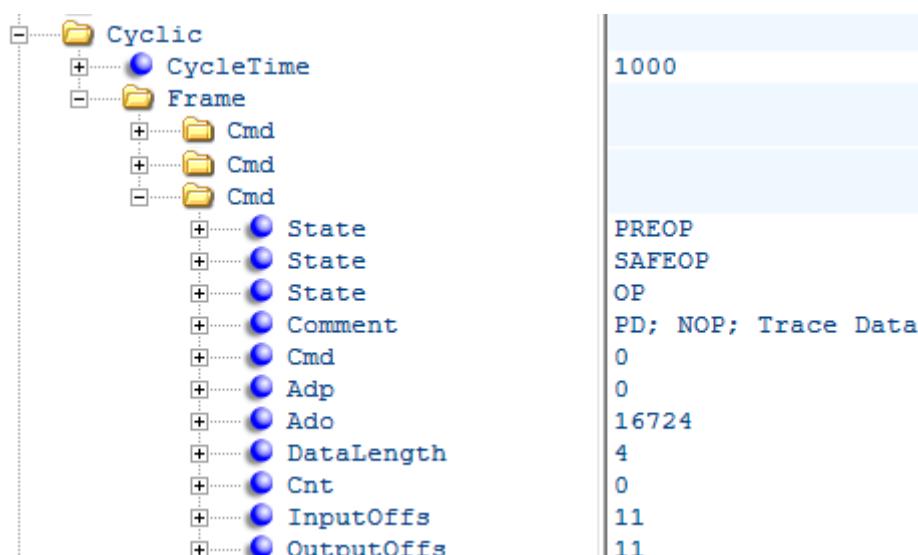
The automatically created variable names can also be edited:



The generated process variables can be found in the exported ENI file:



As well as the NOP cmd:



### 3.9.2 Trace Data configuration via API

The application can configure the trace data size using the `emTraceDataConfig()` API. The trace data configuration must take place between the initialization of the EC-Master (`emInitMaster()`) and the configuration of the network (`emConfigureNetwork()`). During `emConfigureNetwork()` the EC-Master tries to expand the process image output area by the trace data buffer and generates the corresponding NOP cmd.

Access to the trace data buffer is via an offset to the process data output image. The offset can be determined via the API `emTraceDataGetInfo()`.

Configuration of the trace data buffer:

```
//emInitMaster();

emTraceDataConfig(dwInstanceId, sizeof(EC_T_DWORD));

//emConfigureNetwork;
```

Access to the trace data buffer:

```
EC_T_TRACE_DATA_INFO oTraceDataInfo;
emTraceDataGetInfo(dwInstanceId, &oTraceDataInfo);

EC_SET_FRM_DWORD(oTraceDataInfo.pbyData + oTraceDataInfo.dwOffset, 0x11223344);
```

**Warning:**

- Trace data, encapsulated in an additional EtherCAT Cmd, must fit in the first cyclic frame.
- Trace data is not available for the fixed cyclic frame layout  
*EC\_T\_CYCFRAME\_LAYOUT::eCycFrameLayout\_FIXED*.

## 3.10 EtherCAT Master Stack Source Code

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

- SDK Header files
- Real-time Ethernet Driver files (multiple Real-time Ethernet Drivers may be shipped)
- Link OS driver files (only valid for the Real-time Ethernet Drivers)
- 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. Typically no supported build environment files like IDE projects are shipped with the source code.

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

### 3.10.1 Components

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

1. Real-time Ethernet Driver Binary (e.g. a downloadable object moduel in VxWorks or a DLL in Windows).  
The Real-time Ethernet Driver binary will be dynamically bound to the application at runtime. (currently not for On Time RTOS-32 which uses static libraries)
2. Master Stack Library
3. Remote API Server Library

#### Real-time Ethernet Driver Binaries

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

- Real-time Ethernet Driver files. Only one single Real-time Ethernet Driver must be selected even if multiple Real-time Ethernet Drivers are shipped. For each Real-time Ethernet Driver a separate binary has to be created.
- Link OS layer files
- Windows: a dynamic link library (.dll) has to be created. The name of the DLL has to be emlIIXxxx.dll where Xxxx shall be replaced by the Real-time Ethernet Driver type (e.g. emlII18255x.dll for the I8255x Real-time Ethernet Driver).
- VxWorks: a downloadable kernel module (.out) has to be created. The name of the module has to be eml-Ixxxx.out where Xxxx shall be replaced by the Real-time Ethernet Driver type (e.g. emlII18255x.out for the I8255x Real-time Ethernet Driver). sysLoSalAdd.c should be included in BSP if needed and should not be compiled within the Real-time Ethernet Driver 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 Real-time Ethernet Driver type (e.g. emlliI8255x.dll for the I8255x Real-time Ethernet Driver).
- INtime: a shared library (.rsl) has to be created. The name of the RSL has to be emllXxxx.rsl where Xxxx shall be replaced by the Real-time Ethernet Driver type (e.g. emlliI8255x.rsl for the I8255x Real-time Ethernet Driver).

## 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.

### See also:

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

### 3.10.2 Excluding features

It is possible to reduce the footprint of the master library and improve its execution performance by compiling less features.

#### **EXCLUDE\_EOE\_ENDPOINT**

FP-EoE-Endpoint

#### **EXCLUDE\_HOTCONNECT**

FP-Hot-Connect

#### **EXCLUDE\_JUNCTION\_REDUNDANCY**

FP-Cable-Redundancy

#### **EXCLUDE\_MASTER\_OBD**

FP-Master-Object-Dictionary

#### **EXCLUDE\_RED\_DEVICE**

FP-Cable-Redundancy

#### **EXCLUDE\_SPLITTED\_FRAME\_PROCESSING**

FP-Split-Frame-Processing

#### **EXCLUDE\_DC\_SUPPORT**

Class-A

The following defines and their impact are described below:

#### **EXCLUDE\_ADS\_ADAPTER**

```
emAdsAdapterStart()
```

```
emAdsAdapterStop()
```

### **EXCLUDE\_AOE\_SUPPORT**

```
emAoeGetSlaveNetId()
emAoeRead()
emAoeReadReq()
emAoeWrite()
emAoeWriteReq()
emAoeReadWrite()
emAoeWriteControl()
```

### **EXCLUDE\_BAD\_CONNECTIONS**

```
emBadConnectionsDetect()
```

### **EXCLUDE\_CONFIG\_EXTEND**

```
EC_T_CFG_SLAVE_INFO::bExtended
emConfigExtend()
```

### **EXCLUDE\_DCX**

DCM DCX mode

### **EXCLUDE\_EEPROM\_SUPPORT**

```
emReadSlaveEEProm()
emReadSlaveEEPromReq()
emWriteSlaveEEProm()
emWriteSlaveEEPromReq()
emReloadSlaveEEProm()
emReloadSlaveEEPromReq()
emAssignSlaveEEProm()
emAssignSlaveEEPromReq()
emActiveSlaveEEProm()
emActiveSlaveEEPromReq()
```

### **EXCLUDE\_EOE\_DEFERRED\_SWITCHING**

```
EC_T_USER_JOB::eUsrJob_SwitchEoeFrames
```

### **EXCLUDE\_EOE\_ENDPOINT**

```
emEoeRegisterEndpoint()
```

### **EXCLUDE\_EXECJOB\_REENTRANCY\_SUPPORT**

### **EXCLUDE\_FOE\_SUPPORT**

```
emFoeFileUpload()
emFoeUploadReq()
emFoeFileDownload()
emFoeDownloadReq()
emFoeSegmentedUploadReq()
emFoeSegmentedDownloadReq()
```

**EXCLUDE\_FORCE\_PROCESSDATA**

```
emForceProcessDataBits()
emReleaseProcessDataBits()
emReleaseAllProcessDataBits()
```

**EXCLUDE\_FRAME\_LOGGING**

```
emLogFrameEnable()
```

**EXCLUDE\_FRAME\_LOSS\_SIMULATION**

```
emIoControl - EC_IOCTL_SET_FRAME LOSS SIMULATION
emIoControl - EC_IOCTL_SET_RXFRAME LOSS SIMULATION
emIoControl - EC_IOCTL_SET_TXFRAME LOSS SIMULATION
```

**EXCLUDE\_GEN\_OP\_ENI**

```
emConfigExtend()
```

**EXCLUDE\_INTERFACE\_LOCK**

No API protection against InitMaster/DeinitMaster

**EXCLUDE\_LINE\_CROSSED\_DETECTION**

No line crossed detection

**EXCLUDE\_LOG\_MESSAGES**

No Log messages generated

**EXCLUDE\_MAILBOX\_STATISTICS**

```
EC_T_MASTER_INFO::MailboxStatistics
```

**EXCLUDE\_MASTER\_OBD**

```
EC_T_MASTER_INFO::BusDiagnosisInfo
```

**EXCLUDE\_MASTERSYNCUNITS**

```
EC_T_CFG_SLAVE_INFO::awMasterSyncUnitIn
emGetMasterSyncUnitInfoNumOf()
emGetMasterSyncUnitInfo()
```

**EXCLUDE\_MEMORY\_PROVIDER**

```
emIoControl - EC_IOCTL_REGISTER_PDMMEMORYPROVIDER
```

**EXCLUDE\_MULTIPLE\_CYC\_ENTRIES**

```
emIoControl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO
EC_T_USER_JOB::eUsrJob_ProcessRxFramesByTaskId
EC_T_USER_JOB::eUsrJob_SendCycFramesByTaskId
```

**EXCLUDE\_PORT\_OPERATION**

```
emBlockNode()
emOpenBlockedPorts()
emSetSlavePortState()
```

**EXCLUDE\_RAWMBX\_SUPPORT**

```
emClntSendRawMbx()
```

**EXCLUDE\_RED\_DEVICE**

```
EC_T_MASTER_INFO::RedundancyDiagnosisInfo
```

**EXCLUDE\_RESCUE\_SCAN**

```
emRescueScan()
```

**EXCLUDE\_S2SMBX\_SUPPORT**

```
EC_T_INIT_MASTER_PARMS::dwMaxS2SMBxSize
EC_T_INIT_MASTER_PARMS::dwMaxQueuedS2SMBxTfer
```

**EXCLUDE\_SLAVE\_HANDLING**

```
EC_T_CFG_SLAVE_INFO::bDisabled
EC_T_CFG_SLAVE_INFO::bDisconnected
emSetSlaveDisabled()
emSetSlavesDisabled()
emSetSlaveDisconnected()
emSetSlavesDisconnected()
```

**EXCLUDE\_SLAVE\_IDENTIFICATION**

```
/EtherCATConfig/Config/Slave/Info/Identification
EC_T_CFG_SLAVE_INFO::wIdentifyAdo
emReadSlaveIdentification()
```

**EXCLUDE\_SLAVE\_STATISTICS**

```
emGetSlaveStatistics()
emClearSlaveStatistics()
```

**EXCLUDE\_SOE\_SUPPORT**

```
emSoeRead()
emSoeReadReq()
emSoeWrite()
emSoeWriteReq()
emSoeAbortProcCmd()
```

**EXCLUDE\_SPLITTED\_FRAME\_PROCESSING**

```
EC_IOCTL_SET_SPLITTED_FRAME_PROCESSING_ENABLED
EC_T_USER_JOB::eUsrJob_ProcessRxFramesByTaskId
```

**EXCLUDE\_TEXT**

```
ecatGetText()
ecatGetNotifyText()
```

**EXCLUDE\_TRACE\_DATA****EXCLUDE\_TRACE\_DATA\_VARINFO**

```
emTraceDataConfig()
emTraceDataGetInfo()
```

**EXCLUDE\_VARREAD**

```
EC_T_CFG_SLAVE_INFO::wNumProcessVarsInp
EC_T_CFG_SLAVE_INFO::wNumProcessVarsOutp
emGetSlaveInpVarInfoNumOf()
emGetSlaveOutpVarInfoNumOf()
emGetSlaveInpVarInfo()
emGetSlaveInpVarInfoEx()
emGetSlaveOutpVarInfo()
emGetSlaveOutpVarInfoEx()
emGetSlaveOutpVarByObjectEx()
emGetSlaveInpVarByObjectEx()
emFindOutpVarByName()
emFindOutpVarByNameEx()
emFindInpVarByName()
emFindInpVarByNameEx()
```

**EXCLUDE\_VOE\_SUPPORT**

```
emVoeRead()
emVoeWrite()
emVoeWriteReq()
```

**EXCLUDE\_WKCSTATE**

```
EC_T_CFG_SLAVE_INFO::wWkcStateDiagOffsIn
EC_T_CFG_SLAVE_INFO::wWkcStateDiagOffsOut
emGetDiagnosisImagePtr()
```

## 3.11 Reduced Feature Set

On chosen platforms several EC-Master libraries with excluded features are available. They can be used to increase the performances or to reduce the footprint. They are defined incrementally as following. Each level includes the previous one:

### 3.11.1 Rfs1: Convenience functionality excluded

```
EXCLUDE_LOG_MESSAGES  
EXCLUDE_MASTER_OBD  
EXCLUDE_VARREAD
```

### 3.11.2 Rfs2: Rare functionalities excluded

```
EXCLUDE_ADS_ADAPTER  
EXCLUDE_CONFIG_EXTEND  
EXCLUDE_EOE_DEFERRED_SWITCHING  
EXCLUDE_EXECJOB_REENTRANCY_SUPPORT  
EXCLUDE_FRAME_LOGGING  
EXCLUDE_FRAME_LOSS_SIMULATION  
EXCLUDE_GEN_OP_ENI  
EXCLUDE_INTERFACE_LOCK  
EXCLUDE_RAWMBX_SUPPORT  
EXCLUDE_SLAVE_HANDLING  
EXCLUDE_SPLITTED_FRAME_PROCESSING  
EXCLUDE_TRACE_DATA  
EXCLUDE_TRACE_DATA_VARINFO
```

### 3.11.3 Rfs3: Common functionalities excluded

```
EXCLUDE_DC_ADD_ACYC_DISTRIBUTION  
EXCLUDE_DCX  
EXCLUDE EEPROM SUPPORT  
EXCLUDE_EOE_ENDPOINT  
EXCLUDE_FORCE_PROCESSDATA  
EXCLUDE_JUNCTION_REDUNDANCY  
EXCLUDE_MASTER_RED  
EXCLUDE_MASTERSYNCUNITS  
EXCLUDE_MEMORY_PROVIDER  
EXCLUDE_MULTIPLE_CYC_ENTRIES  
EXCLUDE_PORT_OPERATION  
EXCLUDE_RED_DEVICE  
EXCLUDE_RESCUE_SCAN  
EXCLUDE_SLAVE_IDENTIFICATION
```

### 3.11.4 Rfs4: Error detection and diagnosis excluded

```
EXCLUDE_BAD_CONNECTIONS  
EXCLUDE_CYCFRAMES_MONITORING  
EXCLUDE_LINE_CROSSED_DETECTION  
EXCLUDE_MAILBOX_STATISTICS  
EXCLUDE_SLAVE_STATISTICS  
EXCLUDE_WKCSTATE
```

### 3.11.5 Rfs5: Only CoE slaves supported

```
EXCLUDE_AOE_SUPPORT  
EXCLUDE_EOE_SUPPORT  
EXCLUDE_FOE_SUPPORT  
EXCLUDE_S2SMBX_SUPPORT  
EXCLUDE_SOE_SUPPORT  
EXCLUDE_VOE_SUPPORT
```

## 4 Platform and Operating Systems (OS)

### 4.1 CMSIS-RTOS for STM32

#### 4.1.1 Setting up and running EcMasterDemo in Keil µVision IDE

##### 1. Prerequisites

- Keil µVision 5 IDE
- STM32H747I-DISCO board
- EtherCAT devices

2. Connect the STM32H747I-DISCO development board to the PC according to user manual.
3. Connect EtherCAT devices to the board.
4. **Create ENI file for EtherCAT configuration.**

**xxd.exe** is capable of converting ENI files to a C file as array, e.g.

```
C:  
> xxd.exe -i eni.xml ENI.c
```

Replace ENI.c file with generated one. File should be manually modified to look like:

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

5. Start Keil µVision IDE and set the EcMasterDemoApp project as active.
6. If needed, change debug project settings.
7. Build and run EcMasterDemoApp.

##### See also:

*Running EcMasterDemo*

#### 4.1.2 OS Compiler settings

Besides the general settings from [Compiling the EcMasterDemo](#) the following settings are necessary to build the example application for CMSIS-RTOS (STM32).

##### Extra include paths

```
<InstallPath>/SDK/INC/CMSIS-RTOS  
<InstallPath>/Examples/Common/CMSIS-RTOS_STM32
```

##### Extra source paths

```
<InstallPath>/Examples/Common/CMSIS-RTOS  
<InstallPath>/Sources/OsLayer/CMSIS-RTOS
```

### Extra library paths to the main EtherCAT components

```
<InstallPath>/SDK/LIB/CMSIS-RTOS/mdk-arm
```

### Extra libraries

```
EcMaster.lib  
em11CmsisEth.lib  
EcMasterDemo.lib
```

## 4.1.3 Setting up and running EcMasterDemo in STM32CubeIDE for STM32H747I-DISCO

### 1. Prerequisites

- STM32CubeIDE V1.5.0
- EC-Master V3.1
- **CMSIS-RTOS sources package.** Use git [https://github.com/ARM-software/CMSIS\\_5.git](https://github.com/ARM-software/CMSIS_5.git) or download from [https://github.com/ARM-software/CMSIS\\_5/archive/develop.zip](https://github.com/ARM-software/CMSIS_5/archive/develop.zip).

---

**Note:** Alternative *ARM CMSIS Drivers for external devices* contains CMSIS-RTOS sources as well.

---

- *ARM CMSIS Drivers for external devices* from <https://www.keil.com/dd2/pack/> (for CMSIS PHY driver sources)
- *STMicroelectronics STM32H7 Series Device Support and Examples* from <https://www.keil.com/dd2/pack/> (for CMSIS MAC driver sources)

### 2. Environment variables

In order to be able to build and run demo application the following environment variables (either system or project variables) has to be defined:

- **CMSIS\_LOC**, has to be set to the CMSIS package location, i.e. C:/CMSIS\_5-5.7.0
- **FW\_LOC**, points to the firmware folder in STM32Cube repository, i.e. <PATH\_TO\_STM32CUBE\_REPOSITORY>/STM32Cube\_FW\_H7\_V1.8.0
- **PATH variable must contain the following paths (needed for tool chain):**

```
C:/ST/STM32CubeIDE_1.5.0/STM32CubeIDE/plugins/com.st.stm32cube.ide.  
→mcu.externaltools.gnu-tools-for-stm32.7-2018-q2-update.win32_1.5.  
→0.202011040924/tools/bin  
C:/ST/STM32CubeIDE_1.5.0/STM32CubeIDE/plugins/com.st.stm32cube.ide.  
→mcu.externaltools.make.win32_1.5.0.202011040924/tools/bin
```

- **PACKS\_LOC**, points to the packs location, where *ARM CMSIS Drivers for external devices* and *STMicroelectronics STM32H7 Series Device Support and Examples* were installed, i.e. C:/Users/<USER\_NAME>/AppData/Local/Arm/Packs

### 3. Build EcMasterDemo

- Build the EcMasterDemo project

- Create EtherCAT network configuration
- Build the EcMasterDemo\_STM32H747I-DISCO project for CM7 CPU

#### 4. Run on a STM32H747I-DISCO board

- Connect the board to PC using CN2 connector. This connection will be used for powering the board and for debugging as well.
- Connect EtherCAT cable to the Ethernet interface on the board and the EtherCAT slave(s).
- Power on EtherCAT slave(s).
- Using your favorite terminal application (i.e. Teraterm) connect to the serial port of STM32H747I-DISCO. Usually it is called *STMicroelectronics STLink Virtual COM Port*. Ensure it has the following settings: 115200, 8, N, 1.
- Create a debug or run configuration, select *STM32 Cortex-M C/C++ Application* as template. For this configuration select *SWD* in *GDB Server Command Line Options*.

---

**Note:** in order to let the application run with different command line parameter please change `szCommandLine` declared in `app_main.c`

---

## 4.2 eCos

### 4.2.1 Setting up and running EcMasterDemo

#### 1. Build the eCos kernel with the parameters associated to the application

As a starting point there is a eCos configuration file (.ecc) file located at `SDK/LIB/eCos/x86/`.

eCos is unable to get command line parameters for `main()`. The parameters for the application are build in the kernel via the configuration tool ( Arguments to `main` ).

Configuration	
Global build options	
Redboot HAL options	
Intel 82544 ethernet driver	v3_0
PC board ethernet driver	v3_0
eCos HAL	v3_0
I/O sub-system	v3_0
Infrastructure	v3_0
eCos kernel	v3_0
Dynamic memory allocation	v3_0
ISO C and POSIX infrastructure	v3_0
ISO C library	v3_0
ISO C library internationalization functions	v3_0
ISO C library setjmp/longjmp functions	v3_0
ISO C library signal functions	v3_0
ISO environment startup/termination	v3_0
Arguments to main()	{(char *)"name", (char *)"-v", (char *)"2", (char *)"-trx", (char *)"i8254x", (char *)"1 ", (char *)"1", (char *)NULL}
Startup context for main()	
main()'s default thread stack size	8192
Include atexit() function	<input checked="" type="checkbox"/>
Make exit() call fflush()	<input checked="" type="checkbox"/>
_exit() stops all threads	<input type="checkbox"/>
Default environment	{ NULL }
Invoke default static constructors	<input checked="" type="checkbox"/>
ISO environment startup/termination build option	
ISO C library standard input/output functions	v3_0
ISO C library general utility functions	v3_0
ISO C library string functions	v3_0
ISO C library date and time functions	v3_0
Math library	v3_0
Wallclock device	v3_0
Common error code support	v3_0
Disk device drivers	v3_0
Block cache and access library	v3_0
FAT filesystem	v3_0
POSIX File IO compatibility layer	v3_0
Linux compatibility layer	v3_0

To use an other example with different parameters, the kernel has to be rebuild.

For the EcMasterDemo example following line has to be passed to the application via the configuration tool:

```
{
    (char *) "name", (char *) "-f", (char *) "perf.xml",
    (char *) " -intelgbe", (char *) "1", (char *) "1", (char *) "-v",
    (char *) "3", (char *) "-t", (char *) "60000", (char *) "-perf", (char *)NULL
}
```

## 2. Compile EcMasterDemo

As a starting point there is the Eclipse project for EcMasterDemo for eCos located at Workspace/eCos/EcMasterDemo. The following macro in Sources/OsLayer/eCos/EcOs.cpp loads the ENI file from disk:

```
"MTAB_ENTRY(fat, "/", "fatfs", "/dev/idedisk1/1", 0)"
```

## 3. Copy the ENI file to target

eCos supports only the 8.3 file format. Adjust the ENI file name and the command line in the configuration tool accordingly.

## 4. Configure Grub to load the application

Adjust the Grub menu file:

```
title eCos EcMasterDemo
kernel (hd0,0) /EcMasterDemo
boot
```

## 5. Load and start the EcMasterDemo with Grub

## 6. Verify that the EcMasterDemo is running successfully

The EcMasterDemo takes some seconds to start. The following message is sent to the serial port on startup finished:

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

### See also:

[Running EcMasterDemo](#)

## 4.2.2 OS Compiler settings

Besides the general settings from [Compiling the EcMasterDemo](#) the following settings are necessary to build the example application for eCos.

### Extra include paths

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

### Extra source paths

```
<InstallPath>/Examples/Common/eCos  
<InstallPath>/Sources/OsLayer/eCos/EcOs.cpp
```

### Extra library paths to the main EtherCAT components

```
<InstallPath>/SDK/LIB/eCos
```

### Extra libraries

```
libEcMaster.a  
libeml1IntelGbe.a  
libtarget.a
```

## 4.3 FreeRTOS

### 4.3.1 Setting up and running EcMasterDemo on Xilinx Zynq UltraScale+ (ZCU104) and Xilinx Zynq-7000 (ZC702 Evaluation Kit)

Install Xilinx SDK 2018.2 or Install Xilinx Vitis IDE 2022.2

#### How to create the demo applications for Xilinx Zynq

##### 1. Create ENI file for EtherCAT configuration.

`xxd.exe` is capable of converting ENI files to a C file as array, e.g.

```
xxd.exe -i eni.xml ENI.c
```

Replace ENI.c file with generated one.

##### 2. Create a BSP project

Based on the delivered hardware project, replace the settings file with the one from the package:

for Xilinx SDK

```

setws .
importprojects .

createbsp -name ZCU104_bsp_cortexa53 -hwproject ZCU104_hw_platform -proc_
↪psu_cortexa53_0 -arch 64 -os freertos10_xilinx
createbsp -name ZCU104_bsp_cortexr5 -hwproject ZCU104_hw_platform -proc_
↪psu_cortexr5_0 -os freertos10_xilinx
createbsp -name ZC702Rev_bsp -hwproject ZC702Rev_hw_platform -proc ps7_
↪cortexa9_0 -os freertos10_xilinx
createbsp -name ZedBoard_bsp -hwproject ZedBoard_hw_platform -proc ps7_
↪cortexa9_0 -os freertos10_xilinx
createbsp -name ZC701ZynqDimm_bsp -hwproject ZC701ZynqDimm_hw_platform -
↪proc ps7_cortexa9_0 -os freertos10_xilinx

```

**replace:**

```
.../<BSP name>/<core name>/libsrv/freertos10_xilinx_v1_1/src/FreeRTOSConfig.h
```

for Vitis IDE

```

set PATH=C:\Xilinx\Vitis\2022.2\bin;%PATH%
set BUILD_SEVENZIP="C:\Program Files\7-Zip\7z.exe"

@echo "Build with Xilinx Software Commandline Tool (XSCT)"

REM Create Xilinx BSP's
@echo "Create Xilinx BSP's"
pushd %BUILDDOUTPUT%\Workspace\Freertos_Zynq_Vitis\
call xsct.bat ZCU104\platform.tcl
call xsct.bat ZCU106\platform.tcl
call xsct.bat Kria_KR260\platform.tcl
call xsct.bat zc702\platform.tcl

```

**replace:**

```
.../<BSP name>/psu_cortexr5_0/Freertos32BitR5/bsp/psu_cortexr5_0/libsrv/
↪freertos10_xilinx_v1_12/src/FreeRTOSConfig.h
```

For the new BSP project, just use the same BSP name and core as in the package.

## How to run the EC-Master demo applications on Xilinx Zynq

### Via USB debugger

Load the application with *Debug Configuration* ▶ *Xilinx C/C++ application (System Debugger)* to the chosen core.

### Via SD card

By creating a *BOOT.bin* file, e.g.:

### for Xilinx SDK

```
bootgen -w on -image ../EcMasterDemo_ZCU104_cortexa53.bif -arch zynqmp -o
↪ BOOT.bin
```

Maybe adjust the boot setting switches on the board

### 4.3.2 Setting up and running EcMasterDemo on TI AM64x EVM for R5 Core

Install MCU-PLUS-SDK-AM64X 08.01.00.36 and Code Composer Studio 11.1 or newer

#### How to create the demo applications on TI AM64x

##### 1. Create ENI file for EtherCAT configuration.

`xxd.exe` is capable of converting ENI files to a C file as array, e.g.

```
xxd.exe -i eni.xml ENI.c
```

Replace ENI.c file with generated one.

##### 2. rebuild BSP for the correct performance measurement

Change: `ti/mcu_plus_sdk_am64x_08_01_00_36/source/kernel/freertos/config/am64x/r5f/FreeRTOSConfig.h`

```
#define configUSE_IDLE_HOOK (0)
```

or:

`ti/mcu_plus_sdk_am64x_08_01_00_36/source/kernel/freertos/portable/TI_ARM_CLANG/ARM_CR5F/port.c`  
`vApplicationIdleHook () replace "wfi" with "nop"`

#### How to run the EC-Master demo applications on TI AM64x

##### 1. Run via Debugger

Follow getting started guide to flash the UART loader into the internal memory. Load the application with *Debug Configuration* ▶ *Code ComposerStudio - Device Debugging* and the Target Configuration to the R5F\_0 core.

##### 2. Run via µSD card

Adjust and rebuild sbl\_sd boot example in the mcu\_plus\_sdk. in main.c replace

```
#define BOOTLOADER_APPIMAGE_MAX_FILE_SIZE (0x60000) /* Size of section MSRAM_2_
→specified in linker.cmd */
uint8_t gAppImageBuf[BOOTLOADER_APPIMAGE_MAX_FILE_SIZE] __attribute__
→((aligned(128), section(".bss.filebuf")));
```

with

```
#define BOOTLOADER_APPIMAGE_MAX_FILE_SIZE (0x800000) /* This has to match the size_
→of DDR section in linker.cmd */
uint8_t gAppImageBuf[BOOTLOADER_APPIMAGE_MAX_FILE_SIZE] __attribute__
→((aligned(128), section(".bss.filebuf")));
```

### 4.3.3 Setting up and running EcMasterDemo on TI AM243x LP and TI AM243x EVM

Install MCU-PLUS-SDK-AM243X 08.01.00.36 and Code Composer Studio 11.2 or newer

## How to create the demo applications on TI AM243x

### 1. Create ENI file for EtherCAT configuration.

`xxd.exe` is capable of converting ENI files to a C file as array, e.g.

```
xxd.exe -i eni.xml ENI.c
```

Replace ENI.c file with generated one.

### 2. rebuild BSP for the correct performance measurement

Change: `ti/mcu_plus_sdk_am243x_08_01_00_36/source/kernel/freertos/config/am243x/r5f/FreeRTOSConfig.h`

```
#define configUSE_IDLE_HOOK (0)
```

or:

`ti/mcu_plus_sdk_am243x_08_01_00_36/source/kernel/freertos/portable/TI_ARM_CLANG/ARM_CR5F/port.c`  
`vApplicationIdleHook()` replace "wfi" with "nop"

## How to run the EC-Master demo applications on TI AM243x

### 1. Run via Debugger

See TI AM64x

## Handling Silicon Revision 2

### 1. Uart

If UART doesn't match the Baudrate of 115200 correct the Uart0 Clock Frequency from 96000000 to 48000000.

### 2. SD card boot

The silicon revision 2 has additional security features. The SD card has to be created with a later MCU-PLUS-SDK-AM243X version. Also the Appimage has to be created with a later SDK. So install `mcu_plus_sdk_am243x_09_01_00_41` and use `Workspace/FreeRTOS_AM243x/EcMasterDemo_am243x-evm/appimage_09_01_00_41.bat`.

### 4.3.4 Setting up and running EcMasterDemo on TI J784s4 EVM for R5 Core

Support for TiNetCpswg on J784s4 is currently limited to TI J784S4X EVM with FreeRTOS in polling mode. It's working with `ti-processor-sdk-rtos-j784s4-evm-08_06_01_03`.

Install PROCESSOR-SDK-RTOS-J784S4 Version: 08.06.01.03 and Code Composer Studio 11.2 or newer

## How to create the demo applications on TI J784s4

### 1. Create ENI file for EtherCAT configuration.

See TI AM64x

### 2. Adjust FreeRTOSConfig.h

See `.../ti-processor-sdk-rtos-j784s4-evm-08_06_01_03/pdk_j784s4_08_06_01_03/packages/ti/kernel/freertos/config/j784s4/r5f/FreeRTOSConfig.h`

Turn off `configUSE_IDLE_HOOK`, else `vApplicationIdleHook()` is called with `asm ("WFI");` is used. With WFI the ARM ccnt counter sleep and the performance measurement fails (WFI could also be replaced by NOP).

Heap size is not set with `configTOTAL_HEAP_SIZE` it's configured in the linker script `linker_r5_freertos.lds`

## How to run the EC-Master demo applications on TI J784s4

### 1. Run via Debugger

See TI AM64x

### 2. Run via µSD card

See sbl\_mmcisd example

## 4.4 tenAsys INtime

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

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

### 4.4.1 Setting up and running EcMasterDemo

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

```
> nodemgr start NodeA  
> sleep 5  
> piperta.exe -node NodeA -stderr EcMasterDemo.rta -intelgbe 1 1 -f eni.xml
```

#### See also:

*Running EcMasterDemo*

### 4.4.2 OS Compiler settings

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

#### Extra include paths

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

#### Extra source paths

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

#### Extra library paths to the main EtherCAT components

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

## 4.5 Linux

### 4.5.1 OS optimizations

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

The power management can disrupt cyclical processing, it is advisable to disable the *CPUIDLE sub-system* and *CPUFREQ sub-system*. The sub-systems can be disabled by changing the kernel command line parameters in the boot loader. On x86, x86\_64 systems this is usually *GRUB*, on embedded devices with ARM, ARM64 is usually *u-boot*. It is also possible to build a custom kernel without these sub-systems.

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

#### CPUIDLE sub-system

**Check if CPUFREQ sub-system is enabled:**

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

If cpuidle appears in the list, it is enabled.

**Disable CPUIDLE via the kernel command-line in GRUB:**

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

#### CPUFREQ sub-system

**Check if CPUFREQ sub-system is enabled:**

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

If cpufreq appears in the list, it is enabled.

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

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

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

**The currently active governor can be determined as follows:**

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

**The available governors with:**

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

**To change governor use:**

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

## ISOLCPUS

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

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

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

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

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

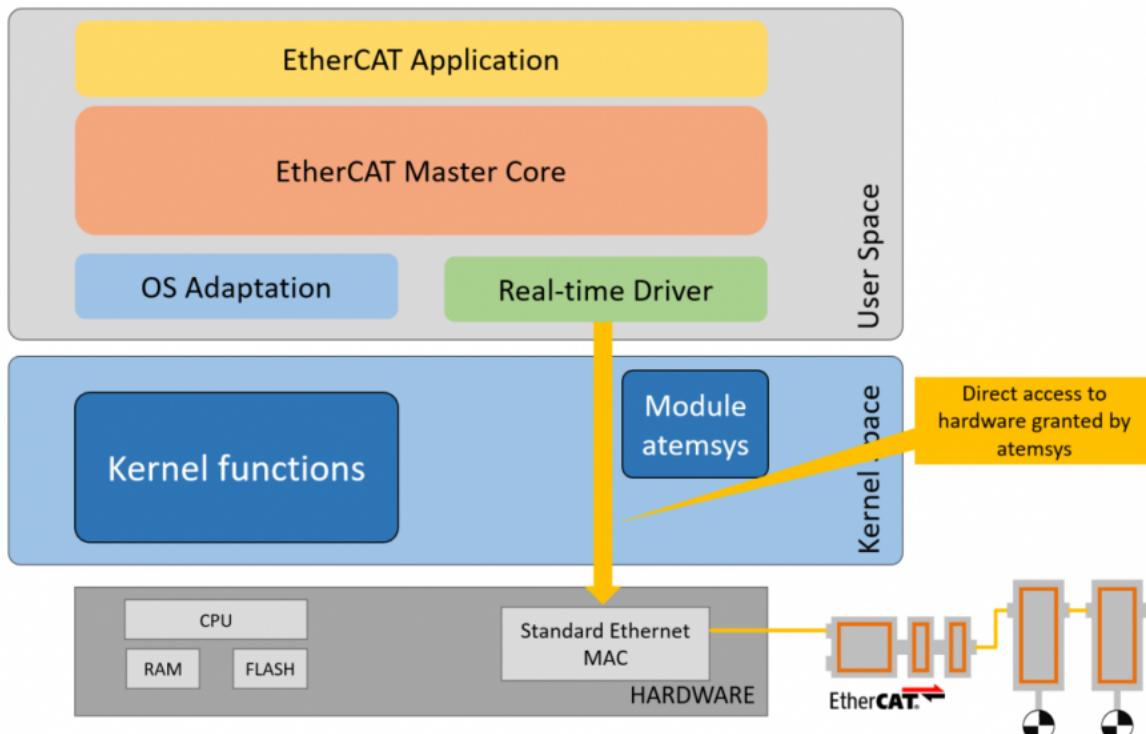
**Running EcMasterDemo on the isolated CPU core by setting the CPU affinity -a:**

```
$ ./EcMasterDemo -a 3
```

## 4.5.2 atemsys kernel module

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

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



## atemsys as Device Tree Ethernet Driver

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

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

---

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

---

## atemsys and PHY OS Driver

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

### 4.5.3 Unbind Ethernet Driver instance

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

#### Unbind from kernel driver

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

##### PCI

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

Example:

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

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

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

##### SOC

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

Example:

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

#### Unload kernel driver

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

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

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

Example:

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

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

```
$ lspci -v
```

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

Modules can be prevented from loading with the following commands:

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

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

Chip	Real-time Ethernet Driver	Kernel driver(s)	Remarks
Broadcom Genet	emllBcmGenet	genet	Unbind not supported
Beckhoff CCAT	emllCCAT	ec_bhf	
CPSW	emllCPSW	ti_cpsw	
Generic	emllDpdk		
DesignWare 3504	emllDW3504	stmmac	
	emllEG20T		
Freescale TSEC/eTSEC v1/2	emllETSEC	gianfar_driver	
Freescale FEC and ENET controller	emllFslFec	fec, fec_ptp	
Cadence GEM/MACB	emllGEM	gem, macb	
Intel Pro/1000	emlli8254x	igb, e1000, e1000e	
Intel Pro/1000	emlliIntelGbe	igb, e1000, e1000e	
Intel Pro/100	emlli8255x	e100	
ICSS	emlliICSS	prueth,pruss	Unbind not supported
RDC R6040	emllR6040		
Realtek RTL8139	emllRTL8139	8139too, 8139cp	
Realtek RTL8169 / RTL8111 / RTL8168	emllRTL8169	r8169	Unbind not supported
SuperH	emllSHEth	sh_eth	Unbind not supported
Generic	emllSockRaw		
Generic	emllSockXdp		

#### 4.5.4 Docker

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

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

**Add atemsys device to container**

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

**Allow max realtime priority**

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

#### Add capability to set priority an lock memory

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

#### Publish RAS server port 6000

```
-p 6000:6000
```

### 4.5.5 Setting up and running EcMasterDemo

1. Unbind Ethernet Driver instance, e.g.

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

2. Load atemsys kernel module

```
$ insmod atemsys.ko
```

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

4. Adjust *LD\_LIBRARY\_PATH* search locations for Real-time Ethernet Driver if necessary, e.g.

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

5. Run EcMasterDemo

```
$ cd /tmp  
$ ./EcMasterDemo -intelgbe 1 1 -f eni.xml -perf
```

#### See also:

*Running EcMasterDemo*

### Run in Docker container

1. Unbind Ethernet Driver instance and load atemsys on the host.
2. Create a directory on the host (e.g. ~/*docker*) and copy files from EC-Master package /bin and *exi.xml* into this directory.

3. Start bash console in container

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

#### Command line arguments:

- **-it** Allocate a pseudo-TTY and run container
- **--name** *atem\_container* Container name
- **--device=/dev/atemsy:/dev/atemsy** Add *atemsy* device to container
- **--ulimit rtprio=99** Allow max realtime priority
- **--cap-add=sys\_nice** Add Linux capability to set priority
- **--cap-add=ipc\_lock** Add Linux capability to lock memory

- `-v ~/docker:/home/docker` Mount previously created directory to container
- `-p 6000:6000` Publish RAS server port *6000*
- `ubuntu bash` Use Docker image `ubuntu` and start bash

#### 4. Run EcMasterDemo in container

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

### 4.5.6 OS Compiler settings

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

**Possible ARCHs (see ATECAT\_ARCHSTR in `SDK/INC/Linux/EcOsPlatform.h`):**

- aarch64 (ARM 64Bit)
- armv4t-eabi (ARM 32Bit)
- armv6-vfp-eabihf (ARM 32Bit)
- armv7-vfp-eabihf (ARM 32Bit)
- PPC (PPC 32Bit with “-te500v2”)
- riscv64 (RISC-V 64Bit)
- x64 (x86 64Bit)
- x86 (x86 32Bit)

The ARM 32Bit architectures *armv4t-eabi* and *armv6-vfp-eabihf/armv7-vfp-eabihf* are incompatible with each other. An ARM VFP system returns success on

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

#### Extra include paths

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

#### Extra source paths

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

#### Extra library paths to the main EtherCAT components

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

#### Extra libraries (in this order)

EcMasterRasServer EcMaster pthread dl rt
--

### 4.5.7 Build using cmake on Linux

Example usage to build Linux x64 Debug with cmake:

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

### 4.5.8 Cross-platform development under Windows

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

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

#### 1. Install MinGW

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

#### 2. Install a cross platform toolchain

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

#### 3. Build using cmake on Linux

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

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

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

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

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

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

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

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

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

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

```

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

```

## 4.6 PC / BIOS

A real-time behavior of a PC system may be optimized by changing various BIOS settings. As there are no real standards the following, settings may or may not exist on your BIOS.

### Disable

- *Legacy USB Support*
- *Hyper-Threading*
- *Intel C-STATE*
- *Intel SpeedStep*

### See also:

[Adjust BIOS Settings](#) in the acontis developer center

## 4.7 QNX Neutrino

### 4.7.1 Thread priority

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

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

```
$ procnto -P priority
```

For more information's about changing the priority range refer to the QNX documentation.

**Attention:** Don't changing the priority range leads to bad timing performance!

### 4.7.2 Unbind Ethernet Driver instance

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

#### QNX >= 6.5

```
ifconfig en1 destroy
```

#### QNX >= 7.1

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

### 4.7.3 IOMMU/SMMU support

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

**smm\_ LinkLayerName - InstanceNumber(32Bit Hex)**

**Example: Real-time Ethernet Driver eml1IntelGbe with instance number 1**

```
smm_eml1IntelGbe-0x00000001
```

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

### 4.7.4 Setting up and running EcMasterDemo

#### 1. QNX Neutrino OS configuration

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

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

```
$ ifconfig en1 destroy
```

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

#### 4. Adjust LD\_LIBRARY\_PATH search locations for Real-time Ethernet Driver if necessary, e.g.

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

#### 5. Run EcMasterDemo

```
$ cd /tmp
$ ./EcMasterDemo -intelgbe 1 1 -f eni.xml -perf
```

**See also:**

[Running EcMasterDemo](#)

### 4.7.5 OS Compiler settings

Besides the general settings from [Compiling the EcMasterDemo](#) the following settings are necessary to build the example application for QNX Neutrino.

#### Extra include paths

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

#### Extra source paths

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

#### Extra library paths to the main EtherCAT components

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

**Extra libraries (in this order)**

```
EcMasterRasServer EcMaster socket
```

## 4.8 Renesas

### 4.8.1 R-IN32M3

#### 1. Prerequisites

Hardware:

- R-IN32M3-EC Evaluation Board,
- adviceLUNA Emulator

Software:

- microVIEW-PLUS debugger,
- GNU compiler (Sourcery G++ Lite for ARM EABI)

2. Verify TCP/IP evaluation sample from Renesas works fine.

3. Download from official Renesas site following files:

- r-in32m3\_tcpip\_evaluation.zip
- r-in32m3\_samplesoft.zip.

4. Create ENI file for EtherCAT configuration.

**xxd.exe** is capable of converting ENI files to a C file as array, e.g.

```
C:  
> xxd.exe -i eni.xml ENI.c
```

Replace ENI.c file with generated one. File should be manually modified to look like:

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

5. Import project **Workspace/RIN32M3/EcMasterDemo** into Eclipse IDE.

Hardcoded parameters for the demo can be changed using DEMO\_PARAMETERS definition.

6. Upload **Workspace/RIN32M3/EcMasterDemo/Release/EcMasterDemo.bin** with debugger and run

See also:

*Running EcMasterDemo*

## 4.8.2 R-IN32M4

### 1. Prerequisites

Hardware:

- SBEV-RIN32M4CL3 Evaluation Board

Software:

- IAR Workbench 9.10.1,
- “R-IN32M4 series R-IN32M4-CL3 Driver/Middleware Release Note - Sample Code” package

### 2. Build EcMasterDemo

1. Install “R-IN32M4 series R-IN32M4-CL3 Driver/Middleware Release Note - Sample Code” and set the path the environment variable *MIDDLEWARE\_LOC* to the same folder.
2. Start IAR Workbench and import EcMasterDemo project into workspace
3. Create ENI file for EtherCAT configuration.

**xxd.exe** is capable of converting ENI files to a C file as array, e.g.

```
C:
> xxd.exe -i eni.xml MasterENI.c
```

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 = ???;
```

4. Import project Workspace/RIN32M3/EcMasterDemo into project.
5. Build the project and upload it to the board

See also:

*Running EcMasterDemo*

### 1. Troubleshooting

If after upload a program into serial FLASH the application has been trapped in HardFault\_Handler\_rom() reset the board with reset button.

## 4.9 IntervalZero RTX

EC-Master is available for the RTX versions listed below:

RTX version	EC-Master version
RTX 2012	V2.9.x.x
RTX 2016	V2.9.x.x
RTX64 2014	V2.9.x.x
RTX64 3.x	V3.1.x.x
RTX64 4.x	V3.1.x.x

### 4.9.1 Unbind Ethernet Driver instance

To use Real-time Ethernet Driver 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.

### 4.9.2 Setting up and running EcMasterDemo

The file EcMasterDemo.rtss has to be executed. The full path to the ENI file has to be given as a command line parameter as well as the appropriate Real-time Ethernet Driver.

```
> RTSSrun EcMasterDemo.rtss -i8255x 1 1 -f C:/eni.xml
```

#### See also:

*Running EcMasterDemo*

### 4.9.3 OS Compiler settings

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

#### Extra include paths

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

#### Extra source paths

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

#### Extra library paths to the main EtherCAT components

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

## 4.10 SylinxOS

### 4.10.1 Setting up and running EcMasterDemo on SylinxOS

Install OS on your hardware with RealEvo-IDE V5.0.4

## How to create the demo application

1. Start RealEvo-IDE V5.0.4
2. Import EcMasterDemo project
3. Adjust path to Base Project
4. Build

## How to run the EcMasterDemo applications

1. Copy Executable, shared libraries and ENI file to the target
2. Run application

```
$ ./EcMasterDemo ...
```

## 4.11 TI-RTOS

### 4.11.1 Setting up and running EcMasterDemo

#### Prerequisites, basic settings:

##### TI SDK RTOS v4.02 for AM335x/AM437x/AM57x

Make sure your Code Composer Studio uses correct versions of SYS/BIOS, XDCtools and PDK. For TI SDK 4.02 corresponding versions are:

- XDCtools: 3.50.3\_33\_core
- PDK 1.0.9
- SYS/BIOS 6.52.0.12

Ensure environment variable PDK\_INSTALL\_PATH is pointing to the installed root directory of SDK. Eg: For AM572x demo project, PDK\_INSTALL\_PATH=C:/ti/pdk\_am57xx\_1\_0\_9/packages

##### TI SDK RTOS for AM654x

At lease Version 9 of the Code Composer Studio is needed and together with the TI Processor SDK-rtos for am65xxevm Version 07.01.00.14 this lead to the packages:

- XDCtools: 3.61.3\_29\_core
- PDK 7.1.0.55
- SYS/BIOS 6.83.0.18

Ensure environment variable PDK\_INSTALL\_PATH is pointing to the installed root directory of SDK. For AM654x demo project, PDK\_INSTALL\_PATH=C:/ti/pdk\_am65xx\_07\_01\_00\_55/packages

## How to create the demo applications

### 1. Create ENI file for EtherCAT configuration.

xxd.exe is capable of converting ENI files to a C file as array, e.g. “xxd.exe -i eni.xml ENI.c”. Replace MasterENI.c file with the generated one

### 2. On TI RTOS the EcMasterDemo can run with either an Real-time Ethernet Driver CPSW or ICSS.

Eg: AM572x with Real-time Ethernet Driver ICSS

Workspace/TI-RTOS\_AM57x in Code Composer Studio and import all projects from this directory:

- EcMaster
- emlIICSS
- EcMasterDemoICSS or
- EcMasterDemoDcICSS

Hardcoded parameters for the demo can be changed using DEMO\_PARAMETERS definition.

## How to run the EC-Master demo applications

- The compiled .out application files of the demo can be uploaded to the device via JTAG debugger from Code Composer Studio Debugger.
- The SD Card bootable demo binary is generated as an APP file from post build script calling pdkAppImage-Create.bat from the PSDK package.

## How to run the EC-Master motion demo application

### 1. Create an appropriate ENI file as en EcMasterDemo with the xxd.exe tool

The DC configuration has to be done appropriately, please see the EtherCAT general documentation and EC-Master manuals for details

### 2. Create an appropriate motion demo configuration file and copy it into the config directory of the project

See example in Examples/EcMasterDemoMotion/Config/DemoConfig.xml See additional info in: Examples/EcMasterDemoMotion/readme.txt

3. Convert DemoConfig.xml to C file DemoConfig.c as array.
4. Build the Project and download it to target Processor.
5. The demo logging is done over UART.

### See also:

[Running EcMasterDemo](#)

## 4.11.2 OS Compiler settings

Besides the general settings from [Compiling the EcMasterDemo](#) the following settings are necessary to build the example application for TI-RTOS.

### Extra include paths

```
<InstallPath>/SDK/INC/TI-RTOS
<InstallPath>/Examples/Common/TI-RTOS
```

### Extra source paths

```
<InstallPath>/Examples/Common/TI-RTOS  
<InstallPath>/Sources/OsLayer/TI-RTOS
```

**Extra library paths to the main EtherCAT components**

```
<InstallPath>/SDK/LIB/TI-RTOS
```

## 4.12 μC3 for STM32

### 4.12.1 Setting up and running EcMasterDemo in IAR for ARM IDE

**1. Prerequisites**

- IAR for ARM IDE, v9.32
  - μC3/Compact
  - STM32H743 target
  - EtherCAT devices
2. Connect the STM32H743 target to the PC.
  3. Connect EtherCAT devices to the board.
  4. **Create ENI file for EtherCAT configuration.**

**xxd.exe** is capable of converting ENI files to a C file as array, e.g.

```
C:  
> xxd.exe -i eni.xml ENI.c
```

Replace ENI.c file with generated one. File should be manually modified to look like:

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

5. Start IAR for ARM IDE and set the EcMasterDemo\_STM32H743.
6. If needed, change debug project settings.
7. Build and run EcMasterDemo.

**See also:**

*Running EcMasterDemo*

## 4.12.2 OS Compiler settings

Besides the general settings from *Compiling the EcMasterDemo* the following settings are necessary to build the example application for µC3 (STM32).

### Extra include paths

```
<InstallPath>/SDK/INC/uC3
<InstallPath>/Examples/Common/uC3
```

### Extra source paths

```
<InstallPath>/Examples/Common/uC3
```

### Extra library paths to the main EtherCAT components

```
<InstallPath>/SDK/LIB/uC3/M7-EWARM
```

### Extra libraries

```
EcMaster.a
em11CmsisEth.a
```

## 4.13 µC3 for i.MX8

### 4.13.1 Setting up and running EcMasterDemo on NXP 8MPLUSLPD4-EVK board

#### 1. Prerequisites

- GCC compiler for aarch64 9.2.1
  - GNU make 4.4.1
  - µC3/Standard for Cortex-A53 MPcore i.MX8M Plus GCC
  - NXP 8MPLUSLPD4-EVK board
  - EtherCAT devices
2. Install GCC toolchain from <https://developer.arm.com/downloads/-/gnu-a/9-2-2019-12>
  3. Install make from <https://xpack-dev-tools.github.io/>
  4. **Create ENI file for EtherCAT configuration.**

**xxd.exe** is capable of converting ENI files to a C file as array, e.g.

```
C:
> xxd.exe -i eni.xml MasterENI.c
```

Replace `Workspace\uC3_iMX8\EcMasterDemo_8MPLUSPD4-EVK\ENI\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 = ???;
```

## 5. Build EcMasterDemo

```
cd Workspace\uC3_iMX8\EcMasterDemo\
make
```

## 6. Build EcMasterDemo\_8MPLUSPD4-EVK

```
cd Workspace\uC3_iMX8\EcMasterDemo_8MPLUSPD4-EVK\
make
```

## 7. Connect the target board to the PC. The J23 ‘DEBUG’ connector is user for serial output, connect it to PC and use your terminal software of choice in order to display EcMasterDemo output. The following connection settings will be used:

- Baude rate: 115200
- Data: 8 bits
- Parity: none
- Stop: 1 bit
- Flow control: none

8. Connect EtherCAT devices to the board, use “ENET1” connector.

9. Prepare an u-boot SD card and copy `Workspace\uC3_iMX8\EcMasterDemo_8MPLUSPD4-EVK\aaarch64\EcMasterDemo_8PLUSPD4-EVK.bin` on it.

## 10. Boot from SD card into u-boot, interrupt booting process if needed:

```
...
Fastboot: Normal
Normal Boot
Hit any key to stop autoboot: 2
u-boot=>
```

## 11. Load the binary to RAM:

```
u-boot=>fatload mmc $mmcdev:1 0x95000000 EcMasterDemo_8PLUSPD4-EVK.bin
```

## 12. Start EcMasterDemo:

```
u-boot=>go 0x95000000
```

## 4.14 μC3 for Renesas RZ/N2H

### 4.14.1 Setting up and running EcMasterDemo for RZ/N2H

#### 1. Prerequisites

- Renesas e2studio 2024-10
- RZ/N2H board
- Segger J-Link
- EtherCAT devices

2. Set environment variables to point to μC3 kernel and driver folders, i.e.

```
set UC3_KERNEL_LOC=C:\uC3_RZN2H\Kernel\Standard
set UC3_DRIVER_LOC=C:\uC3_RZN2H\Driver\Standard
```

3. Install and run e2studio, please select support of RZ family as optional feature.
4. Create a new workspace and import EcMasterDemo project from `Workspace\uC3_RZN2H_e2studio`
5. **Create ENI file for EtherCAT configuration.**

**`xxd.exe` is capable of converting ENI files to a C file as array, e.g.**

```
C:
> xxd.exe -i eni.xml MasterENI.c
```

Replace `Workspace\uC3_RZN2H_e2studio\EcMasterDemo_xxx\ENI\MasterENI.c` file with generated one. This file should be manually modified for `EcMasterDemo_xxx_SRAM` to look like:

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

This file should be manually modified for `EcMasterDemo_xxx_DDR` to look like:

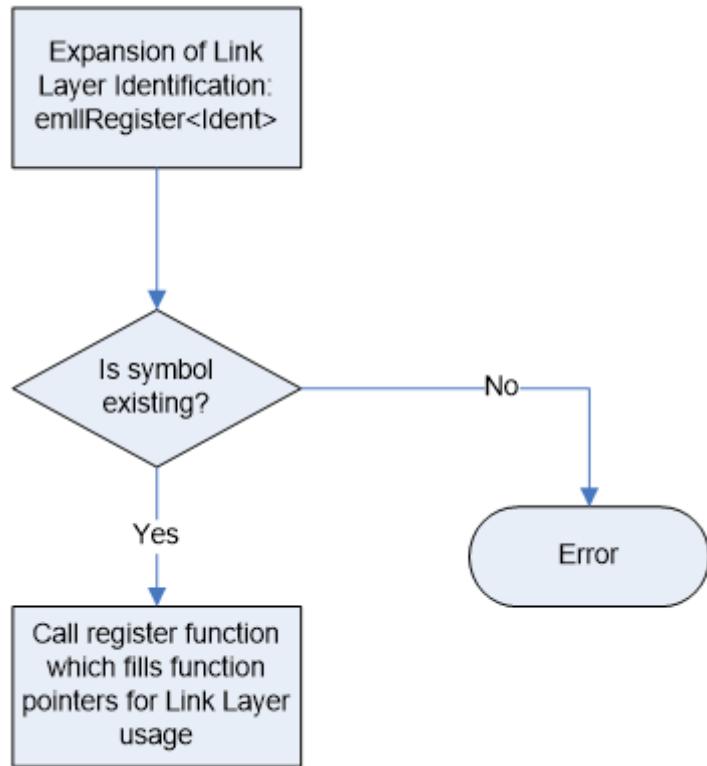
```
const unsigned char MasterENI_xml_data[] __attribute__((section(".eni_file
➥"))) = {
...
};
const unsigned int MasterENI_xml_data_size __attribute__((section(".eni_
➥file"))) = ???;
```

6. Build EcMasterDemo projects and debug it as “Renesas GDB Hardware Debugging” using “J-Link ARM” as debugging hardware and “R9A09G087M44\_CA55\_0” as target device.

## 4.15 Windriver VxWorks

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

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



### 4.15.1 VxWorks native

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

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

```

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

```

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

#### See also:

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

### 4.15.2 SNARF Ethernet Driver

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

### 4.15.3 Setting up and running EcMasterDemo

#### 1. VxWorks OS configuration

See sections above.

#### 2. Determine the network interface

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

#### 3. Connection of the EtherCAT® slaves

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

#### 4. Download an Real-time Ethernet Driver module

The Real-time Ethernet Driver library (e.g. `eml1IntelGbe.out`) which contains hardware support for the corresponding NIC must be downloaded. By default the Ethernet Driver `eml1SnarfGpp` are contained with the binary delivery.

#### 5. Download the example application

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

#### 6. Set up a FTP server connection on host

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

#### 7. Check for exclusive hardware access

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

```
-> muxShow
```

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

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

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

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

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

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

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

```
-> muxDevUnload "rtg", 1  
(e.g. first instance of the FEC on Freescale iMX platform):  
-> muxDevUnload "motfec", 1  
(e.g. first instance of the ETSEC on Freescale PPC platform):  
-> muxDevUnload "motetsec", 1
```

#### 8. Run the example application

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

Loading and running the demo:

```
-> ld<EcMasterDemo.out  
-> sp EcMasterAppMain, "-intelgbe 1 1 -f eni.xml"
```

Example:

```

172.17.7.148 - PuTTYtel

-> ld<em11i8254x.out
value = 78468256 = 0x4ad54a0
-> ld<EcMasterDemo.out
value = 78582152 = 0x4af1188 = G_dwLinkOsUnLockCounter + 0x3dc
-> sp atemDemo, "-f EL9800.xml -i8254x 1 1 -v 2"
Task spawned: id = 0x4af1bac, name = t1
value = 78584748 = 0x4af1bac = G_dwLinkOsUnLockCounter + 0xe00
-> Full command line: -f EL9800.xml -i8254x 1 1 -v 2

tEcTimingTask: bus cycle time: 1000 us (using Sleep)
Run demo now!

=====
Initialize EtherCAT Master
=====
EtherCAT Master V2.6.1 Build 99 Copyright acontis technologies GmbH
Evaluation Version, stop sending ethernet frames after 480 minutes!
Evaluation Version, number of slaves supported = 12!
Evaluation starts now ...
Bus scan successful - 1 slaves found

*****
Number : 0
Vendor : Beckhoff (Product Management), ID 2
Product : EL9820, Code: 0x4570862
Revision: 0x1f4008e Serial Number: 0
ESC Type: Beckhoff ET1100 (0x11) Revision=0 Build=2
Bus AutoInc Address: 0 (0x0)
Bus Station Address: 1001 (0x3e9)
Bus Alias Address : 4103 (0x1007)
Config Station Address: 1001 (0x3e9)
PD OUT Byte.Bit offset: 0.0 Size: 32 bits
Port 0: Connected Port 1: Not_Conn. Port 2: Not_Conn. Port 3: Not_Conn.

=====
Start EtherCAT Master
=====
Master state changed from <UNKNOWN> to <INIT>
Master state changed from <INIT> to <PREOP>
Master state changed from <PREOP> to <SAFEOP>
Master state changed from <SAFEOP> to <OP>

```

**See also:**

*Running EcMasterDemo*

#### 4.15.4 OS Compiler settings

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

##### Extra include paths

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

## Extra source paths

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

## Extra library paths to the main EtherCAT components

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

### GNU/PowerPC

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

## 4.16 Microsoft Windows

### 4.16.1 EcMasterDemo

#### 1. Install EC-Master

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

#### 2. Determine the network interface

For example the option `-ndis 192.168.1.1 1` will be using the network adapter card with the IP address 192.168.1.1.

#### 3. Connect EtherCAT modules

Any EtherCAT module can be directly connected to the target system. EtherCAT requires a 100 Mbit/s connection. If the Ethernet adapter card does not support this speed, an Ethernet switch must be used.

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

#### 4. Run the example application

Execute `EcMasterDemo.exe` from `<InstallPath>/Bin/Windows/<Arch>/`. At least an Ethernet Driver option has to be given.

```
C:
> EcMasterDemo -ndis 192.168.1.1 1 -f D:/eni.xml -sp
```

```
C:\Windows\system32\cmd.exe - EcMasterDemo.exe -winpcap 192.168.1.1 1 -f D:\eni.xml
D:\temp\EC-Master-Windows-x86_64Bit\Bin\Windows\x64>EcMasterDemo.exe -winpcap 192.168.1.1 1 -f D:\eni.xml
0000000005: EcMasterDemo V3.1.1.02 (Unrestricted) for Windows_x64 Copyright acontis technologies GmbH @ 2021
0000000005: Full command line: -winpcap 192.168.1.1 1 -f "D:/eni.xml"
0000000006: EC-Master V3.1.1.02 (Unrestricted) for Windows_x64 Copyright acontis technologies GmbH @ 2021
0000000414: EtherCAT network adapter MAC: 68-05-CA-3D-03-E7
0000000508: Bus scan successful - 6 slaves found
0000000519: Master state changed from <UNKNOWN> to <INIT>
0000000575: Master state changed from <INIT> to <PREOP>
0000000618: Master state changed from <PREOP> to <SAFEOP>
0000000639: Master state changed from <SAFEOP> to <OP>
0000000640: EcMasterDemo will stop in 600s...
```

#### See also:

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

## 4.16.2 EcMasterDemoDotNet (.NET) - Microsoft Windows

### 1. Install EC-Master

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

### 2. Determine the network interface

For example the option `-link ndis 192.168.1.1 1` will be using the network adapter card with the IP address 192.168.1.1.

### 3. Connect EtherCAT modules

Any EtherCAT module can be directly connected to the target system. EtherCAT requires a 100 Mbit/s connection. If the Ethernet adapter card does not support this speed, an Ethernet switch must be used.

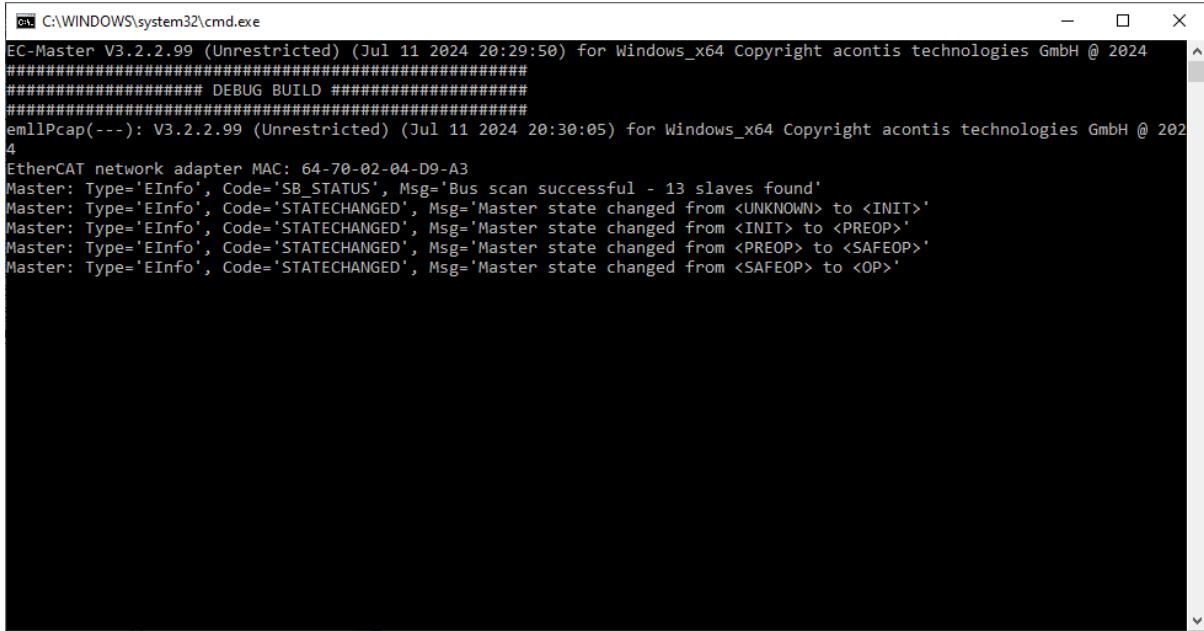
**Warning:** The local IT infrastructure should not be mixed with EtherCAT modules on the same Ethernet adapter. The EC-Master sends many broadcast packets!

### 4. Run the example application

Execute `<InstallPath>/Bin/Windows/<Arch>/EcMasterDemoDotNet.exe`. At least an Ethernet Driver option has to be given.

C:

```
> EcMasterDemoDotNet -mode 1 -file eni.xml -link "ndis 192.168.1.1 1"
→ -sp 6000 (-help will show usage)
```



```
C:\WINDOWS\system32\cmd.exe
EC-Master V3.2.2.99 (Unrestricted) (Jul 11 2024 20:29:50) for Windows_x64 Copyright acontis technologies GmbH @ 2024
#####
##### DEBUG BUILD #####
#####
em11Pcap(--): V3.2.2.99 (Unrestricted) (Jul 11 2024 20:30:05) for Windows_x64 Copyright acontis technologies GmbH @ 2024
EtherCAT network adapter MAC: 64-70-02-04-D9-A3
Master: Type='EInfo', Code='SB_STATUS', Msg='Bus scan successful - 13 slaves found'
Master: Type='EInfo', Code='STATECHANGED', Msg='Master state changed from <UNKNOWN> to <INIT>'
Master: Type='EInfo', Code='STATECHANGED', Msg='Master state changed from <INIT> to <PREOP>'
Master: Type='EInfo', Code='STATECHANGED', Msg='Master state changed from <PREOP> to <SAFEOP>'
Master: Type='EInfo', Code='STATECHANGED', Msg='Master state changed from <SAFEOP> to <OP>'
```

## 4.16.3 EcMasterDemoGuiDotNet (.NET) - Microsoft Windows

### 1. Prerequisites

To run the EcMasterDemoGuiDotNet.exe, the libraries EcMaster.dll, EcMasterRasServer.dll, EcWrapperDotNet.dll, EcWrapper.dll and em11Ndis.dll from Bin/Windows/x64 are needed in Bin/Windows/x64/Release, Bin/Windows/x64/Debug

### 2. Visual Studio C#-project

The C#-project for VS2015 or higher is located at Workspace/WindowsVS2015/EcMasterDemoGuiDotNet/EcMasterDemoGuiDotNet.csproj

3. If the reference EcWrapperDotNet is missing, it must be re-added from Bin/Windows/x64/EcWrapperDotNet.dll

4. EcMasterDemoGuiDotNet is now prepared for Run/Debug

#### 4.16.4 OS Compiler settings

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

##### Extra include paths

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

##### Extra source paths

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

##### Extra library paths to the main EtherCAT components

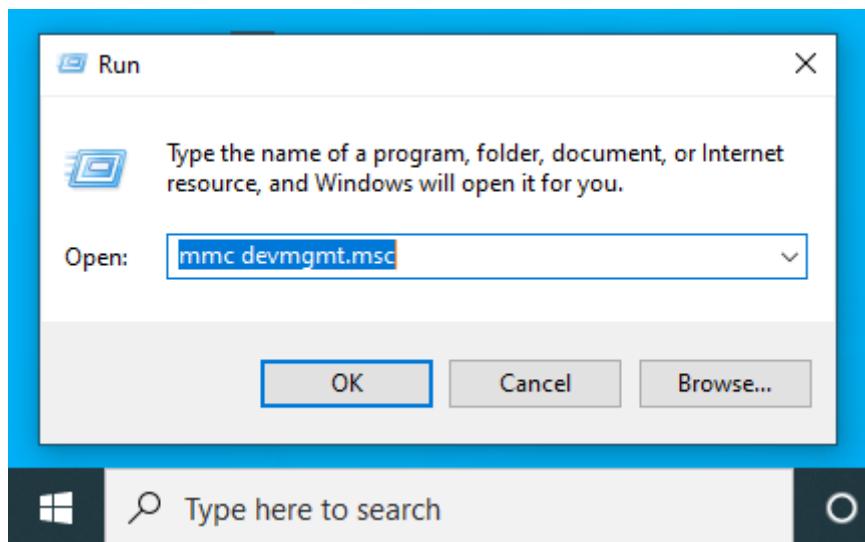
```
<InstallPath>\SDK\LIB\Windows
```

#### 4.16.5 RtaccDevice for Real-time Ethernet Driver

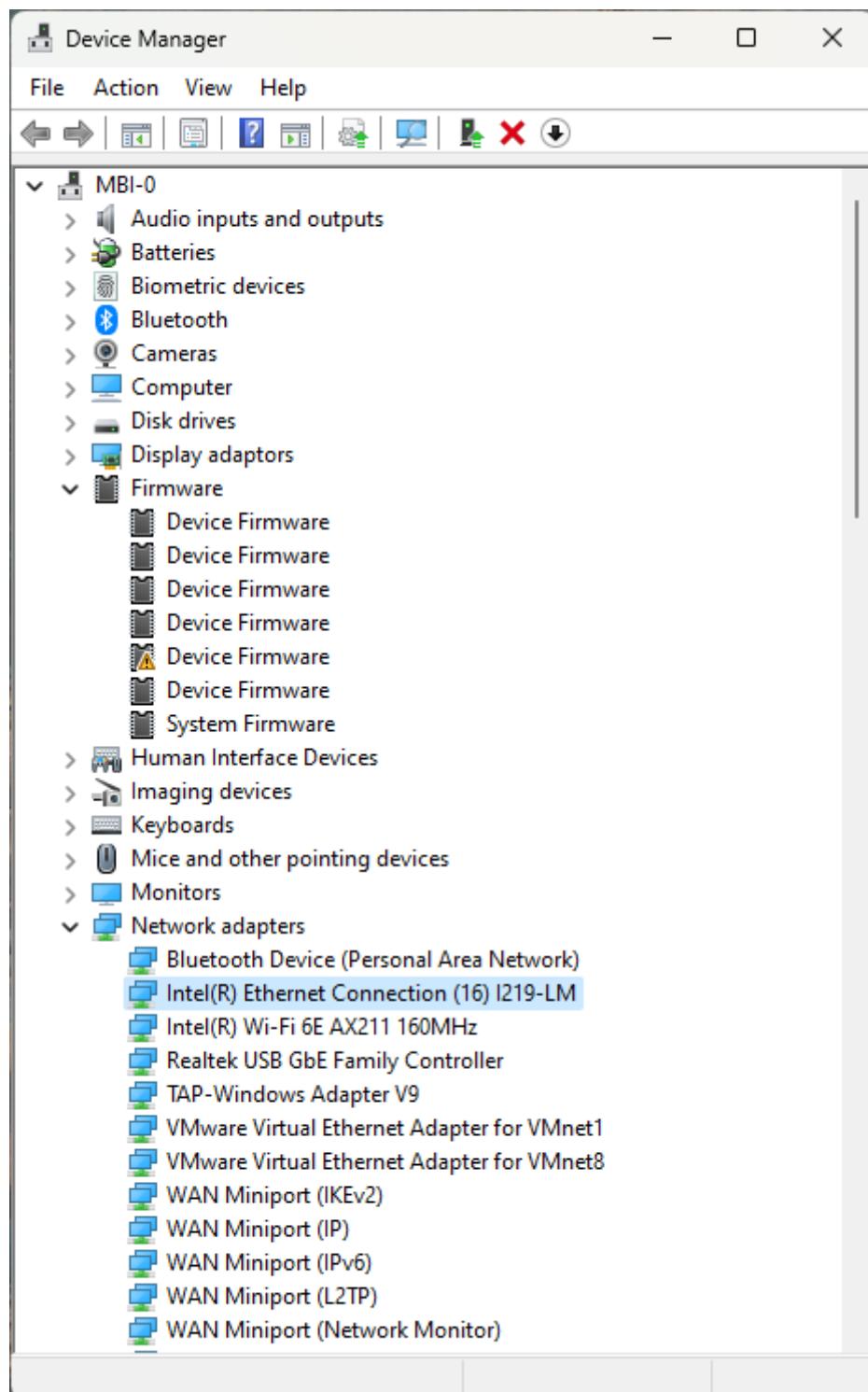
As alternative to the NDIS based or Pcap based Ethernet Driver, an optional Real-time Ethernet Driver on Windows can be installed. The Real-time Ethernet Driver replaces the original Windows driver and also requires an extra license.

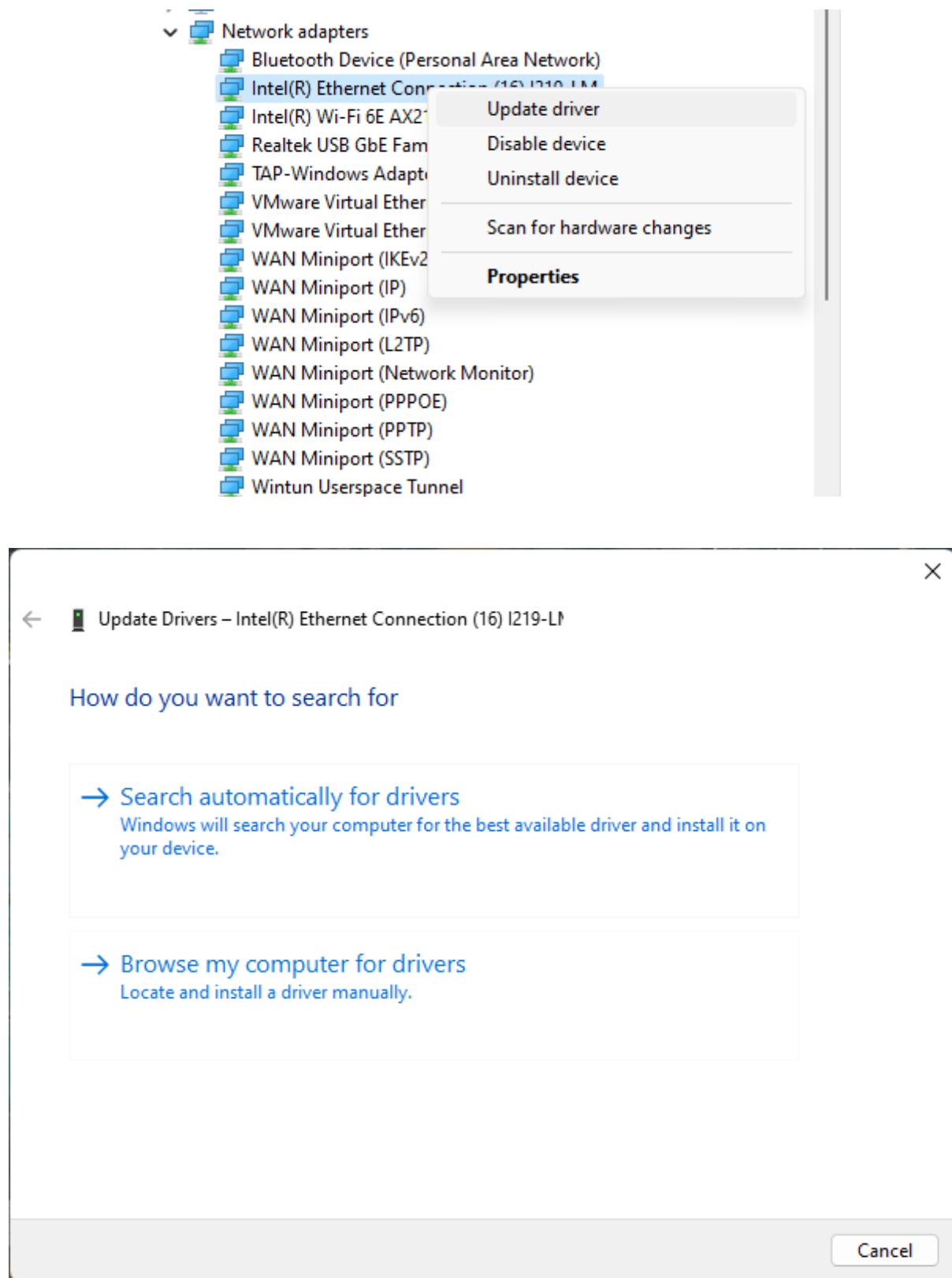
To use the Real-time Ethernet Driver under Windows, it is necessary to install the RtaccDevice driver included in the Real-time Ethernet Driver delivery:

1. Start the “Device Manager”

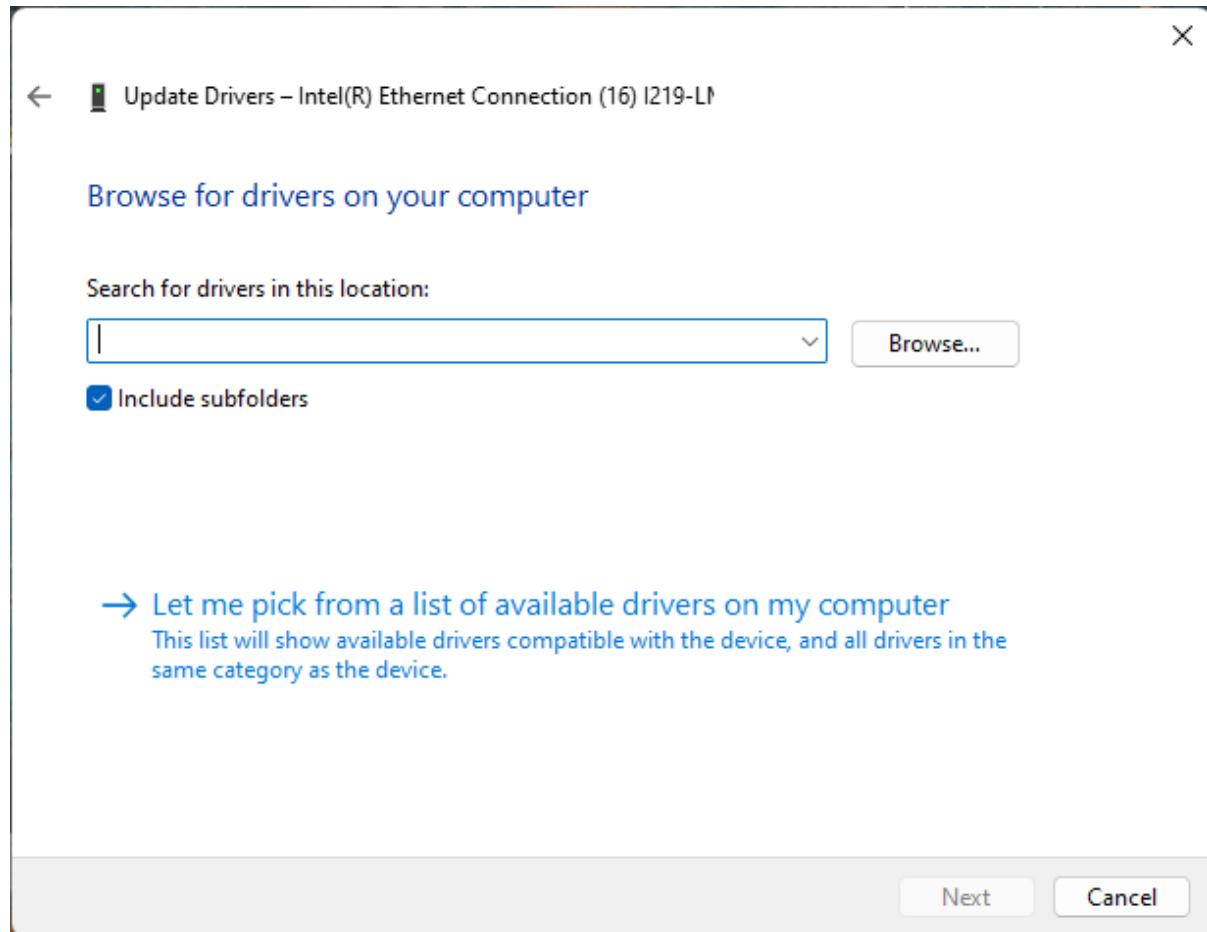


2. Assign RtaccDevice to the network adapter

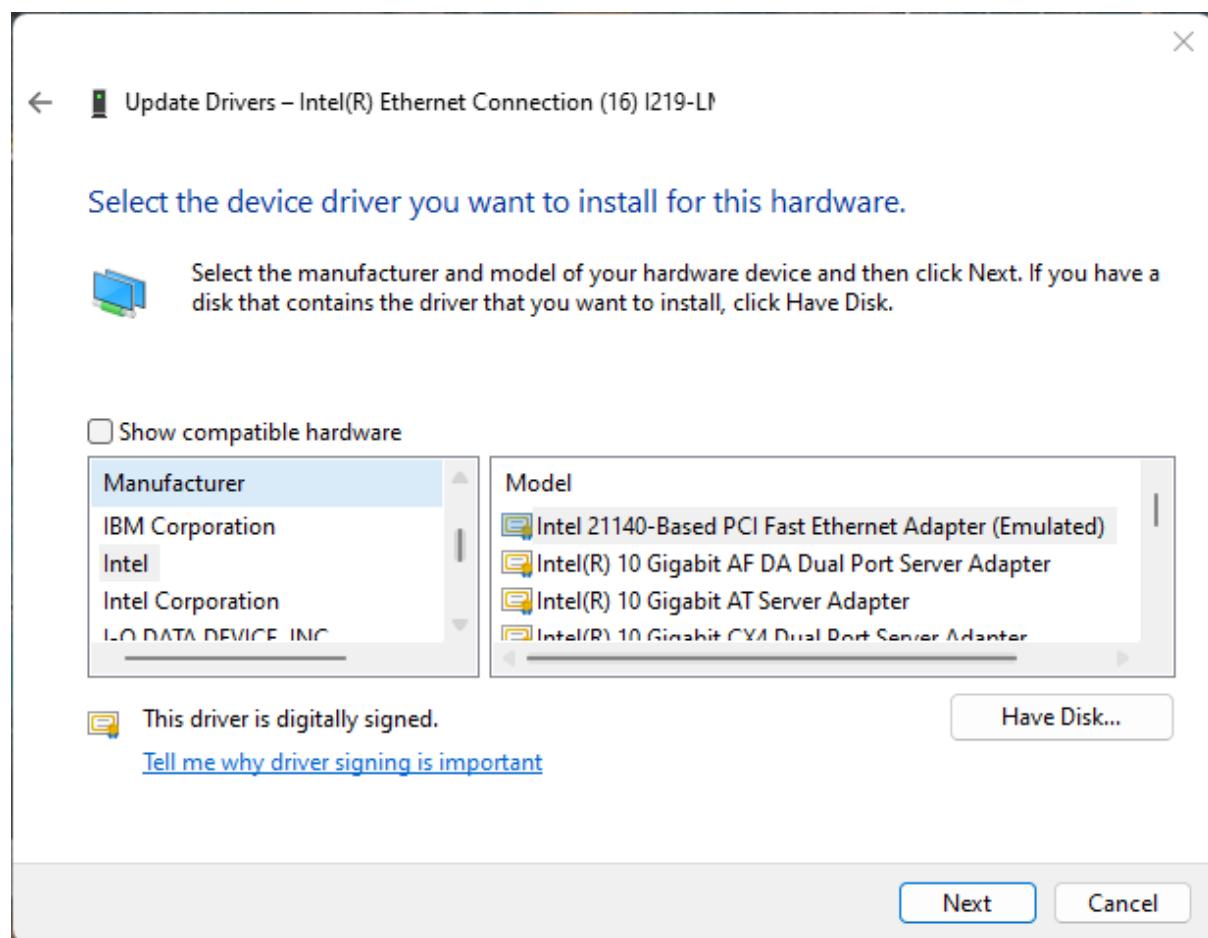




Click on “Browse my computer for driver”



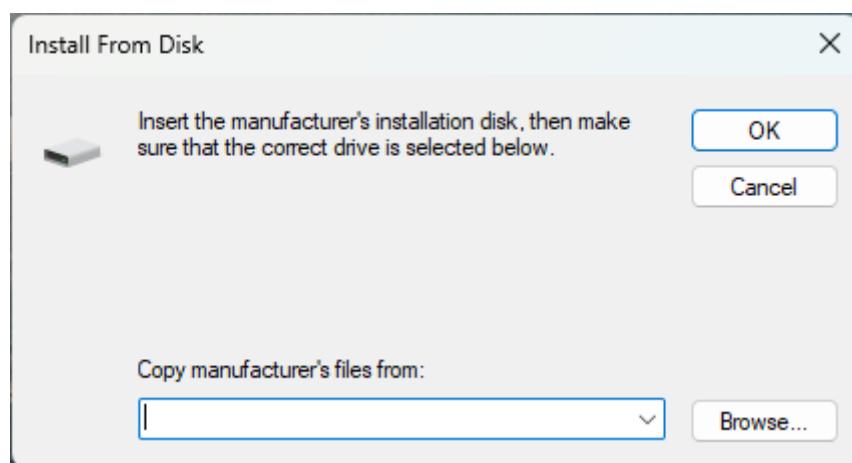
Click on “Let me pick...”



Click on “Have Disk...”

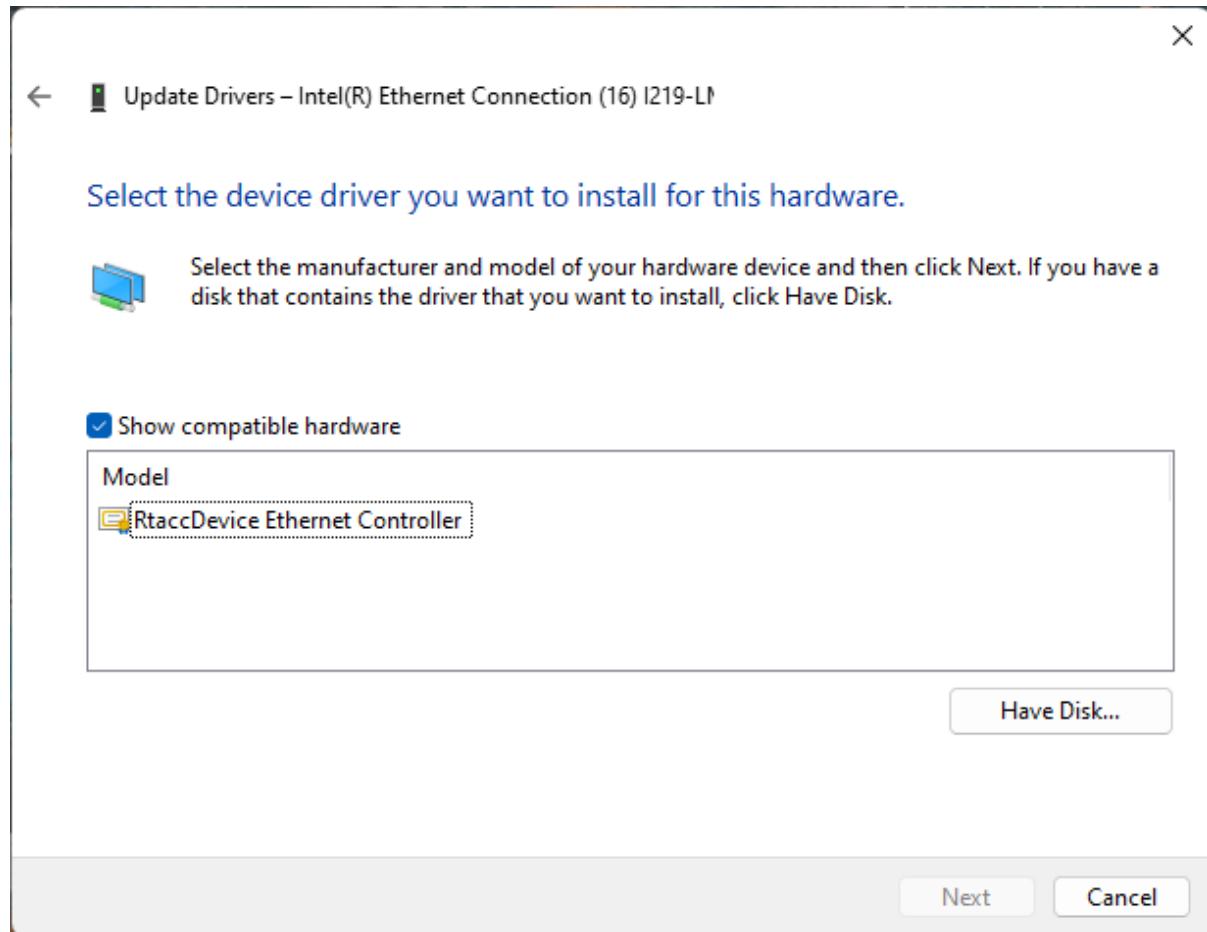
3. Enter the directory of RtaccDevice Driver

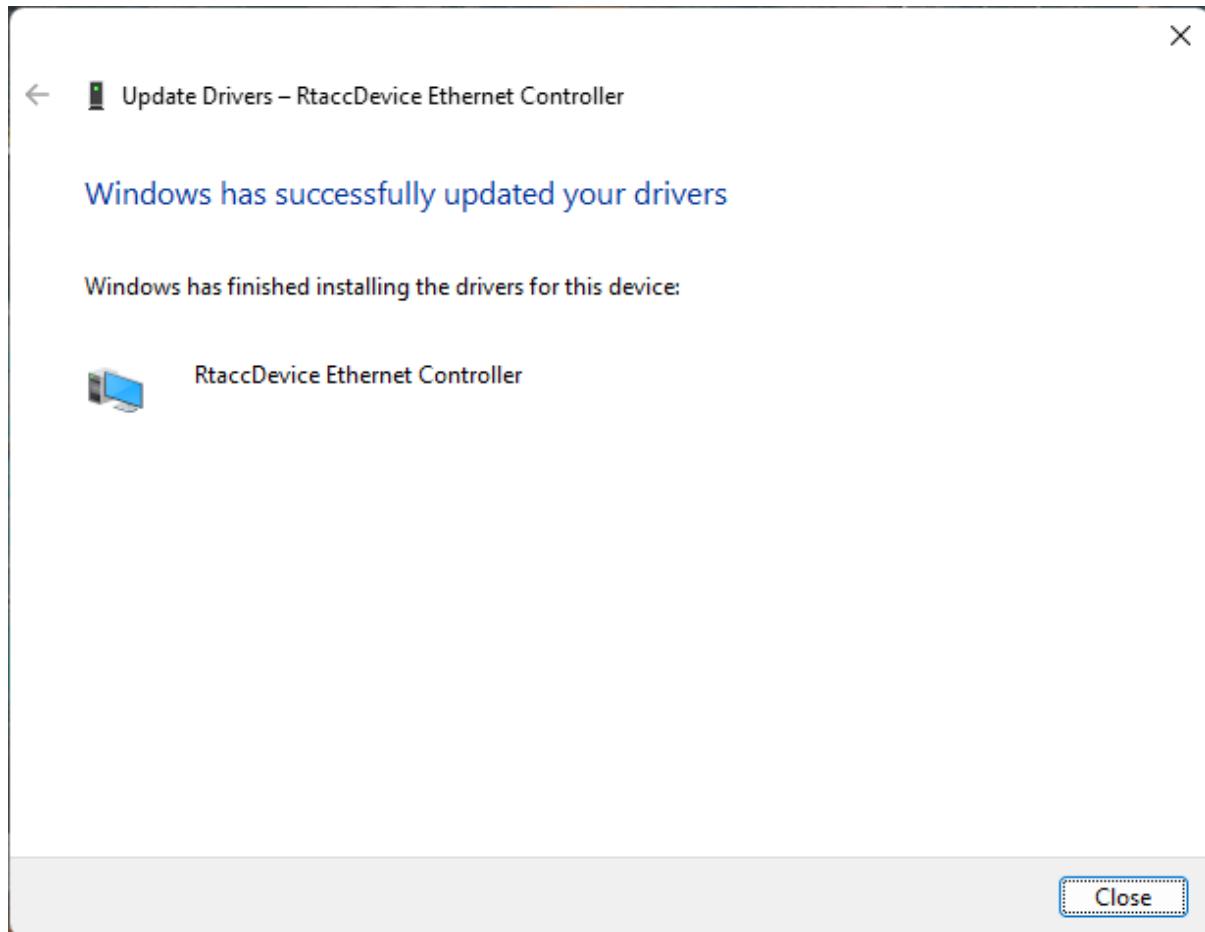
The default folder if not changed is <InstallPath>\Bin\Windows\x64



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

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



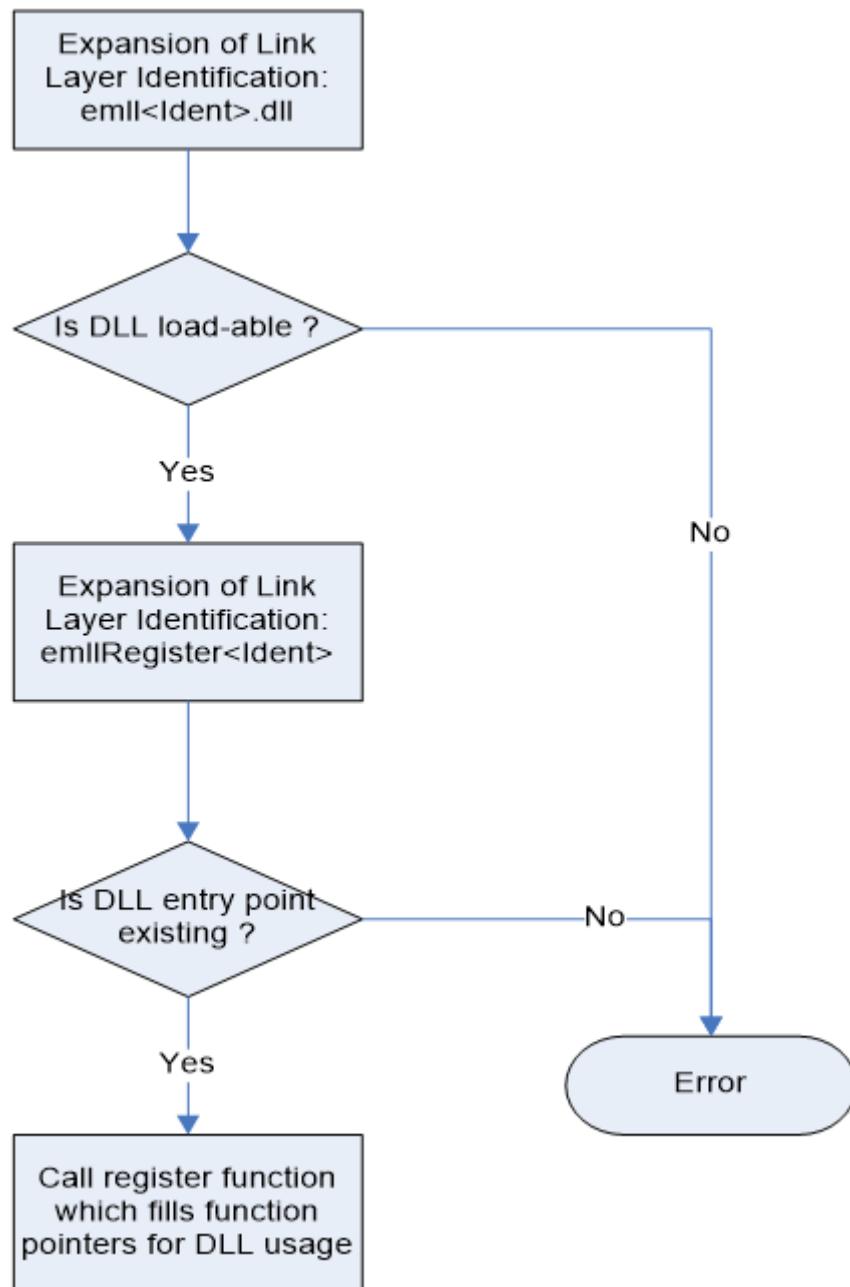


Optionally modify search location Real-time Ethernet Driver

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

## 4.17 Microsoft Windows CE

#### 4.17.1 Identification of the Real-time Ethernet Driver



The Real-time Ethernet Driver module DLL has to be locatable within the applications DLL search path (local or Windows directory). If it is not, an error is given.

#### 4.17.2 KUKA CeWin

For 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 a workaround is needed 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 Real-time Ethernet Driver 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.

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

---

#### Note:

1. **Due to a bug in Windows CE Version 5.0 a workaround is needed 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
  2. The images shipped with CeWin can be used together with the Intel PRO/100 Real-time Ethernet Driver
- 

For example to create a new Windows CE image which includes the PRO/100 Real-time Ethernet Driver the following files have to be included in the Windows CE OS-image:

- [...]/BIN/WinCE500/I8255x/x86/EcMaster.dll

- [...] /BIN/WinCE500/I8255x/CPU/emlli8255x.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

#### **4.17.3 Windows CE 5.0**

To be able to use the Real-time Ethernet Driver 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/emlli8255x.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 Real-time Ethernet Driver; i.e. use IntelGbe instead of I8255x. Search locations for Real-time Ethernet Driver can be adjusted using the PATH environment variable.

#### **4.17.4 Windows CE 6.0**

To be able to use the Real-time Ethernet Driver 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/emlli8255x.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 (merge into platform.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 Real-time Ethernet Driver; i.e. use IntelGbe instead of I8255x. Search locations for Real-time Ethernet Driver can be adjusted using the PATH environment variable.

#### **4.17.5 Windows CE 2013**

To be able to use the Real-time Ethernet Driver the following files have to be included to the Windows CE OS-image:  
Here the proceedings for Intel PRO/100

- [...] /BIN/ARM/WinCE800/EcMaster.dll
- [...] /BIN/ARM/WinCE800/emlli8255x.dll
- [...] /BIN/ARM/WinCE800/VirtualDrv.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/WinCE800/I8255x.reg

Same procedure and settings may be applied for the other Real-time Ethernet Driver; i.e. use IntelGbe instead of I8255x. Search locations for Real-time Ethernet Driver can be adjusted using the PATH environment variable. For built-in chips like FslFec the VirtualDrv.reg is used. Then rebuild is necessary.

## 4.17.6 Setting up and running EcMasterDemo

### 1. Windows CE configuration

See the section Operating system configuration for how to prepare the operating system

### 2. Determine the network interface

Using the command line option the network interface card and Real-time Ethernet Driver 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 Real-time Ethernet Driver (the first PCI device instance) and operate in polling mode.

### 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. Local IT infrastructure should not be mixed with EtherCAT modules at the same switch as the EC-Master will send many broadcast packets! EtherCAT requires a 100Mbit/s connection. If the network adapter card does not support this speed an Ethernet switch has to be used.

### 4. Copy the corresponding Real-time Ethernet Driver module from Bin/WINCE<version>/<arch>:

emllIntelGbe.dll (Intel Pro/1000)

emlliI8255x.dll (Intel Pro/100)

emllRTL8169.dll (Realtek RTL8169/8168/8111)

emllRTL8139.dll (Realtek RTL8139)

### 5. Copy the EC-Master dynamic libraries to the Windows CE target system:

EcMaster.dll (Master core library)

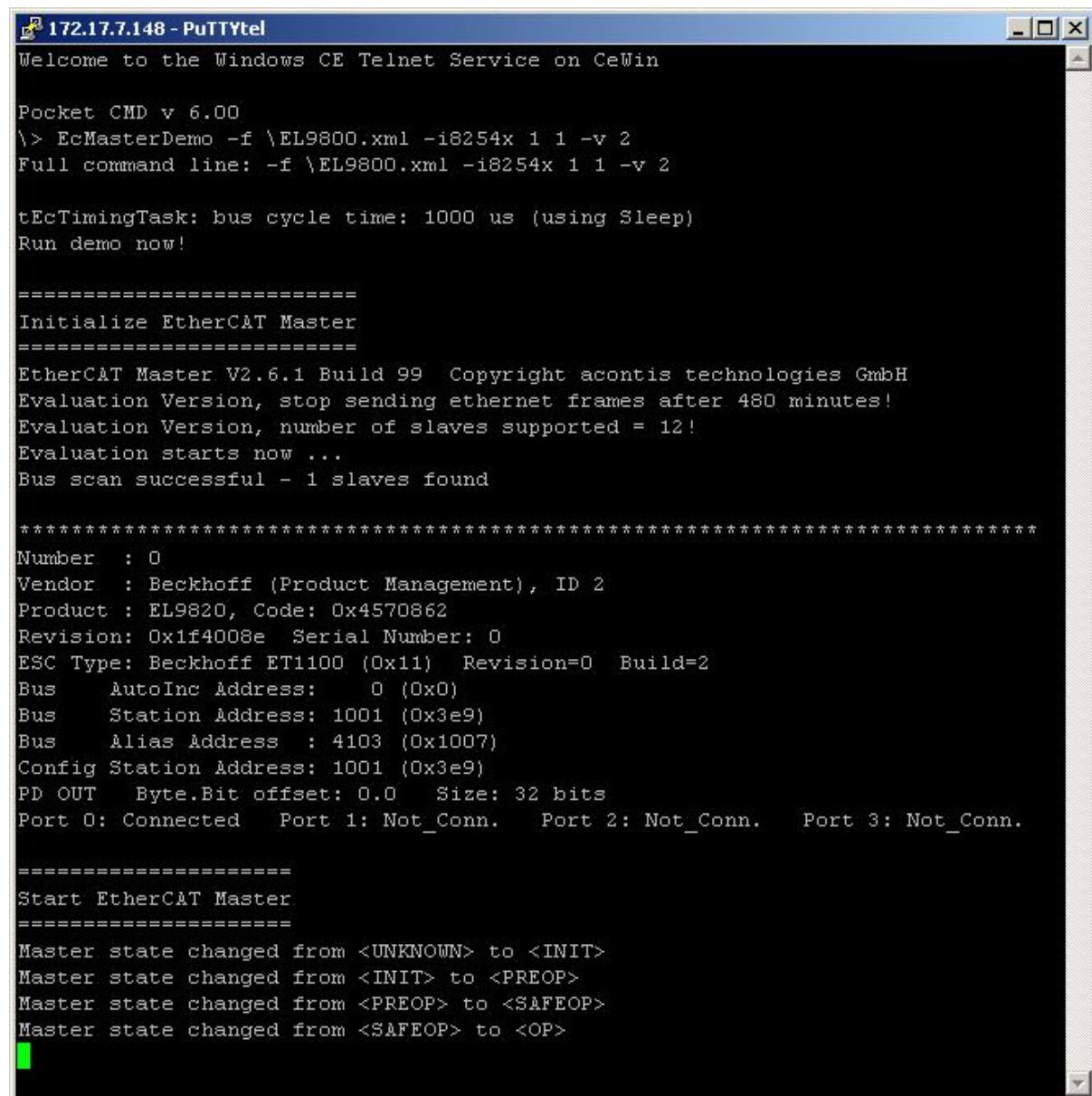
EcMasterRasServer.dll (Remote access service library if needed)

### 6. Copy one of the demo applications (EcMasterDemo, EcMasterDemoSyncSm, ...) from the EC-Master package to the Windows CE target system.

### 7. 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 Real-time Ethernet Driver. Example (starting the application on a network share via telnet):

```
> EcMasterDemo "-f ENI.xml -rtl8169 1 1"
```



The screenshot shows a PuTTY terminal window with the title "172.17.7.148 - PuTTYtel". The window displays the output of the EcMasterDemo application running on a Windows CE system. The output includes:

```
Welcome to the Windows CE Telnet Service on CeWin  
Pocket CMD v 6.00  
\> EcMasterDemo -f \EL9800.xml -i8254x 1 1 -v 2  
Full command line: -f \EL9800.xml -i8254x 1 1 -v 2  
  
tEcTimingTask: bus cycle time: 1000 us (using Sleep)  
Run demo now!  
  
=====  
Initialize EtherCAT Master  
=====  
EtherCAT Master V2.6.1 Build 99 Copyright acontis technologies GmbH  
Evaluation Version, stop sending ethernet frames after 480 minutes!  
Evaluation Version, number of slaves supported = 12!  
Evaluation starts now ...  
Bus scan successful - 1 slaves found  
  
*****  
Number : 0  
Vendor : Beckhoff (Product Management), ID 2  
Product : EL9820, Code: 0x4570862  
Revision: 0x1f4008e Serial Number: 0  
ESC Type: Beckhoff ET1100 (0x11) Revision=0 Build=2  
Bus AutoInc Address: 0 (0x0)  
Bus Station Address: 1001 (0x3e9)  
Bus Alias Address : 4103 (0x1007)  
Config Station Address: 1001 (0x3e9)  
PD OUT Byte.Bit offset: 0.0 Size: 32 bits  
Port 0: Connected Port 1: Not_Conn. Port 2: Not_Conn. Port 3: Not_Conn.  
  
=====  
Start EtherCAT Master  
=====  
Master state changed from <UNKNOWN> to <INIT>  
Master state changed from <INIT> to <PREOP>  
Master state changed from <PREOP> to <SAFEOP>  
Master state changed from <SAFEOP> to <OP>
```

#### See also:

[Running EcMasterDemo](#)

#### 4.17.7 OS Compiler settings

Besides the general settings from [Compiling the EcMasterDemo](#) the following settings are necessary to build the example application for Windows.

##### Extra include paths

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

##### Extra source paths

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

### Extra library paths to the main EtherCAT components

```
<InstallPath>/SDK/LIB/WinCE
```

### Extra libraries (in this order)

```
coredll.lib corelibc.lib EcMaster.lib EcMasterRasServer.lib
```

### Preprocessor definitions

```
CEWIN if running on acontis EC-WinCE.
```

- Don't "Treat wchar\_t as Built-in Type"

### Entry Point:

```
mainWCRTStartup
```

## 4.18 Xenomai

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

### See also:

Chapter [Linux](#)

The binaries are built using the following versions:

- **armv6-vfp-eabihf:**
  - Xenomai 2.6.3, tested on Linux Kernel 3.8.13-xenomai-2.6.4
- **x64:**
  - Xenomai 3.0.2, tested on Linux Kernel 3.18.20 (Cobalt)
  - EVL r0.44 (Xenomai 4), tested on Linux Kernel 5.15.106
- **x86:**
  - Xenomai 2.6.2.1, tested on Linux Kernel 3.5.7
  - Xenomai 3.0.2, tested on Linux Kernel 3.18.20 (Cobalt) and 3.10.32-rt31 (Mercury)

## 4.18.1 Setting up and running EcMasterDemo

### 1. Prepare system

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

### 2. Compile EcMasterDemo

As a starting point there is the Eclipse project for EcMasterDemo for Xenomai located at `Workspace/Xenomai/`. Ensure `OPERATING_SYSTEM`, `ARCH`, `CFLAGS`, `LDFLAGS`, `LD_LIBRARY_PATH` are set accordingly (`export ARCH=x86, ...`) when compiling using Eclipse! For Xenomai 4 the environment variable `ELV_PATH` should contain path of the libevl. Also ensure the following define is present:

```
EC_VERSION_XENOMAI=4
```

### 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 -intelgbe 2 1 -f eni.xml -v 3
$ [...]
$ (gdb) handle SIGXCPU nostop noprint nopass
$ (gdb) run
```

#### See also:

[Running EcMasterDemo](#)

## 4.18.2 OS compiler settings

Besides the general settings from [Compiling the EcMasterDemo](#) the following settings are necessary to build the example application for Xenomai.

### Extra include paths

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

### Extra source paths

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

### Extra library paths to the main EtherCAT components

- Xenomai 2 and 3:

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

- Xenomai 4:

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

### Extra libraries (in this order)

- Xenomai 2:

```
EcMasterRasServer EcMaster pthread dl rt native xenomai
```

- Xenomai 3:

```
EcMasterRasServer EcMaster pthread dl rt
```

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

- Xenomai 4:

```
EcMasterRasServer EcMaster pthread evl dl
```

## 4.19 Zephyr

### 4.19.1 Setting up and running EcMasterDemo

#### 1. Prerequisites

- Up Squared board
- Docker

#### 2. Clone the Zephyr repository and checkout the sha: d489765be4e57ca0d836d391dbda23284ac09e7f

```
git clone https://github.com/zephyrproject-rtos/zephyr
cd zephyr
git checkout d489765be4e57ca0d836d391dbda23284ac09e7f
```

#### 3. Download the latest Docker container of the build environment

```
docker pull zephyrprojectrtos/zephyr-build:latest
```

#### 4. Start the Docker Container (On Windows make sure to use absolute paths with forward slashes)

```
docker run -ti -v <ZEPHYR_REPO_PATH>:/workdir -v
→ <EC_MASTER_BASE_PATH>:/Master zephyrprojectrtos/zephyr-build:latest
```

#### 5. Inside the Container change the directory

```
/Master/Workspace/Zephyr/EcMasterDemo
```

#### 6. Build the EcMasterDemo

```
mkdir build && cd build
cmake .. -DBOARD=up_squared -DRELEASE_MODE=Release
make install
```

#### 7. The stripped project file can be found in

```
<EC_MASTER_BASE_PATH>/Bin/Zephyr/x64/Release/EcMasterDemo.strip
```

#### 8. To run the demo place the stripped project file on the Up Squared board and connect to the serial console on UART 1. After booting into the Application it will prompt for command line arguments on the serial console.

#### See also:

[Running EcMasterDemo](#)

## 4.19.2 OS Compiler settings

Besides the general settings from [Compiling the EcMasterDemo](#) the following settings are necessary to build the example application for Zephyr.

### Extra include paths

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

### Extra source paths

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

### Extra library paths to the main EtherCAT components

```
<InstallPath>/SDK/LIB/Zephyr
```

## 5 Real-time Ethernet Driver

The EtherCAT master stack currently supports a variety of different Real-time Ethernet Driver modules, each of which contained in a single library file, which is loaded by the core library dynamically. The EtherCAT master stack shipment consist of a master core library (e.g. EcMaster.dll for Windows, libEcMaster.a for Linux) and one (or more) libraries each containing support for one specific Real-time Ethernet Driver module. Which library actually is loaded, is depending on the Real-time Ethernet Driver parameters at runtime.

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

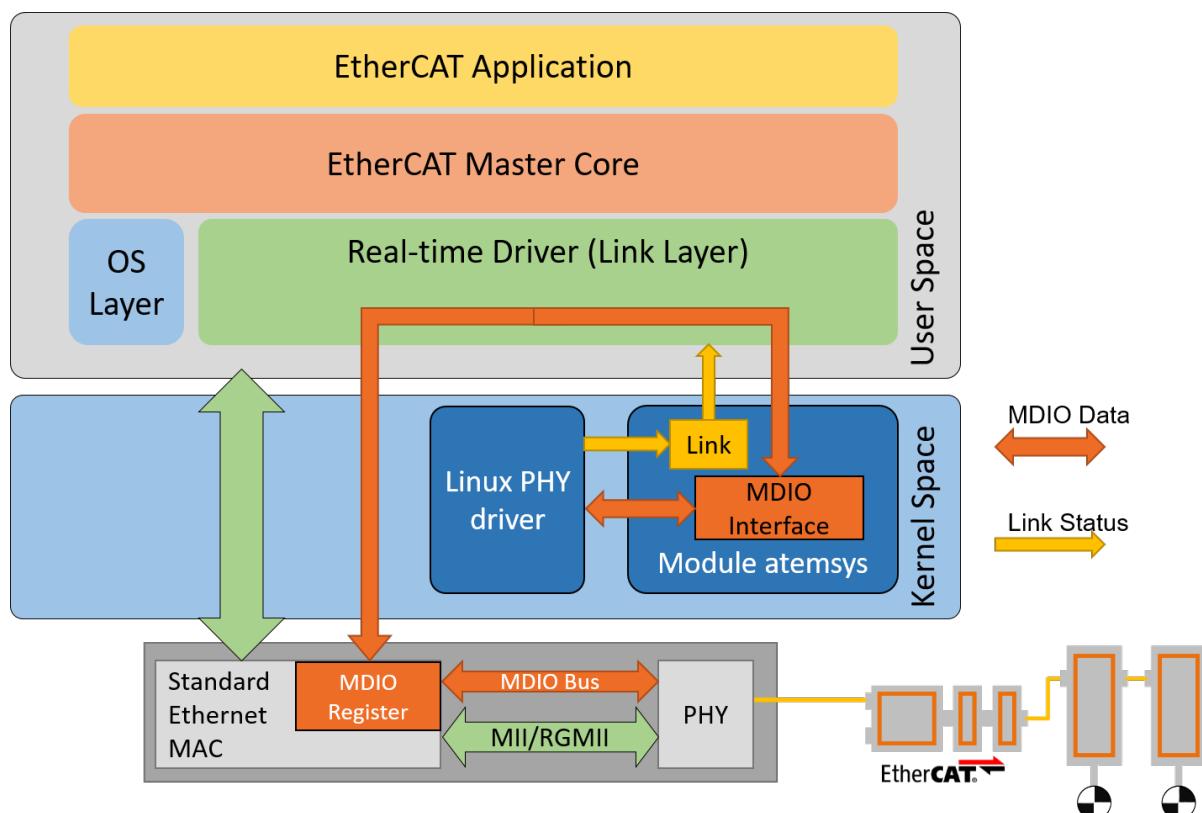
The principle of the Real-time Ethernet Driver selection is that the Real-time Ethernet Driver name (Real-time Ethernet Driver identification) is used to determine the location and name of a registration function called by the EtherCAT master and registers function pointers that allow access to the Real-time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

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

The EtherCAT master supports two Real-time Ethernet Driver operating modes. If the Real-time Ethernet Driver operates in interrupt mode all received Ethernet frames will be processed immediately in the context of the Real-time Ethernet Driver receiver task. When using the polling mode the EtherCAT master will call the Real-time Ethernet Driver receiver polling function prior to processing received frames.

### Real-time Ethernet Driver and PHY OS Driver

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



---

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

---

## 5.1 Real-time Ethernet Driver initialization

The different Real-time Ethernet Driver modules are selected and parameterized by a Real-time Ethernet Driver specific structure. Each Real-time Ethernet Driver specific structure start with a common *EC\_T\_LINK\_PARMS* structure, followed by some Real-time Ethernet Driver specific members. The common link parameter structure is passed to *EC\_T\_INIT\_MASTER\_PARMS::pLinkParms* with the call of *emInitMaster()* like in the following example:

```

/* identify Link Layer in the common struture */
oLinkParmsSockRaw.linkParms.dwSignature = EC_LINK_PARMS_SIGNATURE_SOCKRAW;
oLinkParmsSockRaw.linkParms.dwSize = sizeof(EC_T_LINK_PARMS_SOCKRAW);
OsStrncpy(&oLinkParmsSockRaw.linkParms.szDriverIdent, EC_LINK_PARMS_IDENT_SOCKRAW,
          EC_DRIVER_IDENT_MAXLEN);

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

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

/* more parameters should be set here */

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

```

struct **EC\_T\_LINK\_PARMS**

### Public Members

#### **EC\_T\_DWORD dwSignature**

[in] Signature of the adapter specific structure containing the *EC\_T\_LINK\_PARMS* structure

#### **EC\_T\_DWORD dwSize**

[in] Size of the adapter specific structure containing the *EC\_T\_LINK\_PARMS* structure

#### *EC\_T\_LOG\_PARMS* **LogParms**

[in] Logging parameters

#### **EC\_T\_CHAR szDriverIdent[EC\_DRIVER\_IDENT\_NAMESIZE]**

[in] Name of Link Layer module (driver identification) for Link Layer Selection

#### **EC\_T\_DWORD dwInstance**

[in] Instance of the adapter. if EC\_LINKUNIT\_PCILOCATION is set: contains PCI address

#### *EC\_T\_LINKMODE* **eLinkMode**

[in] Mode of operation

#### **EC\_T\_CPUSET cpuIstCpuAffinityMask**

[in] Interrupt service thread CPU affinity mask

**EC\_T\_DWORD dwIstPriority**

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

**EC\_T\_DWORD dwIstStackSize**

[in] Task stack size

**EC\_T\_DWORD dwLinkSpeed**

[in] 10, 100, 1000 Mbit\`s

***EC\_T\_LINKLAYER\_TIMINGTASK* oLinkLayerTimingTask**

[in] LinkLayer timimg task parameters

**EC\_T\_CHAR szLoadPath[EC\_DRIVER\_PATH\_MAXLEN]**

[in] path from which the libraries should be loaded

enum **EC\_T\_LINKMODE**

*Values:*

enumerator **EcLinkMode\_UNDEFINED**

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

enumerator **EcLinkMode\_INTERRUPT**

Link is in interrupt mode and is triggered by interrupts

enumerator **EcLinkMode\_POLLING**

Link is in polling mode and is polled periodically

struct **EC\_T\_LINKLAYER\_TIMINGTASK****Public Members*****EC\_T\_LINKLAYER\_TIMING* eLinkLayerTiming**

[in] LinkLayer timing task mode

**EC\_T\_DWORD dwCycleTimeUsec**

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

**EC\_T\_LINK\_STARTCYCLE\_CALLBACK pfnStartCycle**

[in] Callback function called cyclically according dwCycleTimeUsec

**EC\_T\_VOID \*pvStartCycleContext**

[in] Context passed to each pfnStartCycle call

**EC\_T\_DWORD dwTtsSendOffsetUsec**

[in] Time between pfnStartCycle call and TTS frame transmission

**EC\_T\_UINT64 nSystemTime**

[in] System

enum **EC\_T\_LINKLAYER\_TIMING**

*Values:*

enumerator **eLinkLayerTiming\_Undefined**  
 Link Layer Timing is undefined must be TTS or TMR

enumerator **eLinkLayerTiming\_TTS**  
 Real-time Ethernet Driver Time Triggered Send

enumerator **eLinkLayerTiming\_TMR**  
 Real-time Ethernet Driver Timer

enum **EC\_T\_PHYINTERFACE**

*Values:*

enumerator **ePHY\_UNDEFINED**  
 undefined

enumerator **ePHY\_FIXED\_LINK**  
 No PHY access at all

enumerator **ePHY\_MII**  
 MII 10 / 100 MBit

enumerator **ePHY\_RMII**  
 Reduced MII, 10 / 100 MBit

enumerator **ePHY\_GMII**  
 Gigabit MII, 10, 100, 1000 MBit

enumerator **ePHY\_SGMII**  
 Serial (SERDES) Gigabit MII, 10, 100, 1000 MBit

enumerator **ePHY\_RGMII**  
 Reduced Gigabit MII, 10, 100, 1000 MBit

enumerator **ePHY\_OSDRIVER**  
 Get interface type from OS

enumerator **ePHY\_RMII\_50MHZ**  
 ePHY\_RMII with 50 MHz clock mode

### 5.1.1 Real-time Ethernet Driver instance selection via PCI location

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

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

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

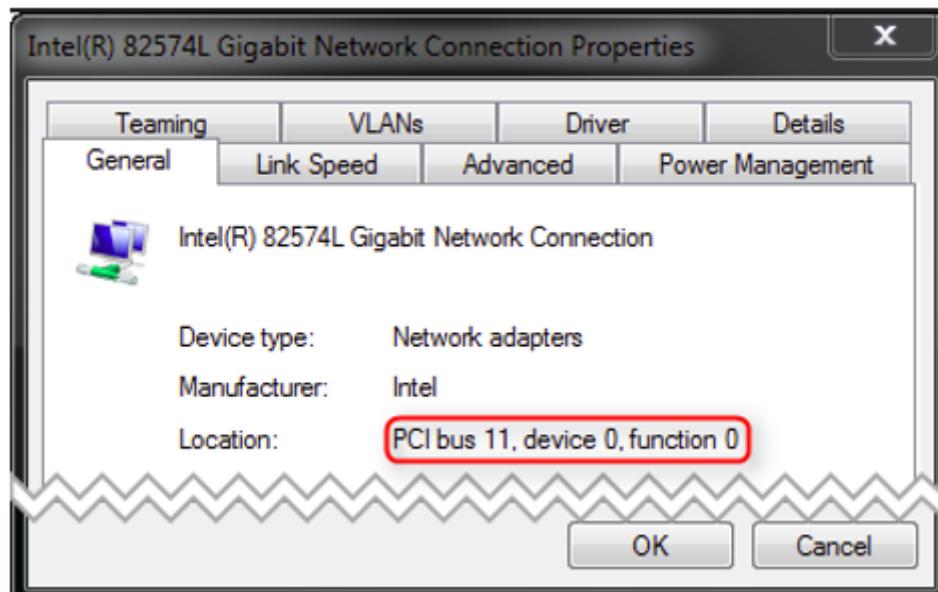
The format of *EC\_T\_LINK\_PARMS::dwInstance* using PCI bus address is:

**0x01bbddff**

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

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

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



## 5.2 Broadcom Genet - emlIBcmGenet

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

```
struct EC_T_LINK_PARMS_BCMGENET
```

### Public Members

`EC_T_LINK_PARMS linkParms`

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

`EC_T_BCMGENET_TYPE eSocType`

Broadcom processor type

`EC_T_BOOL bNotUseDmaBuffers`

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

enum `EC_T_BCMGENET_TYPE`

Values:

enumerator **eBCMGENET\_BCM2711**  
Broadcom BCM2711, Raspberry Pi 4

## 5.3 Broadcom NetXtreme - emlIBcmNetXtreme

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

```
struct EC_T_LINK_PARMS_BCMNETXTREME
```

### Public Members

**EC\_T\_LINK\_PARMS linkParms**

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

**EC\_T\_DWORD dwRxBuffers**

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

**EC\_T\_DWORD dwTxBuffers**

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

### 5.3.1 Supported PCI devices

Broadcom 571x PCI specific definitions (VendorId, DeviceId)

aggedright

- **PCI\_DEVICE\_BCM5717** (0x14E4, 0x1655)

- **PCI\_DEVICE\_BCM571X** (0x14E4, 0x1656)
- **PCI\_DEVICE\_BCM5719** (0x14E4, 0x1657)
- **PCI\_DEVICE\_BCM5720** (0x14E4, 0x1656)

## 5.4 Berkeley Packet Filter - emlIBPF

The Ethernet Driver BPF is always part of the EC-Master for QNX package. It uses already established Ethernet adapters, e.g. `wm0`, `rt0`, etc. It is strongly recommended to use a separate network adapter to connect EtherCAT devices. If the main network adapter is used for both EtherCAT devices and the local area network there may be a main impact on the local area network operation.

---

**Note:** The BPF Ethernet Driver cannot be used for real time applications and may need cycle time of 1 ms or higher.

---

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

```
struct EC_T_LINK_PARMS_BPF
```

**Public Members*****EC\_T\_LINK\_PARMS* *linkParms***

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_BPF

**EC\_T\_CHAR *szIfName*[EC\_BPF\_IF\_NAME\_SIZE]**

Name of Ethernet Interface

**EC\_T\_CHAR *szPrefix*[EC\_BPF\_IF\_NAME\_SIZE]**

Optional prefix of the BPF instance path

**EC\_T\_DWORD *dwRxBuffers***

Receive buffer count

## 5.5 Beckhoff CCAT - emIICCAT

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

```
struct EC_T_LINK_PARMS_CCAT
```

**Public Members*****EC\_T\_LINK\_PARMS* *linkParms***

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_CCAT

***EC\_T\_CCAT\_TYPE* *eCcatType***

CCAT connection type

**EC\_T\_UINT64 *qwCcatBase***

Physical address of register block, only for eCCAT\_EIM

**EC\_T\_DWORD *dwCcatSize***

Size of register block, only for eCCAT\_EIM

**EC\_T\_DWORD *dwRxBufferCnt***

Receive buffer count, only for eCCAT\_EIM

**EC\_T\_DWORD *dwTxBufferCnt***

Transmit buffer count, only for eCCAT\_EIM

**enum *EC\_T\_CCAT\_TYPE***

*Values:*

**enumerator *eCCAT\_PCI***

CCAT connected to PCI bus

**enumerator *eCCAT\_EIM***

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

### 5.5.1 Supported PCI devices

Beckhoff CCAT PCI specific definitions (VendorId, DeviceId)

aggedright

- **PCI\_DEVICE\_CCAT** (0x15EC, 0x2600)

## 5.6 CMSIS - emIICmsisEth

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

```
struct EC_T_LINK_PARMS_CMSIETH
```

### Public Members

**`EC_T_LINK_PARMS linkParms`**

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

**`EC_T_DWORD dwRxBuffersCnt`**

Receive buffer count

**`EC_T_DWORD dwRxBufferLen`**

Receive buffer size for a single Ethernet frame.

**`EC_T_DWORD dwTxBuffersCnt`**

Transmit buffer count

## 5.7 Texas Instruments CPSW - emIICPSW

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

```
struct EC_T_LINK_PARMS_CPSW
```

### Public Members

**`EC_T_LINK_PARMS linkParms`**

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

**`EC_T_CPSW_TYPE eCpswType`**

CPSW type

**`EC_T_DWORD dwPhyAddr`**

PHY address

**`EC_T_DWORD dwPortPrio`**

0 (lowest), 1 (highest)

**EC\_T\_BOOL bMaster**

EC\_TRUE: Initialize MAC

**EC\_T\_BOOL bPhyRestartAutoNegotiation**

EC\_TRUE: Restart auto negotiation on initialization

***EC\_T\_PHYINTERFACE* ePhyInterface**

PHY connection type

**EC\_T\_DWORD dwRxInterrupt**

Receive interrupt number (IRQ)

**EC\_T\_BOOL bNotUseDmaBuffers**EC\_FALSE: Use buffers from DMA for receive (default), EC\_TRUE: use buffers from heap for receive.  
EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC\_TRUE.**EC\_T\_BOOL bNoPhyAccess**

Don't use MDIO to set up the PHY

**EC\_T\_BOOL bAleBypass**

Enable bAleBypass, similar to promiscuous mode

**EC\_T\_DWORD dwExtendRxFrameLength**

Receive longer Rx Frames

enum **EC\_T\_CPSW\_TYPE**

Values:

enumerator **eCPSW\_AM33XX**

TI AM33xx (e.g. Beaglebone)

enumerator **eCPSW\_AM387X**

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

enumerator **eCPSW\_AM437X**

TI AM437x

enumerator **eCPSW\_AM57X**

TI AM57x

### 5.7.1 CPSW usage under Linux

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

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

Currently following Linux versions are supported:

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

---

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

---

The patch needs:

- Linux kernel with enabled CPSW driver.
- Patch applied to Linux kernel.
- EC\_ETHERNET\_PORT defined according to target in cswc.c and davinci\_mdio.c files.
- Kernel must be rebuilt and installed

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

---

**Note:** EtherCAT ports should be used as “slave” since “master” is the Linux driver.

---

## 5.8 Texas Instruments CPSWG for AM6x and Jacinto 7 - emIICP-SWG

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

struct **EC\_T\_LINK\_PARMS\_CPSWG**

### Public Members

**EC\_T\_LINK\_PARMS** **linkParms**

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

**EC\_T\_DWORD** **dwPhyAddr**

PHY address

**EC\_T\_PHYINTERFACE** **ePhyInterface**

PHY interface type

**EC\_T\_BOOL** **bMaster**

`EC_TRUE`: Initialize MAC

**EC\_T\_DWORD** **dwTxDmaDesCnt**

Transmit DMA descriptor buffer count. maximum 500

**EC\_T\_DWORD** **dwRxDmaDesCnt**

Receive DMA descriptor buffer count. maximum 500

**EC\_T\_BOOL** **bNotUseDmaBuffers**

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

enum **EC\_T\_CPSWG\_TYPE**

Values:

enumerator **eCP SWG\_AM654X**  
TI AM654x

enumerator **eCP SWG\_TDA4X**  
TI TDA4x (Jacinto 7)

enumerator **eCP SWG\_AM64X**  
TI AM64x

enumerator **eCP SWG\_AM62X**  
TI AM62x

### 5.8.1 CPSWG usage under Linux

The unbind feature of the am65-cpsw-nuss Linux driver isn't supported and the atemsys kernel module must be assigned as driver for the platform device. So the "ethernet/am654-cpsw-nuss" device tree node must be modified, by changing compatible = "atemsys"; and adding atemsys-Ident = "CPSWG"; and atemsys-Instance = <0x1>;, also remove or comment out dma-coherent;.

```
ethernet@46000000 {
    compatible = "atemsys";
    atemsys-Ident = "CPSWG";
    atemsys-Instance = <0x1>;
    ...
    #dma-coherent;
    ...
}
```

So the "ethernet/am642-cpsw-nuss" device tree node must be modified likewise. Second port of am642-cpsw-nuss is not supported.

```
ethernet@8000000 {
    compatible = "atemsys";
    atemsys-Ident = "CPSWG";
    atemsys-Instance = <0x1>;
    ...
    #dma-coherent;
    ...
}
```

Currently following Linux versions are supported:

- ti-processor-sdk-linux-rt-am65xx-evm-07\_01\_00\_18 (tested on AM65 IDK)
- ti-processor-sdk-linux-rt-am65xx-evm-08.06.00.47 (tested on AM65 IDK)
- ti-processor-sdk-linux-rt-j7-evm-08\_06\_01\_02 (tested on SK-TDA4VM)
- ti-processor-sdk-linux-rt-j7-evm-08\_06\_01\_02 (tested on SK-TDA4VM)
- ti-processor-sdk-linux-edgeai-j721e-evm-09\_01\_00\_06 (tested on SK-TDA4VM)
- ti-processor-sdk-linux-rt-am64xx-evm-08.02.00.14 (tested on TMDS64GPEVM)
- ti-processor-sdk-linux-rt-am64xx-evm-09.02.00.08 (tested on TMDS64GPEVM)
- SolidRun ti\_am64x\_build/20240221 microsd-debian-bookworm-sr1 (tested on AM64 HummingBoard-T)
- toradex\_ti-linux-6.1.y Linux/arm64 6.1.46 Kernel (tested on Toradex Verdin AM62)

## 5.9 Linux DPDK - emllDpdk

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

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

```
struct EC_T_LINK_PARMS_DPDK
```

### Public Members

#### `EC_T_LINK_PARMS linkParms`

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

#### `EC_T_DWORD dwPortId`

DPDK port ID

#### `EC_T_WORD wRxBufferCnt`

Receive buffer count, 0: defaults to DPDK internal setting

#### `EC_T_WORD wTxBufferCnt`

Transmit buffer count, 0: default to DPDK internal setting

#### `EC_T_BOOL bDontCheckLinkStatus`

Don't check link status (forced)

#### `EC_T_BOOL bSetPromiscuousMode`

Change promiscuous mode setting

#### `EC_T_BOOL bEalInitDeinitByApp`

DPDK EAL layer is initialized and deinitialized by application

### Note:

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

The network interface must be bound according to the interface type PCI or SOC:

### 5.9.1 DPDK for PCI device

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

```
$ sudo modprobe uio_pci_generic
```

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

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

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

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

For each port to be bound to emllDpdk, call

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

Example:

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

Check [https://doc.dpdk.org/guides/linux\\_gsg/linux\\_drivers.html](https://doc.dpdk.org/guides/linux_gsg/linux_drivers.html) for more information on the drivers.

## 5.9.2 DPDK for DPAA

### Port specific numbering

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

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

DPDK	dts
0	1
1	5
2	6
3	10

### Reassigning Ports to DPAA and Linux in DTB/DTS

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

---

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

---

The section dpaa-extended-args must be extended:

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

    dpaa-extended-args {

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

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

        fman0_tx0-extd-args {
```

(continues on next page)

(continued from previous page)

```

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

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

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

fman1-extd-args {
    cell-index = <1>;
    compatible = "fsl,fman-extended-args";
    dma-aid-mode = "port";
    fman1_rx0-extd-args {
        cell-index = <0>;
        compatible = "fsl,fman-port-1g-rx-extended-args";
        vsp-window = <8 0>;
    };
    fman1_tx0-extd-args {
        cell-index = <0>;
        compatible = "fsl,fman-port-1g-tx-extended-args";
    };
    fman1_rx1-extd-args {
        cell-index = <1>;
        compatible = "fsl,fman-port-1g-rx-extended-args";
        vsp-window = <8 0>;
    };
    fman1_tx1-extd-args {
        cell-index = <1>;
        compatible = "fsl,fman-port-1g-tx-extended-args";
    };
};
};
};
```

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

```
fsl,dpaa {
    compatible = "fsl,ls1046a\\0fsl,dpaa\\0simple-bus";
    dma-coherent;
    phandle = <0x85>;
    /* assign Port 1 (ethernet@0) to em11Dpaa (all ports assigned to em11Dpaa must
     * be contiguous at the start) */

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

---

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

}
;
```

### 5.9.3 Linux System Requirements

- Kernel configuration

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

```
HUGETLBFS
PROC_PAGE_MONITOR support
```

- Required Tools, Libraries and further system requirements can be checked under [https://doc.dpdk.org/guides/linux\\_gsg/sys\\_reqs.html](https://doc.dpdk.org/guides/linux_gsg/sys_reqs.html)

### 5.9.4 Huge pages setup

Create huge pages (not persistent)

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

### 5.9.5 Limitations

Some parts of the DPDK library can only be initialized once per process. If the adapter needs to be used multiple times, for cable redundancy or calling emInitMaster() multiple times, the application must set EC\_T\_LINK\_PARMS\_DPDK::bEalInitDeinitByApp = EC\_TRUE and call rte\_eal\_init() and rte\_eal\_cleanup() itself.

To initialize DPDK following call is needed:

```
rte_eal_init();
```

To deinitialize DPDK following calls are needed:

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

## 5.10 DW3504 - emIIDW3504

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

```
struct EC_T_LINK_PARMS_DW3504
```

#### Public Members

##### *EC\_T\_LINK\_PARMS* **linkParms**

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_DW3504

##### *EC\_T\_DWORD* **dwPhyAddr**

PHY address

##### *EC\_T\_DWORD* **dwRegisterBasePhys**

Physical base address of register block (8k)

##### *EC\_T\_DW3504\_TYPE* **eDW3504Type**

System on Chip type

##### *EC\_T\_PHYINTERFACE* **ePhyInterface**

PHY connection type

##### *EC\_T\_BOOL* **bNotUseDmaBuffers**

EC\_FALSE: Use buffers from DMA for receive (default), EC\_TRUE: use buffers from heap for receive.  
EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC\_TRUE.

**EC\_T\_DWORD dwTxDmaDesCnt**

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

**EC\_T\_DWORD dwRxDmaDesCnt**

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

**EC\_T\_BOOL bNotUseCacheSync**

Default use of CacheSync EC\_FALSE, Don't call CacheSync on older systems EC\_TRUE

**EC\_T\_BOOL bUsePhyLib**

Use PhyLib instead of Legacy PHY handling for eDW3504\_CycloneV, eDW3504\_LCES1 or eDW3504\_RZN1, for all others the PhyLib is mandatory

**EC\_T\_BOOL bNoPhyAccess**

Don't use MDIO to set up the PHY

enum **EC\_T\_DW3504\_TYPE**

*Values:*

enumerator **eDW3504\_CycloneV**

MAC on Cyclone V SoC

enumerator **eDW3504\_LCES1**

MAC on LCES1 SoC

enumerator **eDW3504\_RZN1**

MAC on Renesas RZN1

enumerator **eDW3504\_STM32MP15x**

MAC on STM32MP15x

enumerator **eDW3504\_ATOM**

MAC on Atom 6000

enumerator **eDW3504\_STM32MP13x**

MAC on STM32MP13x

enumerator **eDW3504\_RK3328**

MAC on Rockchip 3328 - Rock64

enumerator **eDW3504\_RK3399**

MAC on Rockchip 3399 - Orange Pi 4

enumerator **eDW3504\_RK3588S**

MAC on Rockchip 3588s - Orange Pi 5

enumerator **eDW3504\_RK3568**

MAC on Rockchip 3568 - Radxa Rock3 a

enumerator **eDW3504\_SemidriveD9**

MAC on Semidrive D9

enumerator **eDW3504\_RK3576**

MAC on Rockchip 3576 - Tronlong TL-3576-EVM

enumerator **eDW3504\_RK3562**  
 MAC on Rockchip 3562 - Embedfire LubanCat3

enumerator **eDW3504\_AllWinnerT536**  
 MAC on AllWinner T536

enumerator **eDW3504\_RZN2**  
 MAC on Renesas RZN2

### 5.10.1 Supported PCI devices

Synopsis DW3504 PCI specific definitions (VendorId, DeviceId)

aggedright

- **PCI\_DEVICE\_INTEL\_EHL\_DWMAC\_RGMII\_1** (0x8086, 0x4B31)  
 (0x8086, 0x4BB0)
- **PCI\_DEVICE\_INTEL\_EHL\_DWMAC\_RGMII\_2** (0x8086, 0x4BA0)
- **PCI\_DEVICE\_INTEL\_EHL\_DWMAC\_RGMII\_3** (0x8086, 0x4B30)
- **PCI\_DEVICE\_INTEL\_EHL\_DWMAC\_SGMII\_1** (0x8086, 0x4B32)
- **PCI\_DEVICE\_INTEL\_EHL\_DWMAC\_SGMII\_2** (0x8086, 0x4BB1)
- **PCI\_DEVICE\_INTEL\_EHL\_DWMAC\_SGMII\_3** (0x8086, 0x4BA1)
- **PCI\_DEVICE\_INTEL\_EHL\_DWMAC\_SGMII\_4** (0x8086, 0x4B31)
- **PCI\_DEVICE\_INTEL\_EHL\_DWMAC\_SGMII\_5** (0x8086, 0x4BA2)
- **PCI\_DEVICE\_INTEL\_EHL\_DWMAC\_SGMII\_6** (0x8086, 0x4BB2)
- **PCI\_DEVICE\_INTEL\_TGL\_DWMAC\_SGMII\_1** (0x8086, 0x43A2)
- **PCI\_DEVICE\_INTEL\_TGL\_DWMAC\_SGMII\_2** (0x8086, 0x43AC)
- **PCI\_DEVICE\_INTEL\_TGL\_DWMAC\_SGMII\_3** (0x8086, 0xA0AC)
- **PCI\_DEVICE\_INTEL\_ADLS\_DWMAC\_SGMII\_1** (0x8086, 0x7AAC)
- **PCI\_DEVICE\_INTEL\_ADLS\_DWMAC\_SGMII\_2** (0x8086, 0x7AAD)

## 5.11 Freescale TSEC / eTSEC - emIETSEC

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

struct **EC\_T\_LINK\_PARMS\_ETSEC**

### Public Members

**EC\_T\_LINK\_PARMS linkParms**

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_ETSEC

**EC\_T\_DWORD dwRegisterBase**

Physical base address of register block (4k)

**EC\_T\_DWORD dwLocalMdioBase**

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

**EC\_T\_DWORD dwPhyMdioBase**

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

**EC\_T\_DWORD dwPhyAddr**

PHY address on MII bus. ETSEC\_FIXED\_LINK if fixed link configuration

**EC\_T\_DWORD dwTbiPhyAddr**

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

**EC\_T\_DWORD dwFixedLinkVal**

Only evaluated if dwPhyAddr == FIXED\_LINK. Set to one of the ETSEC\_LINKFLAG\_\* macros. I.e. ETSEC\_LINKFLAG\_1000baseT\_Full

**EC\_T\_BYTEx abyStationAddress[6]**

MAC address

**EC\_T\_VOID \*oMiiBusMtx**

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

**EC\_T\_DWORD dwRxInterrupt**

Receive interrupt number (IRQ)

**EC\_T\_BOOL bNotUseDmaBuffers**

EC\_FALSE: Use buffers from DMA for receive (default), EC\_TRUE: use buffers from heap for receive.  
EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC\_TRUE.

**EC\_T\_ETSEC\_TYPE eETSECType**

System on Chip type

**EC\_T\_BOOL bMaster**

Full control over the MAC and need to initialize MAC and the connections to the PHYs

enum **EC\_T\_ETSEC\_TYPE**

*Values:*

enumerator **eETSEC\_P2020RDB**

MAC on Freescale P2020

enumerator **eETSEC\_TWRP1025**

MAC on Freescale TWRP1025

enumerator **eETSEC\_ISTMPC8548**

MAC on Freescale ISTMPC8548

enumerator **eETSEC\_XJ\_EPU20C**

MAC on Freescale XJ EPU20C

enumerator **eETSEC\_TWRLS1021A**

MAC on Freescale TWRLS1021A

enumerator **eETSEC\_TQMLS\_LS102XA**

MAC on Freescale TQMLS LS102XA

### 5.11.1 ETSEC supported MAC's

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

### 5.11.2 Shared MII bus

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

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

---

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

---

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

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

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

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

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

### 5.11.3 Locking

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

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

### 5.11.4 Link check

The driver's API function EcLinkGetStatus (pfEcLinkGetStatus) is called by the EC-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 = emIoControl(( EC_IOCTL_LINKLAYER | EC_LINKIOCTL_UPDATE_LINKSTATUS), EC_
↪NULL);
```

EcLinkGetStatus always returns the last known link status.

### 5.11.5 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;
```

## 5.12 Freescale FslFec - emIIIFslFec

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

struct **EC\_T\_LINK\_PARMS\_FSLFEC**

#### Public Members

**EC\_T\_LINK\_PARMS linkParms**

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_FSLFEC

**EC\_T\_DWORD dwRxBuffers**

Receive buffer count

**EC\_T\_DWORD dwTxBuffers**

Transmit buffer count

**EC\_T\_FEC\_TYPE eFecType**

System on Chip type

**EC\_T\_PHYINTERFACE ePhyInterface**

PHY interface type

**EC\_T\_BOOL bUseDmaBuffers**

Use buffers from DMA (EC\_TRUE) or from heap for receive and AllocSend not supported (EC\_FALSE)

**EC\_T\_DWORD dwPhyAddr**

PHY Address

**EC\_T\_BOOL bNoPinMuxing**

No clock configuration and pin muxing

**EC\_T\_BOOL bDontReadMacAddr**

Read of MAC address disabled

**EC\_T\_DWORD dwRxInterrupt**

Receive interrupt number (IRQ)

**EC\_T\_BOOL bNoPhyAccess**

EC\_FALSE: Link layer should initialize PHY and read link status (connected/disconnected). EC\_TRUE: Client is responsible of PHY initialization and clock initialization

enum **EC\_T\_FEC\_TYPE**

*Values:*

enumerator **eFEC\_IMX25**

MAC on Freescale i.MX25 (ARM9; ARMv5)

enumerator **eFEC\_IMX28**

MAC on Freescale i.MX28 (ARM9; ARMv5)

enumerator **eFEC\_IMX53**

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

enumerator **eFEC\_IMX6**

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

enumerator **eFEC\_VF6**

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

enumerator **eFEC\_IMX7**

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

enumerator **eFEC\_IMX8**

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

enumerator **eFEC\_IMX8M**

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

enumerator **eFEC\_IMXRT1064**

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

enumerator **eFEC\_IMX9**

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

enumerator **eFEC\_IMX8MP**

MAC on NXP i.MX8MPLUS

## 5.13 Cadence GEM/MACB - emIGEM

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

```
struct EC_T_LINK_PARMS_GEM
```

### Public Members

***EC\_T\_LINK\_PARMS* `linkParms`**

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

***EC\_T\_GEM\_RXSOURCE* `eRxSource`**

Source of RX clock, control and data signals

***EC\_T\_DWORD* `dwPhyAddr`**

PHY address

***EC\_T\_DWORD* `dwRxInterrupt`**

Receive interrupt number (IRQ)

***EC\_T\_BOOL* `bUseDmaBuffers`**

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

***EC\_T\_BOOL* `bNoPhyAccess`**

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

***EC\_T\_BOOL* `bUseGmiiToRgmiiConv`**

Use XILINX GMIITORGMII Converter (`EC_TRUE`)

***EC\_T\_DWORD* `dwConvPhyAddr`**

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

***EC\_T\_DWORD* `dwTxDmaDesCnt`**

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

***EC\_T\_DWORD* `dwRxDmaDesCnt`**

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

***EC\_T\_GEM\_TYPE* `eGemType`**

System on Chip type

***EC\_T\_PHYINTERFACE* `ePhyInterface`**

PHY connection type

***EC\_T\_BOOL* `bNoPinMuxing`**

No clock configuration and pin muxing

***EC\_T\_GEM\_CLK\_DIV\_TYPE* `eClkDivType`**

Change Ref Clock settings

**EC\_T\_BOOL bNotUseCacheSync**

Default use of CacheSync EC\_FALSE, Don't call CacheSync on older systems EC\_TRUE

enum **EC\_T\_GEM\_RXSOURCE**

*Values:*

enumerator **eGemRxSource\_MIO**

MIO as source for RX clock, control and data signals

enumerator **eGemRxSource\_EMIO**

EMIO as source for RX clock, control and data signals

enum **EC\_T\_GEM\_TYPE**

*Values:*

enumerator **eGemType\_Zynq7000**

Xilinx Zynq 7000

enumerator **eGemType\_ZynqUltrascale**

Xilinx Zynq Ultrascale

enumerator **eGemType\_BCM2712**

Broadcom BCM2712, Raspberry Pi 5

## 5.14 INtime HPE - emllHPE

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

struct **EC\_T\_LINK\_PARMS\_HPE**

### Public Members

**EC\_T\_LINK\_PARMS linkParms**

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_HPE

EC\_T\_CHAR **szAdapterName[EC\_HPE\_ADAPTER\_NAME\_SIZE]**

Name of Ethernet Interface, e.g. "ie1g0"

EC\_T\_DWORD **dwRxBufferCnt**

Receive buffer count

EC\_T\_DWORD **dwTxBufferCnt**

Transmit buffer count

EC\_T\_BOOL **bDisableForceBroadcast**

Don't change target MAC address to FF:FF:FF:FF:FF:FF

## 5.15 Intel Pro/100 - emIII8255x

The parameters to the Real-time Ethernet Driver Intel Pro/100 are setup-specific. The function `CreateLinkParmsFromCmdLineI8255x()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

```
struct EC_T_LINK_PARMS_I8255X
```

### Public Members

#### `EC_T_LINK_PARMS linkParms`

Common link parameters. Signature must be set to `EC_LINK_PARMS_SIGNATURE_I8255X`

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

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

### 5.15.1 Supported PCI devices

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

aggedright

- **PCI\_DEVICE\_I82801DB** (0x8086, 0x103a)
- **PCI\_DEVICE\_I8255X** (0x8086, 0x1229)
- **PCI\_DEVICE\_I8255X\_ER** (0x8086, 0x1209)
- **PCI\_DEVICE\_I8255X\_VE** (0x8086, 0x1050)

- **PCI\_DEVICE\_I82562\_VM** (0x8086, 0x1039)
- **PCI\_DEVICE\_I82559\_ER** (0x8086, 0x2449)
- **PCI\_DEVICE\_I8255X\_VE2** (0x8086, 0x27DC)
- **PCI\_DEVICE\_I82551\_QM** (0x8086, 0x1059)
- **PCI\_DEVICE\_I8255X\_VE3** (0x8086, 0x1092)

## 5.16 Texas Instruments ICSS - emIIICSS

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

```
struct EC_T_LINK_PARMS_ICSS
```

**Public Members*****EC\_T\_LINK\_PARMS* linkParms**

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_ICSS

***EC\_T\_LINK\_ICSS\_BOARD* eBoardType**

TI System on Chip board type

**EC\_T\_BOOL bMaster**

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

**EC\_T\_BOOL bNoMacAddr**

EC\_TRUE: No MAC address registers access

**EC\_T\_DWORD dwPhyAddr**

PHY address

***EC\_T\_PHYINTERFACE* ePhyInterface**

PHY connection type

**EC\_T\_BOOL bNoPhyReset**

No hardware reset of the PHY

**EC\_T\_BOOL bUseAllSendQueues**

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

**EC\_T\_BOOL bLegacyFirmware**

For am57xx use legacy ICSS firmware from pdk\_am57xx\_1\_0\_6, instead of pdk\_am57xx\_1\_0\_17 with patch for Rx error issue, see [https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1022410/am5746-rx\\_error\\_offset-conditions/3788558#3788558](https://e2e.ti.com/support/processors-group/processors/f/processors-forum/1022410/am5746-rx_error_offset-conditions/3788558#3788558)

enum **EC\_T\_LINK\_ICSS\_BOARD**

Values:

enumerator **EcLinkIcssBoard\_Unsupported**

enumerator **EcLinkIcssBoard\_am572x**

TI AM572x

enumerator **EcLinkIcssBoard\_am571x**

TI AM571x

enumerator **EcLinkIcssBoard\_am3359**

TI AM3359

enumerator **EcLinkIcssBoard\_am572x\_emerson**

TI AM572x on Emerson board

enumerator **EcLinkIcssBoard\_am574x**

TI AM574x

### 5.16.1 TTS Feature

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

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

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

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

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

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

### 5.16.2 TI AM335x ICEV2

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

#### See also:

[http://processors.wiki.ti.com/index.php/AM335x\\_Industrial\\_Communication\\_Engine\\_EVM\\_Rev2\\_1\\_HW\\_User\\_Guide](http://processors.wiki.ti.com/index.php/AM335x_Industrial_Communication_Engine_EVM_Rev2_1_HW_User_Guide)

### 5.16.3 TI AM57xx IDK

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

### 5.16.4 AM5728 IDK and AM5718 IDK boards and Technical Limitations

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

---

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

---

## 5.17 Texas Instruments ICSSG - emIIICSSG on AM654x

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

```
struct EC_T_LINK_PARMS_ICSSG
```

### Public Members

`EC_T_LINK_PARMS linkParms`

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

`EC_T_LINK_ICSSG_BOARD eBoardType`

TI System on Chip board type

`EC_T_BOOL bMaster`

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

```
enum EC_T_LINK_ICSSG_BOARD
```

Values:

enumerator `EcLinkIcssgBoard_Unsupported`

enumerator `EcLinkIcssgBoard_am654x`

TI AM654x

### 5.17.1 TI AM654x IDK

Support for ICSSG on AM654x is currently limited to TI AM654x IDK/EVK with TI RTOS/Sysbios in polling mode.

## 5.18 Intel Pro/1000 - emIIINtelGbe

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

```
struct EC_T_LINK_PARMS_INTELGBE
```

## Public Members

### ***EC\_T\_LINK\_PARMS*** **linkParms**

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_INTELGBE

#### **EC\_T\_WORD** **wRxBufferCnt**

Receive buffer count, 0: default to 96

#### **EC\_T\_WORD** **wRxBufferSize**

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

#### **EC\_T\_WORD** **wTxBufferCnt**

Transmit buffer count, 0: default to 96

#### **EC\_T\_WORD** **wTxBufferSize**

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

#### **EC\_T\_BOOL** **bDisableLocks**

Disable locks

#### **EC\_T\_DWORD** **dwAutoNegTimeout**

Timeout [ms] for auto negotiation

#### **EC\_T\_BOOL** **bNotUseDmaBuffers**

EC\_FALSE: Use buffers from DMA for receive (default), EC\_TRUE: use buffers from heap for receive.  
EcLinkAllocSendFrame is not supported if bNotUseDmaBuffers = EC\_TRUE.

#### **EC\_T\_BOOL** **bNoPhyCtrlOnConnect**

EC\_TRUE: No PHY control (e.g. PHY reset, PHY PM settings, Gbits Ctrl) on link connection detected

NICs equipped with 82577, 82579 or 82567 may need HardCodedPhySettings. This must be set after [\*emInit-Master\(\)\*](#), before using the NIC, e.g.:

```
{
    EC_T_IOCTLPARMS oIoCtlParms;
    OsMemset(&oIoCtlParms, 0, sizeof(EC_T_IOCTLPARMS));
    oIoCtlParms.pbyInBuf      = (EC_T_BYTE*) EC_NULL + 0x20103;
    oIoCtlParms.dwInBufSize   = sizeof(EC_T_DWORD);
    emIoControl(dwInstanceId, EC_IOCTL_LINKLAYER_MAIN + EC_LINKIOCTL_FORCELINKMODE,
    ↪ &oIoCtlParms);
    OsSleep(1000);
}
```

## 5.18.1 TTS Feature

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

See also:

[\*EC\\_T\\_LINKLAYER\\_TIMINGTASK\*](#), [\*EC\\_T\\_LINKLAYER\\_TIMING\*](#)

## 5.18.2 Supported PCI devices

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

aggedright

- **PCI\_DEVICE\_I82540EM\_DESKTOP** (0x8086, 0x100E)
- **PCI\_DEVICE\_I82545EM\_COPPER** (0x8086, 0x100F)
- **PCI\_DEVICE\_I82546EB\_COPPER\_DUAL** (0x8086, 0x1010)
- **PCI\_DEVICE\_I82541EI\_COPPER** (0x8086, 0x1013)
- **PCI\_DEVICE\_I82547GI\_COPPER** (0x8086, 0x1019)
- **PCI\_DEVICE\_I82545GM\_COPPER** (0x8086, 0x1026)
- **PCI\_DEVICE\_I82566MM** (0x8086, 0x1049)
- **PCI\_DEVICE\_I82566DM** (0x8086, 0x104A)
- **PCI\_DEVICE\_I82566MC** (0x8086, 0x104D)
- **PCI\_DEVICE\_N1E5132\_SERVER** (0x8086, 0x105E)
- **PCI\_DEVICE\_I82547EI** (0x8086, 0x1075)
- **PCI\_DEVICE\_I82541GI\_COPPER** (0x8086, 0x1076)
- **PCI\_DEVICE\_I82541GI\_MOBILE** (0x8086, 0x1077)
- **PCI\_DEVICE\_I82541ER** (0x8086, 0x1078)
- **PCI\_DEVICE\_I82546GB\_COPPER\_DUAL** (0x8086, 0x1079)
- **PCI\_DEVICE\_I82541PI\_DESKTOP** (0x8086, 0x107C)
- **PCI\_DEVICE\_I82572EI** (0x8086, 0x107D)
- **PCI\_DEVICE\_I82573E** (0x8086, 0x108B)
- **PCI\_DEVICE\_I82573** (0x8086, 0x108C)
- **PCI\_DEVICE\_I82573L** (0x8086, 0x109A)
- **PCI\_DEVICE\_I82571GB\_QUAD** (0x8086, 0x10A4)
- **PCI\_DEVICE\_I82575\_ZOAR** (0x8086, 0x10A7)
- **PCI\_DEVICE\_I82572GI** (0x8086, 0x10B9)
- **PCI\_DEVICE\_I82571GB\_QUAD\_2** (0x8086, 0x10BC)
- **PCI\_DEVICE\_I82566L** (0x8086, 0x10BD)
- **PCI\_DEVICE\_I82576** (0x8086, 0x10C9)
- **PCI\_DEVICE\_I82567V** (0x8086, 0x10CE)
- **PCI\_DEVICE\_I82574L** (0x8086, 0x10D3)
- **PCI\_DEVICE\_I82567LM3** (0x8086, 0x10DE)
- **PCI\_DEVICE\_I82577LM** (0x8086, 0x10EA)
- **PCI\_DEVICE\_I82577LC** (0x8086, 0x10EB)
- **PCI\_DEVICE\_I82578DM** (0x8086, 0x10EF)
- **PCI\_DEVICE\_I82578DC** (0x8086, 0x10F0)
- **PCI\_DEVICE\_I82567LM** (0x8086, 0x10F5)
- **PCI\_DEVICE\_I82567V3** (0x8086, 0x1501)
- **PCI\_DEVICE\_I82579LM** (0x8086, 0x1502)
- **PCI\_DEVICE\_I82579V** (0x8086, 0x1503)
- **PCI\_DEVICE\_I82576NS** (0x8086, 0x150A)

- **PCI\_DEVICE\_I82583V** (0x8086, 0x150C)
- **PCI\_DEVICE\_I82580\_QUAD** (0x8086, 0x150E)
- **PCI\_DEVICE\_I350** (0x8086, 0x1521)
- **PCI\_DEVICE\_I82576\_ET2** (0x8086, 0x1526)
- **PCI\_DEVICE\_I82580\_QUAD\_FIBRE** (0x8086, 0x1527)
- **PCI\_DEVICE\_I210AT** (0x8086, 0x1531)
- **PCI\_DEVICE\_I210AT\_2** (0x8086, 0x1532)
- **PCI\_DEVICE\_I210\_COPPER** (0x8086, 0x1533)
- **PCI\_DEVICE\_I210IT** (0x8086, 0x1535)
- **PCI\_DEVICE\_I210\_SERDES** (0x8086, 0x1537)
- **PCI\_DEVICE\_I211AT** (0x8086, 0x1539)
- **PCI\_DEVICE\_I210\_COPPER\_FLASHLESS** (0x8086, 0x157B)
- **PCI\_DEVICE\_I210\_BACKPLANE** (0x8086, 0x157C)
- **PCI\_DEVICE\_I217LM** (0x8086, 0x153A)
- **PCI\_DEVICE\_I217V** (0x8086, 0x153B)
- **PCI\_DEVICE\_I218LM** (0x8086, 0x155A)
- **PCI\_DEVICE\_I218V** (0x8086, 0x1559)
- **PCI\_DEVICE\_I218LM\_2** (0x8086, 0x15A0)
- **PCI\_DEVICE\_I218V\_2** (0x8086, 0x15A1)
- **PCI\_DEVICE\_I218LM\_3** (0x8086, 0x15A2)
- **PCI\_DEVICE\_I218V\_3** (0x8086, 0x15A3)
- **PCI\_DEVICE\_I219LM** (0x8086, 0x156F)
- **PCI\_DEVICE\_I219LM\_2** (0x8086, 0x15B7)
- **PCI\_DEVICE\_I219LM\_3** (0x8086, 0x15B9)
- **PCI\_DEVICE\_I219LM\_4** (0x8086, 0x15D7)
- **PCI\_DEVICE\_I219LM\_5** (0x8086, 0x15E3)
- **PCI\_DEVICE\_I219LM\_6** (0x8086, 0x15BD)
- **PCI\_DEVICE\_I219LM\_7** (0x8086, 0x15BB)
- **PCI\_DEVICE\_I219LM\_8** (0x8086, 0x15DF)
- **PCI\_DEVICE\_I219LM\_9** (0x8086, 0x15E1)
- **PCI\_DEVICE\_I219V** (0x8086, 0x1570)
- **PCI\_DEVICE\_I219V\_2** (0x8086, 0x15B8)
- **PCI\_DEVICE\_I219V\_4** (0x8086, 0x15D8)
- **PCI\_DEVICE\_I219V\_5** (0x8086, 0x15D6)
- **PCI\_DEVICE\_I219V\_6** (0x8086, 0x15BE)
- **PCI\_DEVICE\_I219V\_7** (0x8086, 0x15BC)
- **PCI\_DEVICE\_I219V\_8** (0x8086, 0x15E0)
- **PCI\_DEVICE\_I219V\_9** (0x8086, 0x15E2)
- **PCI\_DEVICE\_I219LM\_10** (0x8086, 0x0D4E)
- **PCI\_DEVICE\_I219V\_10** (0x8086, 0x0D4F)
- **PCI\_DEVICE\_I219LM\_11** (0x8086, 0x0D4C)
- **PCI\_DEVICE\_I219V\_11** (0x8086, 0x0D4D)
- **PCI\_DEVICE\_I219LM\_12** (0x8086, 0x0D53)
- **PCI\_DEVICE\_I219V\_12** (0x8086, 0x0D55)
- **PCI\_DEVICE\_I219LM\_18** (0x8086, 0x0DC5)
- **PCI\_DEVICE\_I219V\_18** (0x8086, 0x0DC6)
- **PCI\_DEVICE\_I219LM\_19** (0x8086, 0x0DC7)
- **PCI\_DEVICE\_I219V\_19** (0x8086, 0x0DC8)

- PCI\_DEVICE\_I219LM\_13 (0x8086, 0x15FB)
- PCI\_DEVICE\_I219V\_13 (0x8086, 0x15FC)
- PCI\_DEVICE\_I219LM\_14 (0x8086, 0x15F9)
- PCI\_DEVICE\_I219V\_14 (0x8086, 0x15FA)
- PCI\_DEVICE\_I219LM\_15 (0x8086, 0x15F4)
- PCI\_DEVICE\_I219V\_15 (0x8086, 0x15F5)
- PCI\_DEVICE\_I219LM\_16 (0x8086, 0x1A1E)
- PCI\_DEVICE\_I219V\_16 (0x8086, 0x1A1F)
- PCI\_DEVICE\_I219LM\_17 (0x8086, 0x1A1C)
- PCI\_DEVICE\_I219V\_17 (0x8086, 0x1A1D)
- PCI\_DEVICE\_I219LM\_20 (0x8086, 0x550A)
- PCI\_DEVICE\_I219V\_20 (0x8086, 0x550B)
- PCI\_DEVICE\_I219LM\_21 (0x8086, 0x550C)
- PCI\_DEVICE\_I219V\_21 (0x8086, 0x550D)
- PCI\_DEVICE\_I219LM\_22 (0x8086, 0x550E)
- PCI\_DEVICE\_I219V\_22 (0x8086, 0x550F)
- PCI\_DEVICE\_I219LM\_23 (0x8086, 0x5510)
- PCI\_DEVICE\_I219V\_23 (0x8086, 0x5511)
- PCI\_DEVICE\_I219LM\_24 (0x8086, 0x55A0)
- PCI\_DEVICE\_I219V\_24 (0x8086, 0x55A1)
- PCI\_DEVICE\_I225LM (0x8086, 0x15F2)
- PCI\_DEVICE\_I225V (0x8086, 0x15F3)
- PCI\_DEVICE\_I225I (0x8086, 0x15F8)
- PCI\_DEVICE\_I225K (0x8086, 0x3100)
- PCI\_DEVICE\_I225K\_2 (0x8086, 0x3101)
- PCI\_DEVICE\_I225LMVP (0x8086, 0x5502)
- PCI\_DEVICE\_I225IT (0x8086, 0xD9F)
- PCI\_DEVICE\_I226LM (0x8086, 0x125B)
- PCI\_DEVICE\_I226V (0x8086, 0x125C)
- PCI\_DEVICE\_I226IT (0x8086, 0x125D)

## 5.19 Microchip LAN743x - emllan743x

The parameters to the Real-time Ethernet Driver LAN743x are setup-specific. The function `CreateLinkParmsFromCmdLineLAN743x()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

```
struct EC_T_LINK_PARMS_LAN743X
```

### Public Members

#### `EC_T_LINK_PARMS linkParms`

Common link parameters. Signature must be set to `EC_LINK_PARMS_SIGNATURE_LAN743X`

#### `EC_T_BOOL bNotUseDmaBuffers`

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

#### `EC_T_DWORD dwRxBuffers`

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

#### `EC_T_DWORD dwTxBuffers`

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

### 5.19.1 Supported PCI devices

Microchip LAN743x PCI specific definitions (VendorId, DeviceId)

aggedright

- PCI\_DEVICE\_LAN743X (0x1055, 0x7430)

## 5.20 Beckhoff CUxxxx Multiplier - emlIMultiplier

The parameters to the Multiplier Real-time Ethernet Driver are setup-specific. The function “CreateLinkParmsFromCmdLineMultiplier” in EcSelectLinkLayer.cpp demonstrates how to initialize the Real-time Ethernet Driver instance.

```
struct EC_T_LINK_PARMS_MULTIPLIER
```

### Public Members

***EC\_T\_LINK\_PARMS linkParms***

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_MULTIPLIER

***EC\_T\_MULTIPLIER\_TYPE eMultiplierType***

Type of the Multiplier Ethernet port

***EC\_T\_DWORD dwPort***

Used CU2508 downlink port

***EC\_T\_LINK\_PARMS \*pHwLinkParms***

Link parameters of network adapter connected to the uplink of the Multiplier

enum ***EC\_T\_MULTIPLIER\_TYPE***

*Values:*

enumerator ***eMultiplier\_Cu2508***

Beckhoff CU2508 port multiplier

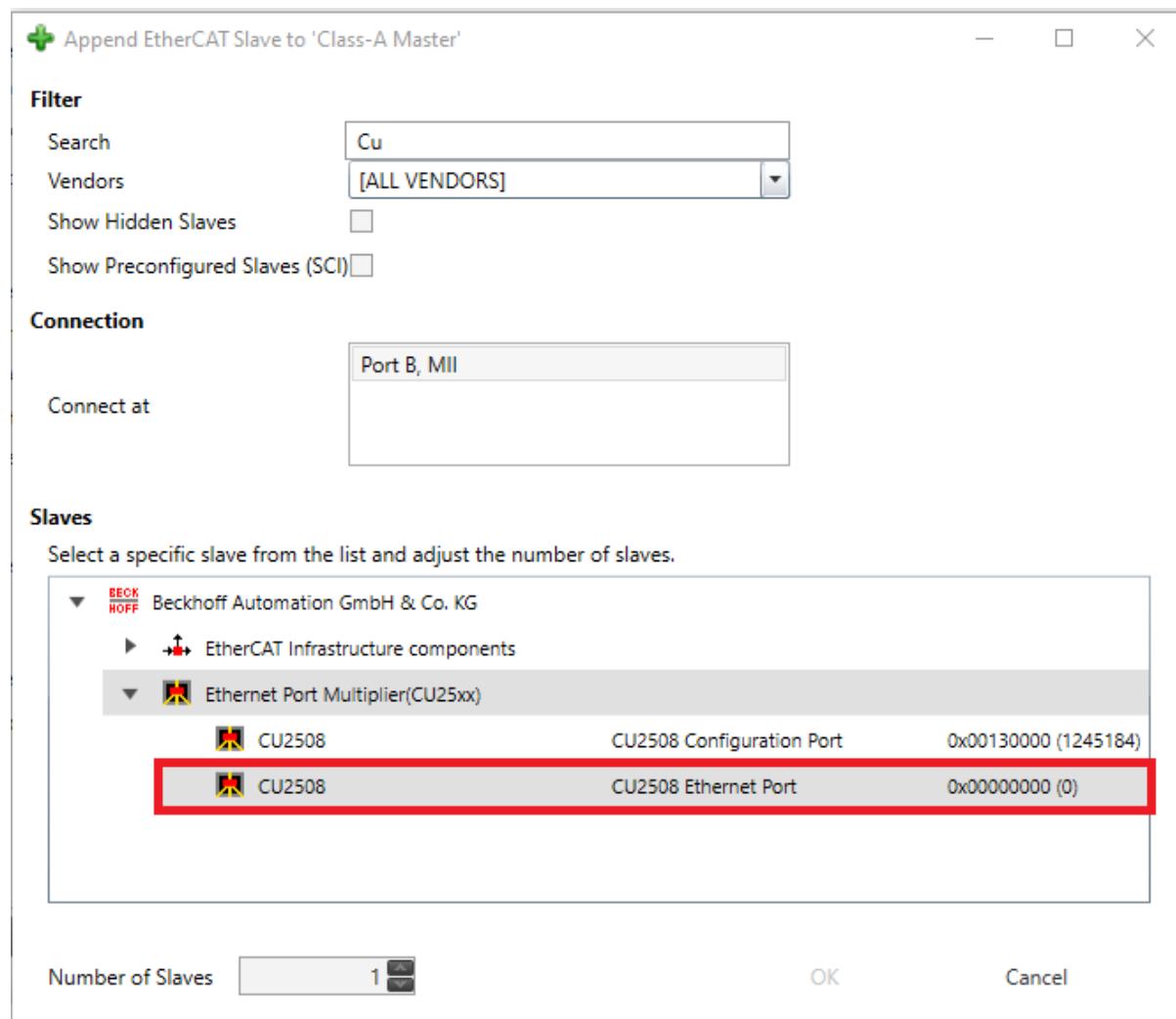
enumerator ***eMultiplier\_Et2000***

Beckhoff ET2000 industrial Ethernet multi-channel probe

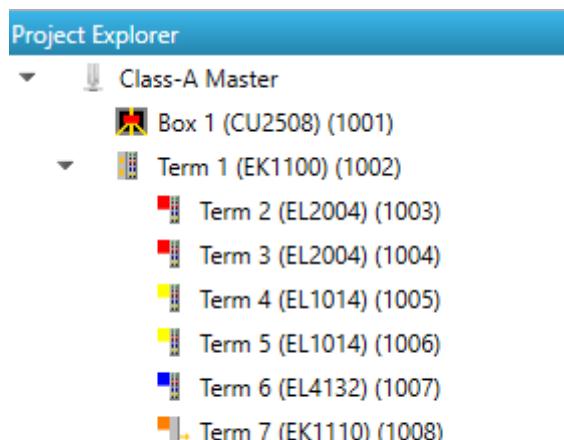
### 5.20.1 Configuration with EC-Engineer

This configuration is valid for one downlink port. For each used CUxxxx multiplier downlink port, a new configuration should be arranged.

- Start EC-Engineer in Offline Configuration modus.
- Add the CU2508 Ethernet Port as your first slave.



- Append the slaves that you are going to connect to the port.



## 5.21 Windows NDIS - emlNdis

As default EC-Master for Windows contains emlNdis.dll to use a native Windows driver for EtherCAT®.

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

- Bin/Windows/x64/EcatNdisSetup-x86\_64Bit.msi or
- Bin/Windows/x86/EcatNdisSetup-x86\_32Bit.msi

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

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

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

```
struct EC_T_LINK_PARMS_NDIS
```

### Public Members

#### **EC\_T\_LINK\_PARMS linkParms**

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_NDIS

#### **EC\_T\_CHAR szAdapterName[EC\_NDIS\_ADAPTER\_NAME\_SIZE]**

ServiceName of network adapter, see HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkCards in registry (zero terminated)

#### **EC\_T\_BYTE abyIpAddress[4]**

IP address of network adapter

#### **EC\_T\_BOOL bDisablePromiscuousMode**

Disable adapter promiscuous mode

#### **EC\_T\_BOOL bDisableForceBroadcast**

Don't change target MAC address to FF:FF:FF:FF:FF:FF

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

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

## 5.22 Windows WinPcap - emlIPcap

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

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

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

```
struct EC_T_LINK_PARMS_WINPCAP
```

### Public Members

#### `EC_T_LINK_PARMS linkParms`

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

#### `EC_T_BYTE abyIpAddress[4]`

IP address

#### `EC_T_CHAR szAdapterId[MAX_LEN_WINPCAP_ADAPTER_ID]`

Adapter ID, format: {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}

#### `EC_T_BOOL bFilterInput`

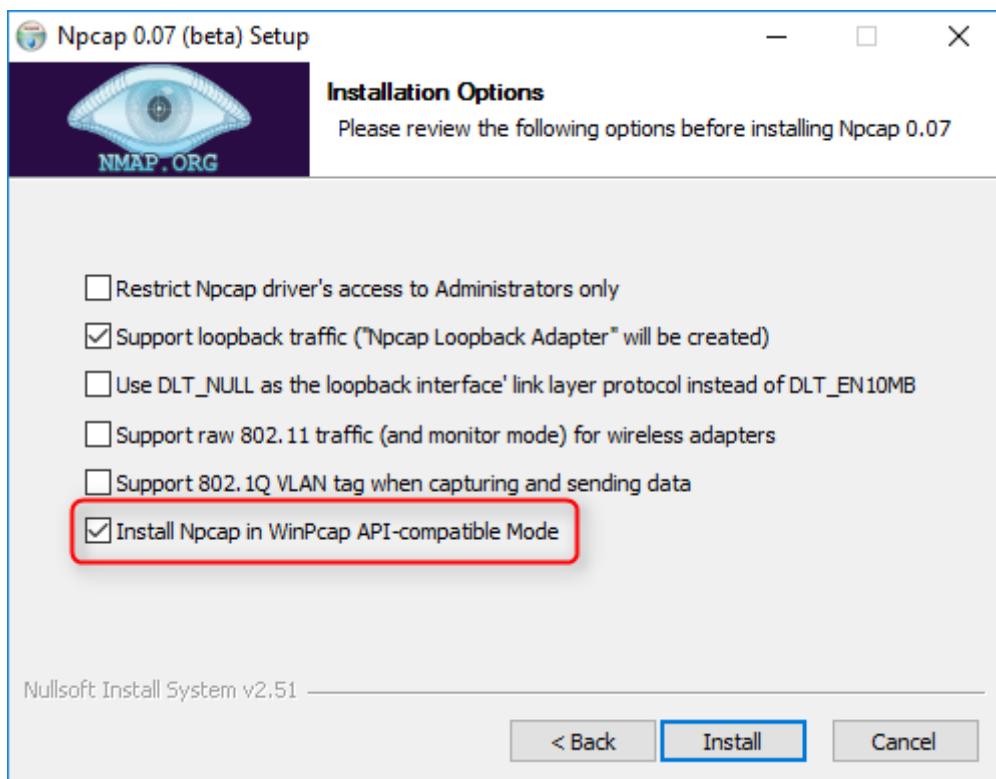
Filter input if `EC_TRUE`. This is needed on some system if the winpcap library notify the sent frames to the network adapter

### 5.22.1 WinPcap, Npcap support

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

The EC-Master installer installs WinPcap by default.

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



## 5.23 RDC R6040 - emIIR6040

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

```
struct EC_T_LINK_PARMS_R6040
```

### Public Members

#### `EC_T_LINK_PARMS linkParms`

Common link parameters. Signature must be set to `EC_LINK_PARMS_SIGNATURE_R6040`

#### `EC_T_DWORD dwTxDmaDesCnt`

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

#### `EC_T_DWORD dwRxDmaDesCnt`

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

### 5.23.1 Supported PCI devices

RDC R6040 PCI specific definitions (VendorId, DeviceId)

aggedright

- **PCI\_DEVICE\_R6040** (0x17F3, 0x6040)

## 5.24 emllRemote

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

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

```
struct EC_T_LINK_PARMS_REMOTE
```

### Public Members

***EC\_T\_LINK\_PARMS linkParms***

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

***EC\_T\_DWORD dwSocketType***

Socket type. Must be set to 1 (`emrassocktype_tcp`)

***EC\_T\_BYTEx abySrcIpAddress[4]***

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

***EC\_T\_WORD wSrcPort***

Source port number (listen) (EC-Simulator)

***EC\_T\_BYTEx abyDstIpAddress[4]***

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

***EC\_T\_WORD wDstPort***

Destination port number (connect) (EC-Master)

***EC\_T\_BYTEx abyMac[6]***

MAC address

***EC\_T\_DWORD dwRxBufferCnt***

Frame buffer count for interrupt service thread (IST)

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

EcSimulatorHilDemo command line example:

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

EcMasterDemo command line example:

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

## 5.25 Realtek RTL8169 - emIIRTL8169

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

```
struct EC_T_LINK_PARMS_RTL8169
```

### Public Members

**`EC_T_LINK_PARMS linkParms`**

Common link parameters. Signature must be set to `EC_LINK_PARMS_SIGNATURE_RTL8169`

**`EC_T_BOOL bNotUseDmaBuffers`**

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

**`EC_T_BOOL bAckErrInIrq`**

Acknowledge errors in interrupt handler

**`EC_T_DWORD dwRxBuffers`**

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

**`EC_T_DWORD dwTxBuffers`**

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

**`EC_T_BOOL bNoPhyAccess`**

Don't use MDIO to set up the PHY

### 5.25.1 RTL8169 usage under Linux

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

### 5.25.2 Supported PCI devices

RealTek RTL8169 PCI specific definitions (VendorId, DeviceId)

aggedright

- **`PCI_DEVICE_RTL8169`** (0x10EC, 0x8169)
- **`PCI_DEVICE_RTL8168`** (0x10EC/0x19EC, 0x8168)
- **`PCI_DEVICE_RTL8169_SC`** (0x10EC, 0x8167)
- **`PCI_DEVICE_DLINK_RTL8169`** (0x1186,

0x4300)

- **`PCI_DEVICE_RTL8103`** (0x10EC, 0x8136)
- **`PCI_DEVICE_KILLER_E2600`** (0x10EC, 0x2600)
- **`PCI_DEVICE_RTL8125`** (0x10EC, 0x8125)
- **`PCI_DEVICE_RTL8126`** (0x10EC, 0x8126)
- **`PCI_DEVICE_RTL8161`** (0x10EC, 0x8161)

## 5.26 Renesas RZ/T1 - emlIRZT1

The parameters to the Real-time Ethernet Driver Renesas RZ/T1 are setup-specific. The function `CreateLinkParmsFromCmdLineRZT1()` in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

```
struct EC_T_LINK_PARMS_RZT1
```

### Public Members

**`EC_T_LINK_PARMS linkParms`**

Common link parameters. Signature must be set to `EC_LINK_PARMS_SIGNATURE_RZT1`

## 5.27 Renesas SHEth - emlISHEth

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

```
struct EC_T_LINK_PARMS_SHETH
```

### Public Members

**`EC_T_LINK_PARMS linkParms`**

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

**`EC_T_SHETH_TYPE eType`**

System on Chip type

**`EC_T_BYTE abyStationAddress[6]`**

MAC address

**`EC_T_DWORD dwBaseAddr`**

Physical address of register block

**`EC_T_BYTEx byPhyAddr`**

PHY address

**`EC_T_BOOL bNotUseDmaBuffers`**

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

**`EC_T_DWORD dwTxDmaDesCnt`**

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

**`EC_T_DWORD dwRxDmaDesCnt`**

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

**enum `EC_T_SHETH_TYPE`**

*Values:*

enumerator **eSHEth\_R8A777X**  
Renesas R8A777X

enumerator **eSHEth\_R8A779X**  
Renesas R8A779X

enumerator **eSHEth\_SH7724**  
Renesas SH7724

enumerator **eSHEth\_SH7757**  
Renesas SH7757

enumerator **eSHEth\_SH7757\_GIGA**  
Renesas SH7757\_GIGA

enumerator **eSHEth\_SH7734**  
Renesas SH7734

enumerator **eSHEth\_SH7763**  
Renesas SH7763

enumerator **eSHEth\_R8A7740**  
Renesas R8A7740

enumerator **eSHEth\_R7S72100**  
Renesas R7S72100

enumerator **eSHEth\_SH7619**  
Renesas SH7619

enumerator **eSHEth\_SH771X**  
Renesas SH771X

enumerator **eSHEth\_R8A77400**  
Renesas R8A77400

enumerator **eSHEth\_R8A77435**  
Renesas R8A77435

enumerator **eSHEth\_R8A77430**  
Renesas R8A77430

enumerator **eSHEth\_R8A77450**  
Renesas R8A77450

enumerator **eSHEth\_RX72N**  
Renesas RX72N

enumerator **eSHEth\_RZG2L**  
Renesas RZG2L

### 5.27.1 SHEth link status update

On some targets like *Armadillo A800 eva* the link status can't be obtained directly by reading MAC PHY status register without access to the MII bus. Accessing the bus would violate timing constraints and is therefore not possible.

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

```
dwRes = emIoControl((EC_IOCTL_LINKLAYER | EC_LINKIOCTL_UPDATE_LINKSTATUS), EC_
    ↴NULL);
```

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

### 5.27.2 SHEth usage under Linux

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

## 5.28 VxWorks SNARF - emlISNARF

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

The network adapter driver has to be loaded prior to initialize the EC-Master stack.

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

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

```
struct EC_T_LINK_PARMS_SNARF
```

#### Public Members

##### `EC_T_LINK_PARMS linkParms`

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

##### `EC_T_CHAR szAdapterName[EC_SNARF_ADAPTER_NAME_SIZE]`

SNARF adapter name (zero terminated)

##### `EC_T_DWORD dwRxBuffers`

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

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

OsMemset(&oLinkParmsAdapter, 0, sizeof(EC_T_LINK_PARMS_SNARF));
```

(continues on next page)

(continued from previous page)

```

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

```

## 5.29 Linux SockRaw - emllSockRaw

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

---

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

---

**To run the application without root privileges, set the Linux capability ‘cap\_net\_raw’ to the application.**

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

**To run python scripts without root privileges, create a python environment and set the Linux capability ‘cap\_net\_raw’ to the python interpreter.**

```

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

```

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

```
struct EC_T_LINK_PARMS_SOCKRAW
```

### Public Members

#### **EC\_T\_LINK\_PARMS linkParms**

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_SOCKRAW

#### **EC\_T\_CHAR szAdapterName[EC\_SOCKRAW\_ADAPTER\_NAME\_SIZE]**

Native ETH device name, e.g. “eth0” (zero terminated)

#### **EC\_T\_BOOL bDisableForceBroadcast**

Don't change target MAC address to FF:FF:FF:FF:FF:FF

#### **EC\_T\_BOOL bReplacePaddingWithNopCmd**

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

#### **EC\_T\_BOOL bUsePacketMmapRx**

Use PACKET\_MMAP PACKET\_RX\_RING for receive

#### **EC\_T\_BOOL bSetCoalescingParms**

Set Coalescing parameters to enhance the link layer performance

**EC\_T\_BOOL bSetPromiscuousMode**  
Enable promiscuous mode at network adapter

## 5.30 Linux SockXdp - emlISockXdp

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

---

**Note:** Root privileges are required.

---

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

struct **EC\_T\_LINK\_PARMS\_SOCKXDP**

### Public Members

**EC\_T\_LINK\_PARMS linkParms**

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

**EC\_T\_CHAR szAdapterName[EC\_SOCKXDP\_ADAPTER\_NAME\_SIZE]**

Native ETH device name, e.g. "eth0" (zero terminated)

**EC\_T\_WORD wQueueId**

Used NIC Queue Id

**EC\_T\_XDP\_MODE eXdpMode**

XDP mode

**EC\_T\_BOOL bDisableForceBroadcast**

Don't change target MAC address to FF:FF:FF:FF:FF:FF

**EC\_T\_BOOL bReplacePaddingWithNopCmd**

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

enum **EC\_T\_XDP\_MODE**

Values:

enumerator **eXDP\_SKB\_MODE**

Generic XDP, XDP runs in the standard network stack

enumerator **eXDP\_DRV\_MODE**

Native XDP, Runs XDP directly in the NIC driver

enumerator **eXDP\_HW\_MODE**

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

enumerator **eXDP\_DRV\_ZEROCOPY**  
Native XDP with ZEROCOPY

### 5.30.1 Linux System Requirements

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

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

`CONFIG_XDP_SOCKETS=y`

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

### 5.30.2 Getting started

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

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

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

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

## 5.31 Texas Instruments CPSWG for AM6x and Jacinto 7 based on Enet LLD - emIITiEnetCpswg

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

```
struct EC_T_LINK_PARMS TIENETCPSWG
```

#### Public Members

`EC_T_LINK_PARMS linkParms`

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

`EC_T_BOOL bMaster`

EC\_TRUE: Initialize MAC

### 5.31.1 TI J784S4X EVM

Support for TiEnetCpswg on J784s4 is currently limited to TI J784S4X EVM with FreeRTOS in polling mode. It's working with ti-processor-sdk-rtos-j784s4-evm-08\_06\_01\_03.

## 5.32 Texas Instruments ICSSG for AM6x and Jacinto 7 based on Enet LLD - emlITiEnetIcssg

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

```
struct EC_T_LINK_PARMS_TIENETICSSG
```

### Public Members

`EC_T_LINK_PARMS linkParms`

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

`EC_T_BOOL bMaster`

`EC_TRUE`: Initialize MAC

`EC_T_BYTEx byPhyAddr`

PHY address

### 5.32.1 TI AM64x EVM

Support for ICSSG/TiEnetIcssg on AM64x is currently limited to TI AM64x EVM with FreeRTOS in polling mode. Working with MCU Plus SDK AM64x 08.01.00.36.

### 5.32.2 TI AM243x LP and TI AM243x EVM

Support for ICSSG/TiEnetIcssg on AM243x is currently limited to TI AM243x LP and TI AM243x EVM with FreeRTOS in polling mode. Working with MCU Plus SDK AM64x 08.01.00.36.

## 5.33 Virtual Local Area Network - emlIVlan

The parameters to the VLAN Real-time Ethernet Driver are setup-specific. The function “`CreateLinkParmsFromCmdLineVlan`” in `EcSelectLinkLayer.cpp` demonstrates how to initialize the Real-time Ethernet Driver instance.

```
struct EC_T_LINK_PARMS_VLAN
```

## Public Members

### ***EC\_T\_LINK\_PARMS linkParms***

Common link parameters. Signature must be set to EC\_LINK\_PARMS\_SIGNATURE\_MULTIPLIER

#### **EC\_T\_WORD wVlanId**

VLAN Identifier (VID)

#### **EC\_T\_WORD wVlanPrio**

VLAN Priority code point (PCP)

### ***EC\_T\_LINK\_PARMS \*pHwLinkParms***

Link parameters of network adapter connected to the uplink of the Multiplier

#### **EC\_T\_VLAN\_MODE eVlanMode**

Vlan switch operation mode

## 6 Application programming interface, reference

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

### 6.1 Generic API return status values

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

#### ***EC\_E\_NOERROR***

The function was successfully executed

#### ***EC\_E\_NOTSUPPORTED***

Unsupported feature or functionality

#### ***EC\_E\_BUSY***

The 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:

- *emNotify - EC\_NOTIFY\_STATUS\_SLAVE\_ERROR*
- *emNotify - 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 remotely 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.

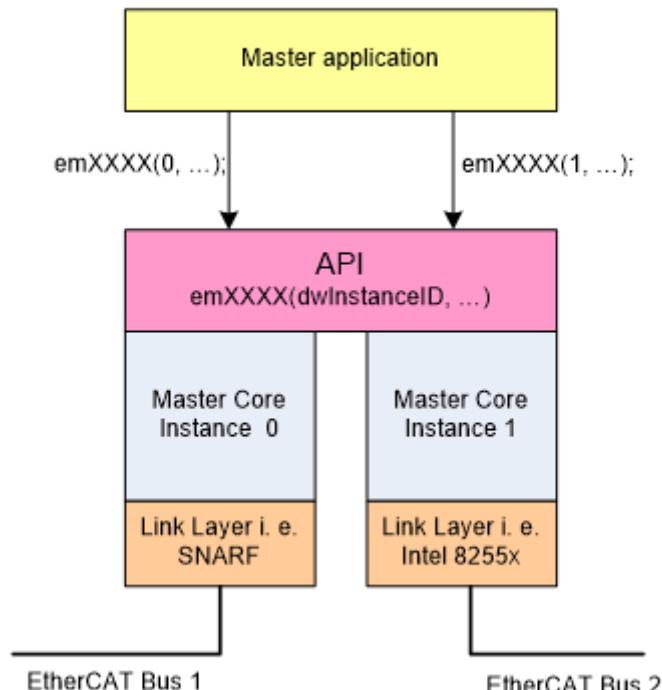
## 6.2 Multiple EtherCAT Bus Support

### 6.2.1 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.

### 6.2.2 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 12 (`MAX_NUMOF_MASTER_INSTANCES`).



### 6.2.3 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:

```
$ -ndis 192.168.1.32 1 -f el9800.xml @ -ndis 192.168.2.32 1 -f el9800.xml
```

## 6.3 General functions

### 6.3.1 emInitMaster

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

```
EC_T_DWORD emInitMaster (
    EC_T_DWORD dwInstanceID,
    EC_T_INIT_MASTER_PARMS *pParms
)
```

Initialize EC-Master.

This function has to be called prior to calling any other function of EC-Master.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pParms** – [in] Pointer to parameter definitions

#### Returns

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master is already initialized
- **EC\_E\_INVALIDPARM** if dwInstanceID is out of range or pParms is EC\_NULL or contains some values out of range
- **EC\_E\_SIGNATURE\_MISMATCH** if **EC\_T\_INIT\_MASTER\_PARMS.dwSignature** mismatch
- **EC\_E\_NOTFOUND** if the link layer cannot be found
- **EC\_E\_FEATURE\_DISABLED** if a configured feature is not included in the license key
- **EC\_E\_NOTSUPPORTED** if a configured feature is not supported (e.g not compiled in the library)
- **EC\_E\_LOCK\_CREATE\_FAILED** if some lock (e.g mutex) cannot be created
- **EC\_E\_NOMEMORY** if some memory cannot be allocated

```
struct EC_T_INIT_MASTER_PARMS
    EC-Master init parameters.
```

#### Public Members

**EC\_T\_DWORD dwSignature**  
 [in] Set to ATECAT\_SIGNATURE

**EC\_T\_DWORD dwSize**  
 [in] Set to sizeof(EC\_T\_INIT\_MASTER\_PARMS)

**EC\_T\_LOG\_PARMS LogParms**  
 [in] Logging parameters

**EC\_T\_OS\_PARMS \*pOsParms**  
 [in] OS layer parameters

***EC\_T\_LINK\_PARMS \*pLinkParms***

[in] Link layer parameters

***EC\_T\_LINK\_PARMS \*pLinkParmsRed***

[in] Link layer parameters for red device (cable redundancy)

***EC\_T\_DWORD dwBusCycleTimeUsec***

[in] Bus cycle time [usec] (set to zero when using dwBusCycleTimeNsec)

***EC\_T\_DWORD dwMaxBussSlaves***

[in] Maximum pre-allocated bus slave objects ( default: 256 )

***EC\_T\_DWORD dwMaxAcycFramesQueued***

[in] Maximum queued acyclic frames

***EC\_T\_DWORD dwAdditionalEoEEndpoints***

[in] Additional EoE endpoints

***EC\_T\_DWORD dwMaxAcycBytesPerCycle***

[in] Maximum bytes sent during eUsrJob\_SendAcycFrames per cycle ( default: 4096 )

***EC\_T\_DWORD dwMaxAcycFramesPerCycle***

[in] Maximum frames sent during eUsrJob\_SendAcycFrames per cycle ( default: 32 )

***EC\_T\_DWORD dwMaxAcycCmdsPerCycle***

[in] Maximum commands sent during eUsrJob\_SendAcycFrames per cycle ( default: 124 )

***EC\_T\_DWORD dwMaxSlavesProcessedPerCycle***

[in] Maximum slave-related state machine calls per cycle (default = all)

***EC\_T\_DWORD dwEcatCmdMaxRetries***

[in] Maximum retries to send pending EtherCAT command frames ( default: 3 )

***EC\_T\_DWORD dwEcatCmdTimeout***

[in] Timeout [ms] to send pending EtherCAT command frames ( default: 3 \* bus cycle time, at least 2 ms )

***EC\_T\_BOOL bReserved***

[in] bVLANEnable: obsolete

***EC\_T\_WORD wReserved***

[in] wVLANId: obsolete

***EC\_T\_BYTExbyReserved***

[in] byVLANPrio: obsolete

***EC\_T\_MASTER\_RED\_PARMS MasterRedParms***

[in] Master Redundancy parameters

***EC\_T\_DWORD dwMaxS2SMbxSize***

[in] Size of the queued S2S mailbox in bytes ( default: 0 )

***EC\_T\_DWORD dwMaxQueuedS2SMbxTfer***

[in] S2S Fifo number of entries ( default: 0 )

**EC\_T\_WORD wMaxSlavesProcessedPerBusScanStep**

[in] Maximum slave-related calls per cycle during bus scans (default = all)

**EC\_T\_BOOL bApiLockByApp**

[in] Lock pending API against *emDeinitMaster()*. EC\_FALSE (default): locked internally. EC\_TRUE: application is responsible for locking.

***EC\_T\_PERF\_MEAS\_INTERNAL\_PARMS* PerfMeasInternalParms**

[in] Internal performance measurement parameters

**EC\_T\_DWORD dwBusCycleTimeNsec**

[in] Bus cycle time [nsec] (set to zero when using dwBusCycleTimeUsec)

**EC\_T\_BOOL bNoConsecutiveAcycFrames**

[in] EC\_FALSE(default) : no restriction. EC\_TRUE : Don't process and send acyclic frames in the same cycle to reduce CPU load

struct **EC\_T\_OS\_PARMS**

**Public Members****EC\_T\_DWORD dwSignature**

[in] Set to EC\_OS\_PARMS\_SIGNATURE

**EC\_T\_DWORD dwSize**

[in] Set to sizeof(EC\_T\_OS\_PARMS)

***EC\_T\_LOG\_PARMS* \*pLogParms**

[in] Pointer to logging parameters

**EC\_PF\_SYSTIME pfSystemTimeGet**

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

**EC\_T\_DWORD dwSupportedFeatures**

[in/out] reserved

**EC\_PF\_QUERY\_MSEC\_COUNT pfSystemQueryMsecCount**

[in] Function to get system's msec count

struct **EC\_T\_LOG\_PARMS**

**Public Members****EC\_T\_DWORD dwLogLevel**

[in] Log level. See *EC\_LOG\_LEVEL\_...*

***EC\_PF\_LOGMSGHK* pfLogMsg**

[in] Log callback function called on every message

**EC\_T\_LOG\_CONTEXT \*pLogContext**

[in] Log context to be passed to log callback

EC\_LOG\_LEVEL... The following log levels are defined:

```
EC_LOG_LEVEL_SILENT
EC_LOG_LEVEL_ANY
EC_LOG_LEVEL_CRITICAL
EC_LOG_LEVEL_ERROR
EC_LOG_LEVEL_WARNING
EC_LOG_LEVEL_INFO
EC_LOG_LEVEL_INFO_API
EC_LOG_LEVEL_VERBOSE
EC_LOG_LEVEL_VERBOSE_ACYC
EC_LOG_LEVEL_VERBOSE_CYC
EC_LOG_LEVEL_UNDEFINED
```

typedef EC\_T\_DWORD (\***EC\_PF\_LOGMSGHK**)(EC\_T\_LOG\_CONTEXT \*pContext, EC\_T\_DWORD dwLogMsgSeverity, const EC\_T\_CHAR \*szFormat, ...)

#### Parameters

- **pContext** – [in] Context pointer. This pointer is used as parameter when the callback function is called
- **dwLogMsgSeverity** – [in] Log message severity, EC\_LOG\_LEVEL\_...
- **szFormat** – [in] String that contains the text to be written. It can optionally contain embedded format specifiers that are replaced by the values specified in subsequent additional arguments and formatted as requested.

#### Returns

EC\_E\_NOERROR or error code

Log messages are passed from the EC-Master to the callback given at *EC\_T\_LOG\_PARMS::pfLogMsg*. EcLogging.cpp demonstrates how messages can be handled by the application. For performance reasons the EC-Master automatically filters log messages according to *EC\_T\_LOG\_PARMS::dwLogLevel*. E.g. messages of severity *EC\_LOG\_LEVEL\_WARNING* are not passed to the application if *EC\_T\_LOG\_PARMS::dwLogLevel* is set to *EC\_LOG\_LEVEL\_ERROR*.

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

struct **EC\_T\_PERF\_MEAS\_INTERNAL\_PARMS**

#### Public Members

##### **EC\_T\_BOOL bEnabled**

[in] enable/disable internal performance counters.

##### **EC\_T\_PERF\_MEAS\_COUNTER\_PARMS CounterParms**

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

##### **EC\_T\_PERF\_MEAS\_HISTOGRAM\_PARMS HistogramParms**

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

---

```
struct EC_T_PERF_MEAS_COUNTER_PARMS
```

### Public Members

**EC\_PF\_PERF\_MEAS\_GETCOUNTERTICKS** **pfGetCounterTicks**

[in] Function returning the current counter ticks

**EC\_T\_VOID** \***pvGetCounterTicksContext**

[in] Context passed into GetCounterTicks

**EC\_T\_UINT64** **qwFrequency**

[in] Frequency in Hz used by the timer in GetCounterTicks

typedef EC\_T\_UINT64 (\***EC\_PF\_PERF\_MEAS\_GETCOUNTERTICKS**)(EC\_T\_VOID \*pvContext)

#### Parameters

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

```
struct EC_T_PERF_MEAS_HISTOGRAM_PARMS
```

### Public Members

**EC\_T\_DWORD** **dwBinCount**

[in] amount of bins to use for the histogram.

**EC\_T\_UINT64** **qwMinTicks**

[in] results below qwMinTicks are stored in the first bin

**EC\_T\_UINT64** **qwMaxTicks**

[in] results above qwMaxTicks are stored in the last bin

## 6.3.2 emDeinitMaster

static EC\_T\_DWORD **ecatDeinitMaster** (EC\_T\_VOID)

**EC\_T\_DWORD** **emDeinitMaster** (EC\_T\_DWORD dwInstanceID)

Deinitialize EC-Master.

Waits for pending API calls if *emInitMaster()* was called with *EC\_T\_INIT\_MASTER\_PARMS::bApiLockByApp* = EC\_FALSE (default).

#### Returns

*EC\_E\_NOERROR* or error code

### 6.3.3 emGetMasterParms

```
static EC_T_DWORD ecatGetMasterParms (
    EC_T_INIT_MASTER_PARMS *pParms,
    EC_T_DWORD dwParmsBufSize
)
EC_T_DWORD emGetMasterParms (
    EC_T_DWORD dwInstanceID,
    EC_T_INIT_MASTER_PARMS *pParms,
    EC_T_DWORD dwParmsBufSize
)
    Get current Master initialization parameters.
```

If the given buffer is larger than the actual size of structure *EC\_T\_INIT\_MASTER\_PARMS*, the parameters of *EC\_T\_INIT\_MASTER\_PARMS.pOsParms*, *EC\_T\_INIT\_MASTER\_PARMS.pLinkParms* and *EC\_T\_INIT\_MASTER\_PARMS.pLinkParmsRed* are appended.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pParms** – [out] Buffer to store Master parameters
- **dwParmsBufSize** – [in] Size of Master parameters buffer

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or the output pointer is EC\_NULL or dwParmsBufSize is too small

### emGetMasterParms() Example

```
EC_T_BYTE abyBuffer[sizeof(EC_T_INIT_MASTER_PARMS)
    + sizeof(EC_T_OS_PARMS) + 512 /* LinkLayer parameters */];
EC_T_INIT_MASTER_PARMS* pParms = (EC_T_INIT_MASTER_PARMS*)abyBuffer;
dwRes = emGetMasterParms(dwInstanceId, pParms, sizeof(abyBuffer));
```

#### See also:

*emInitMaster()*

### 6.3.4 emSetMasterParms

```
static EC_T_DWORD ecatSetMasterParms (EC_T_INIT_MASTER_PARMS *pParms)
```

```
EC_T_DWORD emSetMasterParms (
    EC_T_DWORD dwInstanceID,
    EC_T_INIT_MASTER_PARMS *pParms
)
    Change Master initialization parameters.
```

Currently the following parameters cannot be changed:

- *EC\_T\_INIT\_MASTER\_PARMS.pOsParms*
- *EC\_T\_INIT\_MASTER\_PARMS.pLinkParms*

- *EC\_T\_INIT\_MASTER\_PARMS.pLinkParmsRed*
- *EC\_T\_INIT\_MASTER\_PARMS.dwMaxBusSlaves*
- *EC\_T\_INIT\_MASTER\_PARMS.dwMaxAcycFramesQueued*
- *EC\_T\_INIT\_MASTER\_PARMS.dwAdditionalEoEEndpoints*

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pParams** – [in] New Master parameters

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized

### **emSetMasterParms() Example**

```
EC_T_BYTE abyBuffer[sizeof(EC_T_INIT_MASTER_PARMS) + sizeof(EC_T_OS_PARMS)
    + 512 /* LinkLayer parameters */];
EC_T_INIT_MASTER_PARMS* pParams = (EC_T_INIT_MASTER_PARMS*)abyBuffer;
dwRes = emGetMasterParms(dwInstanceId, pParams, sizeof(abyBuffer));
pParams->wReserved = 1;
/* change Master initialization parameters */
dwRes = emSetMasterParms(dwInstanceId, pParams);
```

#### See also:

[emInitMaster\(\)](#)

### **6.3.5 emScanBus**

static EC\_T\_DWORD **ecatScanBus** (EC\_T\_DWORD dwTimeout)

EC\_T\_DWORD **emScanBus** (EC\_T\_DWORD dwInstanceId, EC\_T\_DWORD dwTimeout)

Scans all connected slaves.

Scans all connected slaves connected to EC-Master. If a configuration has been loaded, a validation between the configuration and the connected slaves is done. This function should not be called from within the JobTask's context.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwTimeout** – [in] Timeout [ms]

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceId is out of range
- *EC\_E\_LINK\_DISCONNECTED* if link is disconnected
- *EC\_E\_TIMEOUT* if dwTimeout elapsed during the API call

- *EC\_E\_BUSCONFIG\_MISMATCH* if the slaves found are not matching the configured ones
- *EC\_E\_LINE\_CROSSED* if a line crossed (cabling wrong) condition has been detected
- *EC\_E\_REDLINEBREAK* if cable redundancy is configured and a line break condition has been detected
- *EC\_E\_JUNCTION\_RED\_LINE\_BREAK* if junction redundancy is configured and a line break condition has been detected
- *EC\_E\_MAX\_BUS\_SLAVES\_EXCEEDED* if the amount of slaves found exceeds *EC\_T\_INIT\_MASTER\_PARMS.dwMaxBusSlaves*
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_AMS\_IS\_RUNNING* if ADS server is running

### **emScanBus() Example**

```
dwRes = emScanBus(dwInstanceId, 5000 /* timeout */);
```

#### See also:

- *EtherCAT Bus Scan*

## 6.3.6 emRescueScan

static EC\_T\_DWORD **ecatRescueScan** (EC\_T\_DWORD dwTimeout)

EC\_T\_DWORD **emRescueScan** (EC\_T\_DWORD dwInstanceId, EC\_T\_DWORD dwTimeout)

Recover the bus from permanent frame loss situations.

Scans all connected slaves. Closes and open ports on the network to rule out slaves which permanently discard frames. The Master notifies every slave port which permanently discard frames with *EC\_NOTIFY\_FRAMELOSS\_AFTER\_SLAVE*. Due to port opening and closing the scanning time is increased about 2 seconds per slave. The Master will not automatically re-open this port. The application can force to open the port again. This function may not be called from within the JobTask's context.

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwTimeout** – [in] Timeout [ms]

#### Returns

*EC\_E\_NOERROR* or error code

### **emRescueScan() Example**

```
dwRes = emRescueScan(dwInstanceId, 5000 /* timeout */);
```

#### See also:

- *emSetSlavePortState()*
- *emNotify - EC\_NOTIFY\_FRAMELOSS\_AFTER\_SLAVE*

### 6.3.7 emConfigureNetwork

```
static EC_T_DWORD ecatConfigureNetwork (
    EC_T_CNF_TYPE eCnfType,
    EC_T_PBYTE pbyCnfData,
    EC_T_DWORD dwCnfDataLen
)
EC_T_DWORD emConfigureNetwork (
    EC_T_DWORD dwInstanceID,
    EC_T_CNF_TYPE eCnfType,
    EC_T_PBYTE pbyCnfData,
    EC_T_DWORD dwCnfDataLen
)
Configure the Network.
```

This function must be called after the initialization. Among others the EtherCAT topology defined in the given XML configuration file will be stored internally.

Analyzing the network including mailbox communication can be done without specifying an ENI file using eCnfType\_GenPreopENI.

---

**Note:** A client must not be registered prior to calling this function. Existing client registrations will be dropped.

---

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eCnfType** – [in] Type of configuration data provided
- **pbyCnfData** – [in] Filename / configuration data, or EC\_NULL if eCnfType is eCnfType\_GenPreopENI
- **dwCnfDataLen** – [in] Length of configuration data in byte, or zero if eCnfType is eCnfType\_GenPreopENI

#### Returns

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master isn't initialized or eCnfType is eCnfType\_GenPreopENI or eCnfType\_GenOpENI and link is disconnected
- **EC\_E\_INVALIDPARM** if dwInstanceID is out of range or pParms is EC\_NULL contains some values out of range
- **EC\_E\_LINK\_DISCONNECTED** if link is disconnected
- **EC\_E\_FEATURE\_DISABLED** if a configured feature is not included in the license key
- **EC\_E\_NOTSUPPORTED** if a configured feature is not supported (e.g not compiled in the library)
- **EC\_E\_CFGFILENOTFOUND** if the ENI file cannot be found
- **EC\_E\_WRONG\_FORMAT** if some format error have been detected in the ENI or EEPROM in case of eCnfType\_GenPreopENI or eCnfType\_GenOpENI
- **EC\_E\_OEM\_SIGNATURE\_MISMATCH** if the OEM signature in the ENI file doesn't match the used OEM key

- *EC\_E\_ENI\_ENCRYPTION\_WRONG\_VERSION* if the ENI encryption version is not supported (e.g. the library is too old)
- *EC\_E\_ENI\_ENCRYPTED* if the ENI is encrypted and no OEM key has been set
- *EC\_E\_XML\_CYCCMDS\_MISSING* if the ENI doesn't contain cyclic commands
- *EC\_E\_XML\_ALSTATUS\_READ\_MISSING* if the ENI doesn't contain any read AL status command
- *EC\_E\_XML\_CYCCMDS\_SIZEMISMATCH* if the size of the cyclic commands in the ENI mismatch
- *EC\_E\_XML\_INVALID\_INP\_OFF* if some input offset in the ENI are invalid
- *EC\_E\_XML\_INVALID\_OUT\_OFF* if some output offset in the ENI are invalid
- *EC\_E\_XML\_INVALID\_CMD\_WITH\_RED* if the ENI contains LRW commands and cable redundancy is configured
- *EC\_E\_XML\_PREV\_PORT\_MISSING* if some previous port information are missing in the ENI
- *EC\_E\_XML\_DC\_CYCCMDS\_MISSING* if the DC related cyclic commands are missing in the ENI
- *EC\_E\_XML\_AOE\_NETID\_INVALID* if the ENI contains some invalid NetID

enum **EC\_T\_CNF\_TYPE**

*Values:*

enumerator **eCnfType\_Unknown**

enumerator **eCnfType\_Filename**

    pbyCnfData: ENI filename to read

enumerator **eCnfType\_Data**

    pbyCnfData: ENI data

enumerator **eCnfType\_Datadiag**

    pbyCnfData: ENI data for diagnosis

enumerator **eCnfType\_GenPreopENI**

    Generate ENI based on bus-scan result to get into PREOP state

enumerator **eCnfType\_GenPreopENIWithCRC**

    Same as eCnfType\_GenPreopENI with CRC protection

enumerator **eCnfType\_GenOpENI**

    Generate ENI based on bus-scan result to get into OP state. The default PDO mapping read from the slaves is activated. See ETG2010 “SII Specification”, Table 14 “Structure Category TXPDO and RXPDO for each PDO”

enumerator **eCnfType\_None**

    Reset configuration

enumerator **eCnfType\_ConfigData**

    pbyCnfData: Binary structured configuration

enumerator **eCnfType\_GenOpENINoStrings**

    Generate ENI based on bus-scan result to get into OP state , does not read strings from EEPROM

**enumerator eCnfType\_FileByApp**

File access provided by user application, See *EC\_T\_CNF\_FILEBYAPP\_DESC*

**enumerator eCnfType\_GenEBI**

Generate EBI based on bus-scan result

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

struct **EC\_T\_CNF\_FILEBYAPP\_DESC**

**Public Members****EC\_T\_VOID \*pvContext**

[in] Arbitrarily application-defined parameter passed to callbacks

**EC\_PF\_CNF\_OPEN \*pfnFileOpen**

[in] Function pointer called instead of OsCfgFileOpen()

**EC\_PF\_CNF\_CLOSE \*pfnFileClose**

[in] Function pointer called instead of OsCfgFileClose()

**EC\_PF\_CNF\_READ \*pfnFileRead**

[in] Function pointer called instead of OsCfgFileRead()

**EC\_PF\_CNF\_ERROR \*pfnFileError**

[in] Function pointer called instead of OsCfgFileError()

**EC\_PF\_CNF\_EOF \*pfnFileEof**

[in] Function pointer called instead of OsCfgFileEof()

typedef EC\_T\_DWORD **EC\_PF\_CNF\_OPEN** (EC\_T\_VOID \*pvContext)

Called by the EtherCAT stack instead of OsCfgFileOpen() within emConfigureNetwork(eCnfType\_FileByApp)

**Parameters**

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

**Returns**

*EC\_E\_NOERROR* or error code

typedef EC\_T\_DWORD **EC\_PF\_CNF\_CLOSE** (EC\_T\_VOID \*pvContext)

Called by the EtherCAT stack instead of OsCfgFileClose() within emConfigureNetwork(eCnfType\_FileByApp)

**Parameters**

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

**Returns**

*EC\_E\_NOERROR* or error code

typedef EC\_T\_DWORD **EC\_PF\_CNF\_READ** (

**EC\_T\_VOID** \*pvContext,

**EC\_T\_BYTE** \*pbyReadData,

**EC\_T\_DWORD** dwReadLen,

**EC\_T\_DWORD** \*pdwNumOutData

)

Called by the EtherCAT stack instead of OsCfgFileRead() within emConfigureNetwork(eCnfType\_FileByApp)

**Parameters**

- **pvContext** – [in] Arbitrarily application-defined parameter passed to callback
- **pbyReadData** – [out] Pointer to the data read
- **dwReadLen** – [in] Amount of bytes to be read by the EtherCAT stack (next part of the configuration file)
- **pdwNumOutData** – [out] Amount of bytes written to pbyReadData by callback

**Returns***EC\_E\_NOERROR* or error codetypedef EC\_T\_DWORD **EC\_PF\_CNF\_ERROR** (EC\_T\_VOID \*pvContext)

Called by the EtherCAT stack instead of OsCfgFileError() within emConfigureNetwork(eCnfType\_FileByApp)

**Parameters**

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

**Returns***EC\_E\_NOERROR* or error codetypedef EC\_T\_DWORD **EC\_PF\_CNF\_EOF** (EC\_T\_VOID \*pvContext, EC\_T\_BOOL \*bEof)

Called by the EtherCAT stack instead of OsCfgFileEof() within emConfigureNetwork(eCnfType\_FileByApp)

**Parameters**

- **pvContext** – [in] Arbitrarily application-defined parameter passed to callback
- **bEof** – [out] Indicates whether the end of the file has been reached (EC\_TRUE if EOF, EC\_FALSE otherwise).

**Returns***EC\_E\_NOERROR* or error code**emConfigureNetwork() Example**

```
/* load ENI */
const EC_T_CHAR* szFileName = "eni.xml";
dwRes = emConfigureNetwork(dwInstanceId, eCnfType_Filename,
    (EC_T_BYTE*)szFileName, (EC_T_DWORD)OsStrlen(szFileName));
```

**6.3.8 emConfigGet**

```
static EC_T_DWORD ecatConfigGet (
    EC_T_BYTE **ppbyCnfData,
    EC_T_DWORD *pdwCnfDataLen
)
EC_T_DWORD emConfigGet (
    EC_T_DWORD dwInstanceID,
    EC_T_BYTE **ppbyCnfData,
    EC_T_DWORD *pdwCnfDataLen
)
Get the master configuration.
```

This function returns the result of ENI parsing in binary format. This data can be stored at a different location (e.g. read only flash). Later on, the Master can be configured without ENI using the type EC\_T\_CNF\_TYPE::eCnfType\_ConfigData.

---

**Note:** The binary format is not version independent and the data becomes invalid, when used with a different version. The returned pointer is valid as long as the Master is initialized and no other configuration was loaded.

---

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **ppbyCnfData** – [out] Configuration data
- **pdwCnfDataLen** – [out] Length of configuration data in byte

### Returns

*EC\_E\_NOERROR* or error code

### **emConfigGet()** Example

```

EC_T_BYTE* pbyConfigData = EC_NULL;
EC_T_DWORD dwConfigDataSize = 0;

/* set config data memory pool */
EC_IOCTL_SET_CONFIGDATA_MEMORY_POOL_DESC oPoolDesc;
oPoolDesc.dwSize = 100000; /* max config file size */
oPoolDesc.pbyStart = (EC_T_BYTE*)OsMalloc(oPoolDesc.dwSize);
dwRes = emIoCtl(dwInstanceId, EC_IOCTL_SET_CONFIGDATA_MEMORY_POOL,
    &oPoolDesc, sizeof(EC_IOCTL_SET_CONFIGDATA_MEMORY_POOL_DESC),
    EC_NULL, 0, EC_NULL);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    goto Exit;
}

/* load config */
dwRes = emConfigLoad(dwInstanceId, eCnfType_Filename,
    (EC_T_BYTE*)szFileName, (EC_T_DWORD)OsStrlen(szFileName));
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    goto Exit;
}

/* get config */
dwRes = emConfigGet(dwInstanceId, &pbyConfigData, &dwConfigDataSize);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    goto Exit;
}

```

### See also:

[emConfigureNetwork\(\)](#)

### 6.3.9 emConfigExtend

**Warning:** Before using this function, please check if the following patents has to be taken into consideration for your application and use case: **JP5212509:ADDRESS SETTING METHOD IN NETWORK SYSTEM**

```
static EC_T_DWORD ecatConfigExtend (
    EC_T_BOOL bResetConfig,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emConfigExtend (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bResetConfig,
    EC_T_DWORD dwTimeout
)
Extends the existing network configuration.
```

This function extends the existing configuration described in the ENI to allow mailbox communication with unexpected slaves. After this function was called, unexpected slaves can reach PREOP state. After the configuration was extended, disconnecting any slave will generate a bus mismatch, because all the slaves are part of the configuration. Recalling this function with bResetConfig set to EC\_FALSE will extend the configuration again by any new connected unexpected slaves. The previous extension is not deleted. Calling the function with bResetConfig set to EC\_TRUE, reset all the previous extensions.

---

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

---

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bResetConfig** – [in] EC\_TRUE: Extended configuration will be removed
- **dwTimeout** – [in] Timeout [ms]

#### Returns

*EC\_E\_NOERROR* or error code

#### emConfigExtend() Example

```
dwRes = emConfigExtend(dwInstanceId, EC_TRUE, 5000 /* timeout */);
```

### 6.3.10 emRegisterClient

```
static EC_T_DWORD ecatRegisterClient (
    EC_PF_NOTIFY pfnNotify,
    EC_T_VOID *pCallerData,
    EC_T_REGISTERRESULTS *pRegResults
)
EC_T_DWORD emRegisterClient (
    EC_T_DWORD dwInstanceId,
    EC_PF_NOTIFY pfnNotify,
    EC_T_VOID *pCallerData,
    EC_T_REGISTERRESULTS *pRegResults
)
```

Register the application as a client with the EtherCAT stack for notifications (EC\_NOTIFY\_... )

It must be called after configuration, otherwise the registration handle is lost. This function may not be called from within the JobTask's context.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pfnNotify** – [in] Notification callback function. This function will be called every time a state change occurs, an error occurs or a mailbox transfer terminates.
- **pCallerData** – [in] Parameter passed to the application when the notification callback is called. The parameter can be arbitrarily defined by the application.
- **pRegResults** – [out] Registration results

### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC\_NULL
- *EC\_E\_NOMEMORY* if some memory cannot be allocated

typedef EC\_T\_DWORD (\***EC\_PF\_NOTIFY**)(EC\_T\_DWORD dwCode, *EC\_T\_NOTIFYPARMS* \*pParms)

### Parameters

- **dwCode** – [in] Notification code, see EC\_NOTIFY\_...
- **pParams** – [in] Notification code depending data.

struct **EC\_T\_REGISTERRESULTS**

### Public Members

**EC\_T\_DWORD dwClnId**  
[out] Client ID

**EC\_T\_BYTExpbyPDIn**  
[out] Pointer to process data input memory

**EC\_T\_DWORD dwPDIInSize**  
[out] Size of process data input memory (in bytes)

**EC\_T\_BYTExpbyPDOOut**  
[out] Pointer to process data output memory

**EC\_T\_DWORD dwPDOOutSize**  
[out] Size of process data output memory (in bytes)

**emRegisterClient() Example**

```
EC_T_REGISTERRESULTS oRegResults;
OsMemset(&oRegResults, 0, sizeof(EC_T_REGISTERRESULTS));
dwRes = emRegisterClient(dwInstanceId, myAppNotify, pvMyAppContext, &oRegResults);
```

**6.3.11 emUnregisterClient**

static EC\_T\_DWORD **ecatUnregisterClient** (EC\_T\_DWORD dwClntId)

**EC\_T\_DWORD emUnregisterClient** (EC\_T\_DWORD dwInstanceId, EC\_T\_DWORD dwClntId)  
Deregister a client from the EtherCAT stack.

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

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClntId** – [in] Client ID from registration with the EtherCAT stack

**Returns**

*EC\_E\_NOERROR* or error code

**6.3.12 emGetSrcMacAddress**

static EC\_T\_DWORD **ecatGetSrcMacAddress** (ETHERNET\_ADDRESS \*pMacSrc)

**EC\_T\_DWORD emGetSrcMacAddress** (  
    EC\_T\_DWORD dwInstanceId,  
    ETHERNET\_ADDRESS \*pMacSrc  
)

Gets the source MAC address.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMacSrc** – [out] 6-byte buffer to write source MAC address to.

**Returns**

*EC\_E\_NOERROR* or error code

**See also:**

*EC\_T\_INIT\_MASTER\_PARMS::pLinkParms*

**emGetSrcMacAddress() Example**

```
/* get MAC address of EtherCAT network adapter */
ETHERNET_ADDRESS oMacSrc;
OsMemset(&oMacSrc, 0, sizeof(ETHERNET_ADDRESS));
dwRes = emGetSrcMacAddress(dwInstanceId, &oMacSrc);
```

### 6.3.13 emSetMasterState

static EC\_T\_DWORD **ecatSetMasterState** (EC\_T\_DWORD dwTimeout, *EC\_T\_STATE* eReqState)

```
EC_T_DWORD emSetMasterState (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwTimeout,
    EC_T_STATE eReqState
)
```

) Set the master (and all slaves) into the requested EtherCAT state.

If the function is called with EC\_NOWAIT, the client may wait for reaching the requested state using the notification callback (EC\_NOTIFY\_STATECHANGED).

Master by default will just change to a higher state, if all slaves have reached the requested state. It may happen that some slaves are in higher state at network than Master, e.g.:

- Master and all slaves are in PREOP
- Application requests SAFEOP
- Master starts transition for all slaves
- Some slaves changed to SAFEOP, but some fail and therefore stay in PREOP
- Master state stays in PREOP, function returns with error

The application can request SAFEOP again to re-request state of previously failed slaves. Transition to lower state: The master changes to lower state, even if one slave is not able to follow. This function may not be called from within the JobTask's context with dwTimeout other than EC\_NOWAIT.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwTimeout** – [in] Timeout [ms]. This function will block until the requested state is reached or the timeout elapsed. If the timeout value is set to EC\_NOWAIT the function will return immediately.
- **eReqState** – [in] Requested System state

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range
- *EC\_E\_LINK\_DISCONNECTED* if link is disconnected
- *EC\_E\_TIMEOUT* if dwTimeout elapsed during the API call
- *EC\_E\_BUSCONFIG\_MISMATCH* if the slaves found are not matching the configured ones
- *EC\_E\_LINE\_CROSSED* if a line crossed (cabling wrong) condition has been detected
- *EC\_E\_REDLINEBREAK* if cable redundancy is configured and a line break condition has been detected
- *EC\_E\_JUNCTION\_RED\_LINE\_BREAK* if junction redundancy is configured and a line break condition has been detected
- *EC\_E\_MAX\_BUS\_SLAVES\_EXCEEDED* if the amount of slaves found exceeds *EC\_T\_INIT\_MASTER\_PARMS.dwMaxBusSlaves*

- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_AMS\_IS\_RUNNING* if AMS server is running

enum **EC\_T\_STATE**

*Values:*

enumerator **eEcatState\_UNKNOWN**  
Unknown state

enumerator **eEcatState\_INIT**  
EtherCAT state INIT

enumerator **eEcatState\_PREOP**  
EtherCAT state PREOP (pre-operational)

enumerator **eEcatState\_SAFEOP**  
EtherCAT state SAFEOP (safe operational)

enumerator **eEcatState\_OP**  
EtherCAT state OP (operational)

enumerator **eEcatState\_BOOTSTRAP**  
EtherCAT state BOOTSTRAP

### **emSetMasterState() Example**

```
/* set EtherCAT master (and all slaves) into requested state */
dwRes = emSetMasterState(dwInstanceId, 5000, eEcatState_PREOP);
```

**See also:**

*emIoControl - EC\_IOCTL\_ALL\_SLAVES\_MUST\_REACH\_MASTER\_STATE*

#### **6.3.14 emSetMasterStateReq**

```
static EC_T_DWORD ecatSetMasterStateReq (
    EC_T_DWORD dwTimeout,
    EC_T_STATE eReqState
)
EC_T_DWORD emSetMasterStateReq (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwTimeout,
    EC_T_STATE eReqState
)
```

Request to set the master (and all slaves) into the requested EtherCAT state and return immediately.

The Master by default will just change to a higher state, if all slaves have reached the requested state. It may happen that some slaves are in higher state at network than Master, e.g.:

- Master and all slaves are in PREOP
- Application requests SAFEOP
- Master starts transition for all slaves
- Some slaves changed to SAFEOP, but some fail and therefore stay in PREOP

- Master state stays in PREOP, function returns with error

The application can request SAFEOP again to re-request state of previously failed slaves. Transition to lower state: The master changes to lower state, even if one slave is not able to follow.

See also EC\_NOTIFY\_STATECHANGED.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwTimeout** – [in] Timeout [ms]. This function will block until the requested state is reached or the timeout elapsed. If the timeout value is set to EC\_NOWAIT the function will return immediately.
- **eReqState** – [in] Requested System state

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range
- *EC\_E\_LINK\_DISCONNECTED* if link is disconnected
- *EC\_E\_TIMEOUT* if dwTimeout elapsed during the API call
- *EC\_E\_BUSCONFIG\_MISMATCH* if the slaves found are not matching the configured ones
- *EC\_E\_LINE\_CROSSED* if a line crossed (cabling wrong) condition has been detected
- *EC\_E\_REDLINEBREAK* if cable redundancy is configured and a line break condition has been detected
- *EC\_E\_JUNCTION\_RED\_LINE\_BREAK* if junction redundancy is configured and a line break condition has been detected
- *EC\_E\_MAX\_BUS\_SLAVES\_EXCEEDED* if the amount of slaves found exceeds *EC\_T\_INIT\_MASTER\_PARMS.dwMaxBusSlaves*
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_AMS\_IS\_RUNNING* if ADS server is running

#### **emSetMasterStateReq() Example**

```
/* set EtherCAT master (and all slaves) into requested state */
dwRes = emSetMasterStateReq(dwInstanceId, 5000, eEcatState_PREOP);
```

#### See also:

*emIoControl - EC\_IOCTL\_ALL\_SLAVES\_MUST\_REACH\_MASTER\_STATE*

#### See also:

*emSetMasterState()*

### 6.3.15 emGetMasterState

```
static EC_T_STATE ecatGetMasterState (EC_T_VOID)
EC_T_STATE emGetMasterState (EC_T_DWORD dwInstanceID)
    Get the EtherCAT master current state.
```

**Parameters**

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

**Returns**

EtherCAT master state

#### emGetMasterState() Example

```
EC_T_STATE eMasterState = emGetMasterState(dwInstanceId);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
    "Current Master State: %s:\n", ecatStateToStr(eMasterState)));
```

### 6.3.16 emGetMasterStateEx

```
static EC_T_DWORD ecatGetMasterStateEx (
    EC_T_WORD *pwCurrState,
    EC_T_WORD *pwReqState
)
EC_T_DWORD emGetMasterStateEx (
    EC_T_DWORD dwInstanceID,
    EC_T_WORD *pwCurrState,
    EC_T_WORD *pwReqState
)
    Get the EtherCAT master current and requested state. Possible return values for current and requested state:
```

- *DEVICE\_STATE\_UNKNOWN*
- *DEVICE\_STATE\_INIT*
- *DEVICE\_STATE\_PREOP*
- *DEVICE\_STATE\_SAFEOP*
- *DEVICE\_STATE\_OP*

**Parameters**

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

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or the output pointers are *EC\_NULL*

**emGetMasterStateEx() Example**

```
EC_T_WORD wCurrState = 0;
EC_T_WORD wReqState = 0;
/* get EtherCAT master current state */
dwRes = emGetMasterStateEx(dwInstanceId, &wCurrState, &wReqState);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
    "Current state: %s, requested state: %s\n",
    ecatDeviceStateText(wCurrState), ecatDeviceStateText(wReqState)));
```

**6.3.17 emStart**

static EC\_T\_DWORD **ecatStart** (EC\_T\_DWORD dwTimeout)

**EC\_T\_DWORD emStart** (EC\_T\_DWORD dwInstanceId, EC\_T\_DWORD dwTimeout)

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

*Deprecated:*

Use *emSetMasterState()* instead

---

**Note:** 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.

---

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwTimeout** – [in] Timeout [ms] 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.

**Returns**

*EC\_E\_NOERROR* or error code

**emStart() Example**

```
dwRes = emStart(dwInstanceId, 5000 /* timeout */);
```

**6.3.18 emStop**

static EC\_T\_DWORD **ecatStop** (EC\_T\_DWORD dwTimeout)

**EC\_T\_DWORD emStop** (EC\_T\_DWORD dwInstanceId, EC\_T\_DWORD dwTimeout)

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

*Deprecated:*

Use *emSetMasterState()* instead

---

**Note:** If the function is called with EC\_NOWAIT, the client may wait for reaching the INIT state using the notification callback (ECAT\_NOTIFY\_STATECHANGE). This function may not be called from within the JobTask's context.

---

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwTimeout** – [in] Timeout [ms] 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.

### Returns

*EC\_E\_NOERROR* or error code

### **emStop() Example**

```
dwRes = emStop(dwInstanceId, 5000 /* timeout */);
```

## 6.3.19 emExecJob

```
static EC_T_DWORD ecatExecJob (
    EC_T_USER_JOB eUserJob,
    EC_T_USER_JOB_PARMS *pUserJobParms
)
EC_T_DWORD emExecJob (
    EC_T_DWORD dwInstanceId,
    EC_T_USER_JOB eUserJob,
    EC_T_USER_JOB_PARMS *pUserJobParms
)
```

Execute or initiate the requested master job.

To achieve maximum speed, this function is implemented non re-entrant. It is highly recommended that only one single task is calling all required jobs to run the stack. If multiple tasks are calling this function, the calls have to be synchronized externally. Calling it in a context that doesn't support operating system calls can lead to unpredictable behavior.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eUserJob** – [in] user requested job
- **pUserJobParms** – [in] optional user job parameters

### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceId is out of range or the output pointer is EC\_NULL
- *EC\_E\_LINK\_DISCONNECTED* if the link is disconnected

- *EC\_E\_FEATURE\_DISABLED* for eUsrJob\_SwitchEoeFrames if EC\_IOCTL\_SET\_EOE\_DEFERRED\_SWITCHING\_ENABLED hasn't be called before
- *EC\_E\_ADS\_IS\_RUNNING* if ADS server is running

*Brief job overview:*

enum **EC\_T\_USER\_JOB**

*Values:*

```
enumerator eUsrJob_Undefined
enumerator eUsrJob_ProcessAllRxFrames
enumerator eUsrJob_SendAllCycFrames
enumerator eUsrJob_MasterTimer
enumerator eUsrJob_SendAcycFrames
enumerator eUsrJob_SendCycFramesByTaskId
enumerator eUsrJob_MasterTimerMinimal
enumerator eUsrJob_ProcessRxFramesByTaskId
enumerator eUsrJob_ProcessAcycRxFrames
enumerator eUsrJob_SwitchEoeFrames
enumerator eUsrJob_StartTask
enumerator eUsrJob_StopTask
enumerator eUsrJob_StampSendAllCycFrames
enumerator eUsrJob_StampSendCycFramesByTaskId
enumerator eUsrJob_SimulatorTimer
enumerator eUsrJob_MonitorTimer
```

union **EC\_T\_USER\_JOB\_PARMS**

### Public Members

```
EC_T_BOOL bAllCycFramesProcessed
EC_T_DWORD dwNumFramesSent
EC_T_DWORD dwTaskIdToSend
struct EC_T_USER_JOB_PARMS::_SEND_CYCFRAME_BY_TASKID SendCycFramesByTaskId
struct EC_T_USER_JOB_PARMS::_PROCESS_RXFRAME_BY_TASKID ProcessRxFramesByTaskId
struct EC_T_USER_JOB_PARMS::_SWITCH_EOE_FRAMES SwitchEoeFrames
struct EC_T_USER_JOB_PARMS::_START_TASK StartTask
struct EC_T_USER_JOB_PARMS::_STOP_TASK StopTask
struct _PROCESS_RXFRAME_BY_TASKID
```

**Public Members**

```
EC_T_BOOL bCycFramesProcessed
EC_T_DWORD dwTaskId
struct _SEND_CYCFRAME_BY_TASKID
```

**Public Members**

```
EC_T_DWORD dwTaskId
struct _START_TASK
```

**Public Members**

```
EC_T_DWORD dwTaskId
struct _STOP_TASK
```

**Public Members**

```
EC_T_DWORD dwTaskId
struct _SWITCH_EOE_FRAMES
```

**Public Members**

```
EC_T_DWORD dwMaxPortsToProcess
EC_T_DWORD dwNumFramesProcessed
```

*Detailed job description:*

1. **eUsrJob\_ProcessAllRxFrames**

When the Real-time Ethernet Driver operates in polling mode this call will process all currently received frames, when the Real-time Ethernet Driver operates in interrupt mode all received frames are processed immediately and this call just returns with nothing done.

pUserJobParms->bAllCycFramesProcessed

This flag is set to a value of EC\_TRUE it indicates that all previously initiated cyclic frames ([eUsrJob\\_SendAllCycFrames](#)) are received and processed within this call. Not used if pUserJobParms set to EC\_NULL.

Return: EC\_E\_NOERROR if successful, error code in case of failures.

2. **eUsrJob\_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 Real-time Ethernet Driver operates in interrupt mode, the process data input values will be updated immediately after receiving the frames. If the Real-time Ethernet Driver operates in polling mode, the next call to [emExecJob\(\)](#) with the [eUsrJob\\_ProcessAllRxFrames](#) job will check for received frames and update the process data input values.

pUserJobParms->dwNumFramesSent

Indicates number of frames send within this call. Not used if pUserJobParms set to EC\_NULL.

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 (*emNotify - EC\_NOTIFY\_FRAME\_RESPONSE\_ERROR*).

### 3. ***eUsrJob\_SendAcycFrames***

Acyclic EtherCAT datagrams stored in the acyclic frame buffer FIFO will be sent when executing this call.

pUserJobParms->dwNumFramesSent

Indicates number of frames send within this call. Not used if pUserJobParms set to EC\_NULL.

Return: EC\_E\_NOERROR if successful, error code in case of failures.

### 4. ***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 *emExecJob()* (*eUsrJob\_MasterTimer*). The state-machines are handling the EtherCAT state change transfers.

Return: EC\_E\_NOERROR if successful, error code in case of failures.

### 5. ***eUsrJob\_SendCycFramesByTaskId***

Send cyclic frames related to a specific task id. If more than one cyclic entries are configured 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.

**See also:**

*Multiple cyclic entries configuration*

pUserJobParms->SendCycFramesByTaskId.dwTaskId

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 (*emNotify - EC\_NOTIFY\_FRAME\_RESPONSE\_ERROR*).

### 6. ***eUsrJob\_ProcessRxFramesByTaskId***

*eUsrJob\_ProcessAcycRxFrames*

**See also:**

Feature-Pack Split Frame Processing

### 7. ***eUsrJob\_SwitchEoeFrames***

This job must be called if *emIoControl - EC\_IOCTL\_SET\_EOE\_DEFERRED\_SWITCHING\_ENABLED* has been called before. It can be called in parallel to Send / Process jobs in a lower prioritized task

pUserJobParms->SwitchEoeFrames.dwMaxPortsToProcess

Indicates number of ports to be processed within this call. If zero, all ports will be processed.

pUserJobParms->SwitchEoeFrames.dwNumFramesProcessed

Returns number of frames processed within this call.

Return: EC\_E\_NOERROR if successful

### 8. ***eUsrJob\_StartTask***

Inform EC-Master that the current task is started. Specify pUserJobParms.StartTask.dwTaskId or pass pUserJobParms set to EC\_NULL for task ID 0.

#### 9. **eUsrJob\_StopTask**

Inform EC-Master that the current task is stopped. Specify pUserJobParms.StopTask.dwTaskId or pass pUserJobParms set to EC\_NULL for task ID 0.

#### **emExecJob() Example**

```
EC_T_USER_JOB oUserJob;
OsMemset(&oUserJob, 0, sizeof(EC_T_USER_JOB));
oUserJob = eUsrJob_StartTask;
EC_T_USER_JOB_PARMS oUserJobParms;
OsMemset(&oUserJobParms, 0, sizeof(EC_T_USER_JOB));
dwRes = emExecJob(dwInstanceId, oUserJob, &oUserJobParms);
```

### 6.3.20 **emGetVersion**

```
static EC_T_DWORD ecatGetVersion (
    EC_T_DWORD *pdwVersion,
    EC_T_DWORD *pdwVersionType
)
EC_T_DWORD emGetVersion (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD *pdwVersion,
    EC_T_DWORD *pdwVersionType
)
```

Gets the version information.

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pdwVersion** – [out] Pointer to EC\_T\_DWORD to carry out version number as a 32-bit value
- **pdwVersionType** – [out] Pointer to EC\_T\_DWORD to carry out version type. See [EC\\_VERSION\\_TYPE](#)

#### Returns

- [EC\\_E\\_NOERROR](#) if successful
- [EC\\_E\\_INVALIDSTATE](#) if master isn't initialized
- [EC\\_E\\_INVALIDPARAM](#) if dwInstanceId is out of range or the output pointer is EC\_NULL

#### EC Version Type

**EC\_VERSION\_TYPE\_UNDEFINED**  
EC-Master Type Undefined

**EC\_VERSION\_TYPE\_UNRESTRICTED**  
EC-Master Type Unrestricted

**EC\_VERSION\_TYPE\_PROTECTED**  
EC-Master Type Protected

**EC\_VERSION\_TYPE\_DONGLED**  
EC-Master Type Dongled

**EC\_VERSION\_TYPE\_EVAL**  
EC-Master Type Eval

### emGetVersion() Example

```
/* get stack version */
EC_T_DWORD dwVersion = EC_E_ERROR;
EC_T_DWORD dwVersionType = 0;
dwRes = emGetVersion(dwInstanceId, &dwVersion, &dwVersionType);
```

## 6.3.21 emSetLicenseKey

static EC\_T\_DWORD **ecatSetLicenseKey** (const EC\_T\_CHAR \*pszLicenseKey)

```
EC_T_DWORD emSetLicenseKey (
    EC_T_DWORD dwInstanceId,
    const EC_T_CHAR *szLicenseKey
)
```

Sets the license key for the protected version of EC-Master.

Must be called after initialization and before configuration. This function may not be called if a non protected version is used.

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szLicenseKey** – [in] License key as zero terminated string with 26, 53 or 56 characters.

#### Returns

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master isn't initialized
- **EC\_E\_INVALIDPARAM** if dwInstanceId is out of range
- **EC\_E\_INVALIDSIZE** the format of the license key is wrong. The correct length is 26, 53 or 56 characters
- **EC\_E\_LICENSE\_MISSING** the license key doesn't match the MAC Address

### emSetLicenseKey() Example

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

#### See also:

- *emInitMaster()*
- *emConfigureNetwork()*

### 6.3.22 emSetOemKey

```
static EC_T_DWORD ecatSetOemKey (EC_T_UINT64 qwOemKey)
static EC_T_DWORD emSetOemKey (EC_T_DWORD dwInstanceId, EC_T_UINT64 qwOemKey)
    Provide OEM Key needed for OEM Masters to parse ENI files and provide access via RAS. Must be called
    after initialization and before configuration.
```

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **qwOemKey** – [in] 64 bit OEM key

#### emSetOemKey() Example

```
dwRes = emSetOemKey(dwInstanceId, 0x1234567812345678);
```

#### See also:

- [\*emInitMaster\(\)\*](#)
- [\*emConfigureNetwork\(\)\*](#)

### 6.3.23 emIoControl

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

```
EC_T_DWORD emIoControl (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwCode,
    EC_T_IOCTLPARMS *pParms
)
```

A generic control interface between the application, the EtherCAT stack and its Real-time Ethernet Drivers.

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwCode** – [in] Control code
- **pParms** – [in/out] Control code depending parameters

#### Returns

- [\*EC\\_E\\_NOERROR\*](#) if successful
- [\*EC\\_E\\_INVALIDSTATE\*](#) if master isn't initialized
- [\*EC\\_E\\_INVALIDPARAM\*](#) if dwInstanceId is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer
- [\*EC\\_E\\_NOMEMORY\*](#) if some memory cannot be allocated
- [\*EC\\_E\\_ADS\\_IS\\_RUNNING\*](#) if ADS server is running

```
struct EC_T_IOCTLPARMS
```

**Public Members**

**EC\_T\_BYTE \*pbyInBuf**  
 [in] Pointer to control input parameter.

**EC\_T\_DWORD dwInBufSize**  
 [in] Size of the input buffer provided at pbyInBuf in bytes

**EC\_T\_BYTE \*pbyOutBuf**  
 [out] Pointer to control output buffer where the results will be copied into

**EC\_T\_DWORD dwOutBufSize**  
 [in] Size of the output buffer provided at pbyOutBuf in bytes

**EC\_T\_DWORD \*pdwNumOutData**  
 [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer

**6.3.24 emIoControl - 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 `emConfigureNetWork()` and before `emStart()`.

**emIoControl - EC\_IOCTL\_GET\_PDMEMORYSIZE****Parameter**

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Pointer to memory where the memory size information will be stored (type: EC\_T\_MEMREQ\_DESC).
- dwOutBufSize: [in] Size of the output buffer in bytes.
- pdwNumOutData: [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer.

**Return**

EC\_E\_NOERROR or error code

struct **EC\_T\_MEMREQ\_DESC**

**Public Members**

**EC\_T\_DWORD dwPdOutSize**  
 Size of the output process data image

**EC\_T\_DWORD dwPdInSize**  
 Size of the input process data image

### 6.3.25 emIoControl - 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 [emIoControl - 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 [emConfigureNetwork\(\)](#) but prior to registering any client. Every client that registers with the master ([emRegisterClient\(\)](#)) will get back the memory pointers to PDOOut/PDIn data registered within this call.

#### **emIoControl - EC\_IOCTL\_REGISTER\_PDMEMORYPROVIDER**

##### **Parameter**

- pbyInBuf: [in] Memory provider (EC\_T\_MEMPROV\_DESC)
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

##### **Return**

EC\_E\_NOERROR or error code

struct **EC\_T\_MEMPROV\_DESC**

##### **Public Members**

###### **EC\_T\_PVOID pvContext**

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

###### **EC\_T\_PBYTE pbyPDOOutData**

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 [EC\\_T\\_MEMPROV\\_DESC.pfPDOOutDataReadRequest](#) is called

###### **EC\_T\_DWORD dwPDOOutDataLength**

Length of the output process data buffer

###### **EC\_T\_PBYTE pbyPDIInData**

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 [EC\\_T\\_MEMPROV\\_DESC.pfPDIInDataWriteRequest](#) is called

###### **EC\_T\_DWORD dwPDIInDataLength**

Length of the input process data buffer

###### **EC\_T\_PFMEMREQ pfPDOOutDataReadRequest**

This function will be called cyclically within the process data transfer cycle prior to read data from the output process data buffer. If EC\_NULL is set, the fixed buffer [EC\\_T\\_MEMPROV\\_DESC.pbyPDOOutData](#) is used.

***EC\_T\_PFMEMREL* *pfPDOOutDataReadRelease***

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

***EC\_T\_PFMEMREQ* *pfPDOOutDataWriteRequest***

This function will be called cyclically within the process data transfer cycle prior to write new data into the output process data buffer. If EC\_NULL is set, the fixed buffer *EC\_T\_MEMPROV\_DESC.pbyPDOOutData* is used.

***EC\_T\_PFMEMREL* *pfPDOOutDataWriteRelease***

This function will be called cyclically within the process data transfer cycle after all data were written into the output process data buffer.

***EC\_T\_PFMEMREQ* *pfPDInDataWriteRequest***

This function will be called cyclically within the process data transfer cycle prior to write new data into the input process data buffer. If EC\_NULL is set, the fixed buffer *EC\_T\_MEMPROV\_DESC.pbyPDInData* is used.

***EC\_T\_PFMEMREL* *pfPDInDataWriteRelease***

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

***EC\_T\_PBYTE* *pbyMasterRedPDOOutData***

Pointer to the MasterRed output process data buffer (ACTIVE to INACTIVE)

***EC\_T\_DWORD* *dwMasterRedPDOOutDataLength***

Length of the MasterRed output process data buffer

***EC\_T\_PBYTE* *pbyMasterRedPDInData***

Pointer to the default input process data buffer (INACTIVE to ACTIVE)

***EC\_T\_DWORD* *dwMasterRedPDInDataLength***

Length of the input process data buffer

***EC\_T\_PFMEMREQ* *pfMasterRedPDOOutReadRequest***

This function will be called within the process data transfer cycle prior to read data.

***EC\_T\_PFMEMREL* *pfMasterRedPDOOutReadRelease***

This function will be called after all data was read from output process data buffer.

***EC\_T\_PFMEMREQ* *pfMasterRedPDOOutWriteRequest***

This function will be called within the process data transfer cycle prior to read data.

***EC\_T\_PFMEMREL* *pfMasterRedPDOOutWriteRelease***

This function will be called after all data was read from output process data buffer.

***EC\_T\_PFMEMREQ* *pfMasterRedPDInWriteRequest***

This function will be called within the process data transfer cycle prior to write data.

***EC\_T\_PFMEMREL* *pfMasterRedPDInWriteRelease***

This function will be called after all data was written to input process data buffer.

***EC\_T\_PFMEMREQ* *pfMasterRedPDInReadRequest***

This function will be called within the process data transfer cycle prior to write data.

***EC\_T\_PFMEMREL* **pfMasterRedPDIInReadRelease****

This function will be called after all data was written to input process data buffer.

```
typedef EC_T_VOID (*EC_T_PFMEMREQ)(EC_T_PVOID pvContext, EC_T_DWORD dwTaskId,
EC_T_PBYTE *ppbyPDData)
```

**Parameters**

- **pvContext** – [in] Arbitrarily application-defined parameter passed to callback
- **dwTaskId** – [in] Task ID of cyclic data transfer (ENI: Cyclic/TaskId). If TASKID\_COMPLETE\_PD is given, the function must return a complete output process data buffer which contains valid data for all cyclic tasks.
- **ppbyPDData** – [out] Pointer to the process data buffer to be used. If set to EC\_NULL, the corresponding fixed buffer from [\*\*EC\\_T\\_MEMPROV\\_DESC\*\*](#) is used. The provided buffer size must correspond to the caller context.

```
typedef EC_T_VOID (*EC_T_PFMEMREL)(EC_T_PVOID pvContext, EC_T_DWORD dwTaskId)
```

**Parameters**

- **pvContext** – [in] Arbitrarily application-defined parameter passed to callback
- **dwTaskId** – [in] Task ID of cyclic data transfer (ENI: Cyclic/TaskId)

**See also:**

- [\*emIoControl - EC\\_IOCTL\\_GET\\_PDMEMORYSIZE\*](#)
- [\*emConfigureNetwork \(\)\*](#)
- [\*emRegisterClient \(\)\*](#)
- Feature Pack “Master Redundancy”

### **6.3.26 emIoControl - EC\_IOCTL\_REGISTER\_CYCFRAME\_RX\_CB**

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

**emIoControl - EC\_IOCTL\_REGISTER\_CYCFRAME\_RX\_CB****Parameter**

- **pbyInBuf**: [in] Cyclic frame received callback descriptor (EC\_T\_CYCFRAME\_RX\_CBDESC)
- **dwInBufSize**: [in] Size of the input buffer provided at pbyInBuf in bytes.
- **pbyOutBuf**: [out] Should be set to EC\_NULL
- **dwOutBufSize**: [in] Should be set to 0
- **pdwNumOutData**: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

```
struct EC_T_CYCFRAME_RX_CBDESC
```

## Public Members

### `EC_T_VOID *pCallbackContext`

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

### `EC_PF_CYCFRAME_RECV pfnCallback`

[in] This function will be called after the cyclic frame is received, if there is more than one cyclic frame after the last frame. The application has to assure that these functions will not block.

typedef EC\_T\_VOID (\*`EC_PF_CYCFRAME_RECV`)(EC\_T\_DWORD dwTaskId, EC\_T\_VOID \*pvContext)

#### Parameters

- `dwTaskId` – [in] Task ID of the received cyclic frame (ENI: Cyclic/TaskId)
- `pvContext` – [in] Arbitrarily application-defined parameter passed to callback

## 6.3.27 emIoControl - `EC_IOCTL_ISLINK_CONNECTED`

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

### `emIoControl - EC_IOCTL_ISLINK_CONNECTED`

#### Parameter

- `pbyInBuf`: [in] Should be set to EC\_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to 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.

#### Return

`EC_E_NOERROR` or error code

With Redundancy support enabled, EC\_FALSE is only set if main and redundancy link are down.

## 6.3.28 emIoControl - `EC_IOCTL_GET_LINKLAYER_MODE`

This call allows the application to determine whether the Real-time Ethernet Driver is currently running in polling or in interrupt mode.

### `emIoControl - EC_IOCTL_GET_LINKLAYER_MODE`

#### Parameter

- `pbyInBuf`: [in] Should be set to EC\_NULL
- `dwInBufSize`: [in] Should be set to 0
- `pbyOutBuf`: [out] Pointer to 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.

**Return**

EC\_E\_NOERROR or error code

struct **EC\_T\_LINKLAYER\_MODE\_DESC****Public Members*****EC\_T\_LINKMODE eLinkMode***

[out] Operation mode of main interface

***EC\_T\_LINKMODE eLinkModeRed***

[out] Operation mode of redundancy interface

### 6.3.29 emIoControl - EC\_IOCTL\_GET\_CYCLIC\_CONFIG\_INFO

Determine cyclic configuration details from ENI configuration file. It can be called only after calling ecatConfigureNetwork() or emConfigureNetwork()

#### emIoControl - EC\_IOCTL\_GET\_CYCLIC\_CONFIG\_INFO

**Parameter**

- pbyInBuf: [in] Pointer to dwCycEntryIndex: cyclic entry index for which to get information
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Pointer to EC\_T\_CYC\_CONFIG\_DESC data type
- dwOutBufSize: [in] Size of the output buffer provided at pbyOutBuf in bytes.
- pdwNumOutData: [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer.

**Return**

EC\_E\_NOERROR or error code

struct **EC\_T\_CYC\_CONFIG\_DESC****Public Members*****EC\_T\_DWORD dwNumCycEntries***

[out] Total number of cyclic entries

***EC\_T\_DWORD dwTaskId***

[out] Task ID of selected cyclic entry (ENI: Cyclic/TaskId)

***EC\_T\_DWORD dwPriority***

[out] Priority of selected cyclic entry

***EC\_T\_DWORD dwCycleTime***

[out] Cycle time of selected cyclic entry

### 6.3.30 emIoControl - EC\_IOCTL\_IS\_SLAVETOSLAVE\_COMM\_CONFIGURED

Determine if any slave to slave communication is configured.

#### emIoControl - EC\_IOCTL\_IS\_SLAVETOSLAVE\_COMM\_CONFIGURED

##### Parameter

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Pointer to EC\_T\_DWORD. If value is EC\_TRUE slave to slave communication is configured, if EC\_FALSE it is not.
- dwOutBufSize: [in] Size of the output buffer in bytes.
- pdwNumOutData: [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer.

##### Return

EC\_E\_NOERROR or error code

### 6.3.31 emIoControl - EC\_LINKIOCTL...

The generic control interface provides access to the main network adapter when adding EC\_IOCTL\_LINKLAYER\_MAIN to the EC\_LINKIOCTL parameter at dwCode.

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

### 6.3.32 emIoControl - EC\_LINKIOCTL\_GET\_ETHERNET\_ADDRESS

Provides MAC addresses of main or red line.

#### emIoControl - EC\_LINKIOCTL\_GET\_ETHERNET\_ADDRESS

##### Parameter

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Pointer to MAC address buffer (6 bytes).
- dwOutBufSize: [in] Size of the output buffer in bytes (at least 6).
- pdwNumOutData: [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer.

##### Return

EC\_E\_NOERROR or error code

### 6.3.33 emIoControl - EC\_LINKIOCTL\_GET\_SPEED

#### emIoControl - EC\_LINKIOCTL\_GET\_SPEED

##### Parameter

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Pointer to EC\_T\_DWORD. Set by Real-time Ethernet Driver to 10/100/1000.
- dwOutBufSize: [in] Size of the output buffer in bytes.
- pdwNumOutData: [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer.

##### Return

EC\_E\_NOERROR or error code

### 6.3.34 emIoControl - EC\_LINKIOCTL\_GET\_PCI\_INFO

Get current network adapter's PCI information

#### emIoControl - EC\_LINKIOCTL\_GET\_PCI\_INFO

##### Parameter

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Pointer to EC\_T\_PCI\_INFO buffer
- dwOutBufSize: [in] Size of the output buffer in bytes. Must be at least the size of EC\_T\_PCI\_INFO
- pdwNumOutData: [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer

##### Return

EC\_E\_NOERROR or error code

struct **EC\_T\_PCI\_INFO**

##### Public Members

**EC\_T\_PCI\_INFO\_LOCATION Location**  
PCI location (bus, device, function)

**EC\_T\_PCI\_INFO\_IDENIFICATION Ident**  
PCI identification (vendor id, device id)

**EC\_T\_DWORD dwIoBarCnt**  
I/O bar count

**EC\_T\_PCI\_INFO\_IOBAR aIoBar**  
PCI I/O bars info

**EC\_T\_DWORD dwMemBarCnt**  
memory bar count

**EC\_T\_PCI\_INFO\_MEMBAR aMemBar**  
PCI Memory bars info

**EC\_T\_DWORD dwInterruptCnt**  
IRQ count

**EC\_T\_PCI\_INFO\_INTERRUPT aInterrupt**  
PCI IRQ info

**EC\_T\_DWORD adwReserved[16]**  
reserved for future use

struct **EC\_T\_PCI\_INFO\_LOCATION**

### Public Members

**EC\_T\_DWORD dwDomainNumber**  
Domain

**EC\_T\_DWORD dwBusNumber**  
Bus number

**EC\_T\_DWORD dwDeviceNumber**  
Device number

**EC\_T\_DWORD dwFunctionNumber**  
Function number

struct **EC\_T\_PCI\_INFO\_IDENIFICATION**

### Public Members

**EC\_T\_DWORD dwVendorId**  
Vendor ID

**EC\_T\_DWORD dwDeviceId**  
Device ID

**EC\_T\_DWORD dwReserved**  
reserved for future use

struct **EC\_T\_PCI\_INFO\_IOPAR**

**Public Members**

**EC\_T\_UINT64 `qwBaseAddress`**  
I/O bar base address

**EC\_T\_DWORD `dwLen`**  
bar length

**EC\_T\_DWORD `dwReserved`**  
reserved for future use

struct **EC\_T\_PCI\_INFO\_MEMBAR**

**Public Members**

**EC\_T\_UINT64 `qwBaseAddress`**  
Memory bar base address

**EC\_T\_DWORD `dwLen`**  
bar length

**EC\_T\_DWORD `dwReserved`**  
reserved for future use

struct **EC\_T\_PCI\_INFO\_INTERRUPT**

**Public Members**

**EC\_T\_DWORD `dwIrq`**  
IRQ number

**EC\_T\_CPUSET `CpuAffinity`**  
reserved for future use

**EC\_T\_DWORD `adwReserved[2]`**  
reserved for future use

**6.3.35 emIoControl - EC\_IOCTL\_SET\_CYCFRAME\_LAYOUT**

Set the cyclic frames layout.

**emIoControl - EC\_IOCTL\_SET\_CYCFRAME\_LAYOUT****Parameter**

- **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:** [out] Should be set to **EC\_NULL**
- **dwOutBufSize:** [in] Should be set to 0

- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

enum **EC\_T\_CYCFRAME\_LAYOUT***Values:*enumerator **eCycFrameLayout\_STANDARD**

Layout according ENI with command add/reordering, no relationship to PD

enumerator **eCycFrameLayout\_DYNAMIC**

Layout is dynamically modified to send as less as possible cyclic frames and commands

enumerator **eCycFrameLayout\_FIXED**

Layout strictly match ENI, frame buffers and PD area overlapped

enumerator **eCycFrameLayout\_IN\_DMA**

Layout strictly match ENI, frame buffers and PD area overlapped, frame buffers in DMA

### **6.3.36 emIoControl - EC\_IOCTL\_SET\_MASTER\_DEFAULT\_TIMEOUTS**

Set master default timeouts.

#### **emIoControl - EC\_IOCTL\_SET\_MASTER\_DEFAULT\_TIMEOUTS**

**Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_MASTERDEFAULTTIMEOUTS\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

struct **EC\_T\_MASTERDEFAULTTIMEOUTS\_DESC****Public Members****EC\_T\_DWORD dwMasterStateChange**

Default state change timeout [ms], applied if emSetMasterState called with EC\_NOWAIT ( default: 20 sec )

**EC\_T\_DWORD dwInitCmdRetry**

Timeout [ms] between retry sending an init-command ( default: 10 ms )

**EC\_T\_DWORD dwMbxCmd**

Timeout [ms] between retry sending an mailbox command ( default: 10000 ms )

**EC\_T\_DWORD dwMbxPolling**

Mailbox polling cycle [ms] ( default: 10 ms )

**EC\_T\_DWORD dwDcmInSync**

Timeout [ms] to wait for DCM InSync in state change PREOP to SAFEOP ( default: infinite )

**EC\_T\_WORD wInitCmd**

Timeout [ms] to InitCmds if not specified in ENI ( default: 1100 ms )

**EC\_T\_DWORD dwSlaveIdentification**

Timeout [ms] to wait for the reading of the slave identification ( default: 5000 )

**EC\_T\_DWORD dwGenerateEni**

Timeout [ms] to wait for eCnfType\_GenPreopENI, eCnfType\_GenOpENI, eCnfType\_Gen... ( default: 10 ms )

**EC\_T\_DWORD dwBootMbxPolling**

Mailbox polling cycle [ms] for Boot ( default: 10 ms )

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

**See also:**

- *emSetMasterState()*

### **6.3.37 emIoControl - EC\_IOCTL\_SET\_COPYINFO\_IN\_SEDCYCFRAMES**

Set copy info processed in either SendCycFrames or in ProcessAllRxFrames.

#### **emIoControl - EC\_IOCTL\_SET\_COPYINFO\_IN\_SEDCYCFRAMES**

**Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_BOOL. EC\_TRUE: SendCycFrames, EC\_FALSE: ProcessAll-RxFrames
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

Default: Set by ProcessAllRxFrames.

### **6.3.38 emIoControl - EC\_IOCTL\_SET\_BUS\_CYCLE\_TIME**

Set bus cycle time in usec master parameter without calling *emInitMaster()* again.

#### **emIoControl - EC\_IOCTL\_SET\_BUS\_CYCLE\_TIME**

**Parameter**

- 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: [out] Should be set to EC\_NULL

- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

Implicitly recalculates Order Timeout and *EC\_T\_INIT\_MASTER\_PARMS::dwEcatCmdTimeout*.

**6.3.39 emIoControl - EC\_IOCTL\_ADDITIONAL\_VARIABLES\_FOR\_SPECIFIC\_DATA\_TYPES**

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

**emIoControl - EC\_IOCTL\_ADDITIONAL\_VARIABLES\_FOR\_SPECIFIC\_DATA\_TYPES****Parameter**

- pbyInBuf: [in] Pointer to value of EC\_T\_BOOL. EC\_TRUE: enable, EC\_FALSE: disable.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

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

**6.3.40 emIoControl - EC\_IOCTL\_SLV\_ALIAS\_ENABLE**

Enables slave alias addressing for all slaves.

**emIoControl - EC\_IOCTL\_SLV\_ALIAS\_ENABLE****Parameter**

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

---

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

---

### 6.3.41 emIoControl - EC\_IOCTL\_SET\_IGNORE\_INPUTS\_ON\_WKC\_ERROR

Set ignore inputs on WKC error

#### **emIoControl - EC\_IOCTL\_SET\_IGNORE\_INPUTS\_ON\_WKC\_ERROR**

##### **Parameter**

- pbyInBuf: [in] Pointer to value of EC\_T\_BOOL. EC\_TRUE: inputs are ignored on WKC error.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

##### **Return**

EC\_E\_NOERROR or error code

Calling this IOCTL with EC\_TRUE as parameter will ignore the inputs data of cyclic commands on WKC error. The default behavior will copy the input data if WKC is non zero and below the expected value. If WKC is not matching the expected value a notification [emNotify - EC\\_NOTIFY\\_CYCCMD\\_WKC\\_ERROR](#) is generated and the application must consider this status for the current cycle.

### 6.3.42 emIoControl - EC\_IOCTL\_SET\_ZERO\_INPUTS\_ON\_WKC\_ERROR

Set zero inputs on WKC error

#### **emIoControl - EC\_IOCTL\_SET\_ZERO\_INPUTS\_ON\_WKC\_ERROR**

##### **Parameter**

- pbyInBuf: [in] Pointer to value of EC\_T\_BOOL. EC\_TRUE: inputs are set to zero on WKC error.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

##### **Return**

EC\_E\_NOERROR or error code

Calling this IOCTL with EC\_TRUE as parameter will set the inputs data of cyclic commands to zero on WKC error. The default behavior will copy the input data if WKC is non zero and below the expected value. If WKC is not matching the expected value a notification [emNotify - EC\\_NOTIFY\\_CYCCMD\\_WKC\\_ERROR](#) is generated and the application must consider this status for the current cycle.

### 6.3.43 emIoControl - EC\_IOCTL\_SET\_ZERO\_INPUTS\_ON\_WKC\_ZERO

Set zero inputs on WKC is zero

#### emIoControl - EC\_IOCTL\_SET\_ZERO\_INPUTS\_ON\_WKC\_ZERO

##### Parameter

- pbyInBuf: [in] Pointer to value of EC\_T\_BOOL. EC\_TRUE: inputs are set to zero on WKC is zero.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

##### Return

EC\_E\_NOERROR or error code

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

### 6.3.44 emIoControl - EC\_IOCTL\_SET\_GENENI\_ASSIGN\_EEPROM\_BACK\_TO\_EM

Enable or disable creation of “assign EEPROM back to EM” InitCmd if ENI generated based on bus-scan result.

#### emIoControl - EC\_IOCTL\_SET\_GENENI\_ASSIGN\_EEPROM\_BACK\_TO\_EM

##### Parameter

- pbyInBuf: [in] Pointer to value of EC\_T\_BOOL. EC\_TRUE: generate InitCmd.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

##### Return

EC\_E\_NOERROR or error code

The ENI’s “assign EEPROM back to EM” InitCmd depends on the attribute “AssignToPdi” of the EEPROM tag in the slave’s description within the ESI file. Because this attribute is not reflected in the SII in the slave’s EEPROM, the Master cannot know its value and inserts for legacy reasons the InitCmd if not disabled using this IOCTL.

### 6.3.45 emIoControl - EC\_IOCTL\_SET\_EOE\_DEFERRED\_SWITCHING\_ENABLED

Enable or disable deferred EoE switching

#### emIoControl - EC\_IOCTL\_SET\_EOE\_DEFERRED\_SWITCHING\_ENABLED

##### Parameter

- pbyInBuf: [in] Pointer to value of EC\_T\_BOOL. EC\_TRUE: Deferred EoE switching enabled.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.

- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

Enabling deferred EoE switching reduces the CPU load of JOB\_ProcessAllRxFrames in case of EoE communication. `eUsrJob_SwitchEoeFrames` has to be called explicitly to switch the received EoE frames between the EoE slaves and EoE end point(s).

**6.3.46 emIoControl - EC\_IOCTL\_SET\_MAILBOX\_POLLING\_CYCLES**

This call changes the mailbox polling cycles.

**emIoControl - EC\_IOCTL\_SET\_MAILBOX\_POLLING\_CYCLES****Parameter**

- pbyInBuf: [in] Pointer to struct EC\_T\_SET\_MAILBOX\_POLLING\_CYCLES\_DESC
- dwInBufSize: [in] Size of the input buffer in bytes. E.g. sizeof(EC\_T\_SET\_MAILBOX\_POLLING\_CYCLES\_DESC)
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

```
struct EC_T_SET_MAILBOX_POLLING_CYCLES_DESC
```

**Public Members**

**EC\_T\_DWORD dwSlaveId**  
[in] Slave Id

**EC\_T\_WORD wCycles**  
[in] Number of cycles between polling [ms]

**6.3.47 emIoControl - EC\_IOCTL\_SET\_MASTER\_MAX\_STATE**

This call sets maximal master state. `emSetMasterState()` returns with `EC_E_INVALIDSTATE` if the requested master state exceeds the maximal master state.

**emIoControl - EC\_IOCTL\_SET\_MASTER\_MAX\_STATE****Parameter**

- pbyInBuf: [in] Pointer to value of EC\_T\_STATE
- dwInBufSize: [in] Size of the input buffer in bytes. E.g. sizeof(EC\_T\_STATE)
- pbyOutBuf: [out] Should be set to EC\_NULL

- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

**See also:**enum *EC\_T\_STATE***6.3.48 emIoControl - EC\_IOCTL\_ACTIVATE\_VOE\_RECV\_FIFO**

This call activates the VoE receive FIFO and sets its size.

**emIoControl - EC\_IOCTL\_ACTIVATE\_VOE\_RECV\_FIFO****Parameter**

- pbyInBuf: [in] Pointer to value of EC\_T\_WORD, size of the FIFO, use 0 to set it to the original size.
- dwInBufSize: [in] Size of the input buffer in bytes. E.g. sizeof(EC\_T\_WORD)
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

**6.3.49 emIoControl - EC\_IOCTL\_SET\_GEN\_ENI\_PARM**

This call changes the behavior when the configuration of the EtherCAT network is generated according to a bus scan result of *emConfigureNetwork()* with the parameter *EC\_T\_CNF\_TYPE::eCnfType\_GenPreopENI* or *EC\_T\_CNF\_TYPE::eCnfType\_GenOpENI*. In that case, default settings are taken to set e.g. the name of the EtherCAT slave device. The next table gives an overview about the possible parameters to be changed.

Identifier	Description
EC_GEN_ENI_PARM_ID_SLAVE_PREFIX	The prefix of the EtherCAT slave device and also the names of its variables with the given value. By default the prefix is 'Slave'.

**emIoControl - EC\_IOCTL\_SET\_GEN\_ENI\_PARM****Parameter**

- pbyInBuf: [in] Pointer to struct EC\_T\_SET\_GEN\_ENI\_PARM
- dwInBufSize: [in] Size of the input buffer in bytes. E.g. sizeof(EC\_T\_SET\_GEN\_ENI\_PARM)
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

---

```
struct EC_T_SET_GEN_ENI_PARM
```

#### Public Members

**EC\_T\_DWORD dwParmId**

ID of parameter to be set (EC\_GEN\_ENI\_PARM\_ID\_...)

**EC\_T\_GEN\_ENI\_PARM GenEniParm**

Value of parameter to be set

```
union EC_T_GEN_ENI_PARM
```

#### Public Members

**EC\_T\_CHAR szSlaveNamePrefix[MAX\_SHORT\_STRLEN + 1]**

**EC\_T\_BOOL bIgnoreScanBusError**

### 6.3.50 emIoControl - EC\_IOCTL\_REALLOC\_MBX\_QUEUE

This call realloc the number of queues of the different mailbox protocols.

#### emIoControl - EC\_IOCTL\_REALLOC\_MBX\_QUEUE

##### Parameter

- pbyInBuf: [in] Pointer to struct EC\_T\_REALLOC\_MBX\_QUEUE\_DESC
- dwInBufSize: [in] Size of the input buffer in bytes. E.g. sizeof(EC\_T\_REALLOC\_MBX\_QUEUE\_DESC)
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

##### Return

EC\_E\_NOERROR or error code

```
struct EC_T_REALLOC_MBX_QUEUE_DESC
```

#### Public Members

**EC\_T\_WORD wSlaveFixedAddress**

Slave fixed address, 0 for all slaves

**EC\_T\_WORD wMbxBProtocols**

Combination of Mbx protocols EC\_MBX\_PROTOCOL\_COE...

**EC\_T\_DWORD dwMaxMbxBferQueued**

Maximal number of transfers queued for the specified mailbox protocol of the specified slave

## 6.4 Process Data Access

### 6.4.1 emGetProcessData

```
static EC_T_DWORD ecatGetProcessData (
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwOffset,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwLength,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emGetProcessData (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwOffset,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout
)
```

Retrieve Process data synchronized.

If process data are required outside the cyclic master job task (which is calling ecatExecJob), direct access to the process data is not recommended as data consistency cannot be guaranteed. A call to this function will send a data read request to the master stack and then check every millisecond whether new data are provided. The master stack will provide new data after calling ecatExecJob(eUsrJob\_MasterTimer) within the job task. This function is usually only called remotely (using the Remote API).

---

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

---

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC\_TRUE: read output data, EC\_FALSE: read input data.
- **dwOffset** – [in] Byte offset in Process data to read from.
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **dwTimeout** – [in] Timeout [ms]

#### Returns

*EC\_E\_NOERROR* or error code

#### emGetProcessData() Example

```
/* get Process data (synchronized with JobTask) */
EC_T_BYTE abyData[1] = { 0 };
dwRes = emGetProcessData(dwInstanceId, EC_FALSE, 0 /* byte offset */,
    abyData, 1 /* byte length */, 5000);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
    "Process Data: 0x%02X\n", abyData[0]));
```

## 6.4.2 emGetProcessDataBits

```
static EC_T_DWORD ecatGetProcessDataBits (
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwBitOffsetPd,
    EC_T_BYTE *pbyDataDst,
    EC_T_DWORD dwBitLengthDst,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emGetProcessDataBits (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwBitOffsetPd,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataBitLen,
    EC_T_DWORD dwTimeout
)
```

) Reads a specific number of bits from the process image to the given buffer with a bit offset (synchronized).

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

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC\_TRUE: read output data, EC\_FALSE: write input data.
- **dwBitOffsetPd** – [in] Bit offset in Process data image.
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataBitLen** – [in] Buffer length [bit]
- **dwTimeout** – [in] Timeout [ms]. The timeout value must not be set to EC\_NOWAIT.

### Returns

*EC\_E\_NOERROR* or error code

### See also:

*emGetProcessData()*

### emGetProcessDataBits() Example

```
/* get bits from Process Data Image (synchronized with JobTask) */
EC_T_BYTE abyData[1] = { 0 };
dwRes = emGetProcessDataBits(dwInstanceId, EC_FALSE, 0 /* bit offset */,
    abyData, 8 /* bit length */, 5000);
```

## 6.4.3 emSetProcessData

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

```
EC_T_DWORD emSetProcessData (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwOffset,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout
)
```

Write Process data synchronized.

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

---

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

---

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC\_TRUE: write output data, EC\_FALSE: write input data.
- **dwOffset** – [in] Byte offset in Process data to write to.
- **pbyData** – [in] Buffer containing transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **dwTimeout** – [in] Timeout [ms]

### Returns

*EC\_E\_NOERROR* or error code

### emSetProcessData() Example

```
/* write value 0x12 to Process Data Image (synchronized with JobTask) */
EC_T_BYTE abyData[1] = { 0x12 };
dwRes = emSetProcessData(dwInstanceId, EC_FALSE, 0 /* byte offset */,
    abyData, 1 /* byte length */, 5000);
```

## 6.4.4 emSetProcessDataBits

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

```
EC_T_DWORD emSetProcessDataBits (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwBitOffsetPd,
    EC_T_BYT *pbyData,
    EC_T_DWORD dwDataBitLen,
    EC_T_DWORD dwTimeout
)
```

) Writes a specific number of bits from a given buffer to the process image with a bit offset (synchronized).

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

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC\_TRUE: write output data, EC\_FALSE: write input data.
- **dwBitOffsetPd** – [in] Bit offset in Process data image.
- **pbyData** – [in] Buffer containing transferred data
- **dwDataBitLen** – [in] Buffer length [bit]
- **dwTimeout** – [in] Timeout [ms]. The timeout value must not be set to EC\_NOWAIT.

#### Returns

*EC\_E\_NOERROR* or error code

#### See also:

*emSetProcessData()*

#### emSetProcessDataBits() Example

```
/* write bits to Process Data Image (synchronized with JobTask) */
EC_T_BYT abyData[1] = { 0x12 };
dwRes = emSetProcessDataBits(dwInstanceId, EC_TRUE, 0 /* bit offset */,
    abyData, 8 /* bit length */, 5000);
```

### 6.4.5 emForceProcessDataBits

```
static EC_T_DWORD ecatForceProcessDataBits (
    EC_T_DWORD dwClientId,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwBitOffsetPd,
    EC_T_WORD wBitLength,
    EC_T_BYT *pbyData,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emForceProcessDataBits (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwClientId,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwBitOffsetPd,
    EC_T_WORD wDataBitLen,
    EC_T_BYT *pbyData,
    EC_T_DWORD dwTimeout
)
Force 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 terminated by calling the corresponding functions. This function may not be called from within the JobTask's context.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **bOutputData** – [in] EC\_TRUE: write output data, EC\_FALSE: write input data.
- **dwBitOffsetPd** – [in] Bit offset in Process data image
- **wDataBitLen** – [in] Buffer length [bit]
- **pbyData** – [in] Buffer containing transferred data
- **dwTimeout** – [in] Timeout [ms]. The timeout value must not be set to EC\_NOWAIT.

### Returns

*EC\_E\_NOERROR* or error code

### emForceProcessDataBits() Example

```
/* force specific number of bits from given buffer to process image with a bit-
 * offset */
EC_T_BYTE abyData[1] = { 1 };
dwRes = emForceProcessDataBits(dwInstanceId, dwClientId,
    EC_FALSE, 0 /* bit offset */, 1 /* bit length */, abyData, 5000 /* timeout */);
```

### See also:

- *emSetProcessData()*
- *emReleaseProcessDataBits()*
- *emReleaseAllProcessDataBits()*

### 6.4.6 emReleaseProcessDataBits

```
static EC_T_DWORD ecatReleaseProcessDataBits (
    EC_T_DWORD dwClientId,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwBitOffsetPd,
    EC_T_WORD wBitLength,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emReleaseProcessDataBits (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwClientId,
    EC_T_BOOL bOutputData,
    EC_T_DWORD dwBitOffsetPd,
    EC_T_WORD wBitLength,
    EC_T_DWORD dwTimeout
)
    Release previously forced process data.
```

- Forced output: Value set by application become valid again. Because forced process data bits are written 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.
- Forced input: Value read from the slaves become valid again.

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

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **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 [ms]. The timeout value must not be set to EC\_NOWAIT.

#### Returns

*EC\_E\_NOERROR* or error code

#### emReleaseProcessDataBits() Example

```
/* release previously forced process data */
dwRes = emReleaseProcessDataBits(dwInstanceId, dwClientId,
    EC_FALSE, 0 /* bit offset */, 1 /* bit length */, 5000 /* timeout */);
```

#### See also:

- *emSetProcessData()*
- *emForceProcessDataBits()*

### 6.4.7 emReleaseAllProcessDataBits

```
static EC_T_DWORD ecatReleaseAllProcessDataBits (
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emReleaseAllProcessDataBits (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTimeout
)
    Release all previously forced process data for a dedicated client.
```

- Forced output: Value set by application become valid again. Because forced process data bits are written 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.
- Forced input: Value read from the slaves become valid again.

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

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTimeout** – [in] Timeout [ms]. The timeout value must not be set to EC\_NOWAIT.

**Returns***EC\_E\_NOERROR* or error code**emReleaseAllProcessDataBits() Example**

```
/* release all previously forced process data of client */
dwRes = emReleaseAllProcessDataBits(dwInstanceId, dwClientId, 5000 /* timeout */);
```

**See also:**

- [emSetProcessData\(\)](#)
- [emForceProcessDataBits\(\)](#)

**6.4.8 emGetProcessImageInputPtr**

static EC\_T\_BYTE \***ecatGetProcessImageInputPtr**(EC\_T\_VOID)  
**EC\_T\_BYTE \*emGetProcessImageInputPtr**(EC\_T\_DWORD dwInstanceId)  
 Gets the process data input image pointer.

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

Process data input image pointer

**emGetProcessImageInputPtr() Example**

```
EC_T_BYTE* pbyProcessImageInput = emGetProcessImageInputPtr(dwInstanceId);
```

**See also:**

- [emConfigureNetwork\(\)](#)
- [emIoControl - EC\\_IOCTL\\_GET\\_PDMEMORYSIZE](#)
- [emIoControl - EC\\_IOCTL\\_REGISTER\\_PDMEMORYPROVIDER](#)
- [emExecJob\(\)](#) (`eUsrJob_ProcessAllRxFrames`) in case of Polling Mode

**6.4.9 emGetProcessImageOutputPtr**

static EC\_T\_BYTE \***ecatGetProcessImageOutputPtr**(EC\_T\_VOID)  
**EC\_T\_BYTE \*emGetProcessImageOutputPtr**(EC\_T\_DWORD dwInstanceId)  
 Gets the process data output image pointer.

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

Process data output image pointer

### **emGetProcessImageOutputPtr() Example**

```
EC_T_BYTE* pbyProcessImageOutput = emGetProcessImageOutputPtr(dwInstanceId);
```

**See also:**

- *emConfigureNetwork()*
- *emIoControl - EC\_IOCTL\_GET\_PDMEMORYSIZE*
- *emIoControl - EC\_IOCTL\_REGISTER\_PDMEMORYPROVIDER*
- *emExecJob()* (`eUsrJob_SendAllCycFrames`)

### **6.4.10 emGetDiagnosisImagePtr**

static EC\_T\_BYTE \***ecatGetDiagnosisImagePtr** (EC\_T\_VOID)

**EC\_T\_BYTE \*emGetDiagnosisImagePtr** (EC\_T\_DWORD dwInstanceId)

Gets the diagnosis image pointer.

**Parameters**

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

**Returns**

Diagnosis image pointer

### **emGetDiagnosisImagePtr() Example**

```
EC_T_BYTE* pbyProcessImageOutput = emGetDiagnosisImagePtr(dwInstanceId);
```

### **6.4.11 emGetDiagnosisImageSize**

static EC\_T\_DWORD **ecatGetDiagnosisImageSize** (EC\_T\_VOID)

**EC\_T\_DWORD emGetDiagnosisImageSize** (EC\_T\_DWORD dwInstanceId)

Gets the diagnosis image size.

**Parameters**

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

**Returns**

Diagnosis image size

### **6.4.12 emGetSlaveInpVarInfoNumOf**

```
static EC_T_DWORD ecatGetSlaveInpVarInfoNumOf (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD *pwSlaveInpVarInfoNumOf
)
```

```
EC_T_DWORD emGetSlaveInpVarInfoNumOf (
```

    EC\_T\_DWORD dwInstanceId,

    EC\_T\_BOOL bFixedAddressing,

    EC\_T\_WORD wSlaveAddress,

    EC\_T\_WORD \*pwSlaveInpVarInfoNumOf

)

Gets the number of input variables of a specific slave.

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pwSlaveInpVarInfoNumOf** – [out] Number of found process variable entries

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or the output pointer is EC\_NULL
- *EC\_E\_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found

**See also:**

- [emGetSlaveInpVarInfo\(\)](#)
- [emGetSlaveInpVarInfoEx\(\)](#)

**emGetSlaveInpVarInfoNumOf() Example**

```
/* get number of input variables of specific slave */
EC_T_WORD numVars = 0;
dwRes = emGetSlaveInpVarInfoNumOf(dwInstanceId, EC_TRUE, 1004, &numVars);
```

**6.4.13 emGetSlaveInpVarInfo**

```
static 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
)
EC_T_DWORD emGetSlaveInpVarInfo (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wNumOfVarsToRead,
    EC_T_PROCESS_VAR_INFO *pSlaveProcVarInfoEntries,
    EC_T_WORD *pwReadEntries
)
```

Gets the process variable information entries of an specific slave.

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing

- **wNumOfVarsToRead** – [in] Number process variable entries that have been stored in pSlaveProcVarInfoEntries
- **pSlaveProcVarInfoEntries** – [out] The read process variable information entries
- **pwReadEntries** – [out] The number of read process variable information entries

**Returns***EC\_E\_NOERROR* or error codestruct **EC\_T\_PROCESS\_VAR\_INFO****Public Members****EC\_T\_CHAR szName[MAX\_PROCESS\_VAR\_NAME\_LEN]**

[out] Name of the found process variable

**EC\_T\_WORD wDataType**

[out] Data type of the found process variable (according to ETG.1000, section 5). See also EcCommon.h, DEFTYPE\_BOOLEAN

**EC\_T\_WORD wFixedAddr**

[out] Station address of the slave that is owner of this variable

**EC\_T\_INT nBitSize**

[out] Size in bit of the found process variable

**EC\_T\_INT nBitOffs**

[out] Bit offset in the process data image

**EC\_T\_BOOL bIsInputData**

[out] Determines whether the found process variable is an input variable or an output variable

**MAX\_PROCESS\_VAR\_NAME\_LEN**

Maximum length of a process variable name: 71 characters

**emGetSlaveInpVarInfo() Example**

```
/* get process variable information entries of specific slave */
EC_T_PROCESS_VAR_INFO aSlaveInpVarInfoNumOf[4];
EC_T_WORD wReadEntries = 0;
EC_T_WORD numOfVars = 0;
emGetSlaveInpVarInfoNumOf(dwInstanceId, EC_TRUE, 1004, &numOfVars);
dwRes = emGetSlaveInpVarInfo(dwInstanceId, EC_TRUE,
    1001, numOfVars, aSlaveInpVarInfoNumOf, &wReadEntries);
```

### 6.4.14 emGetSlaveInpVarInfoEx

```
static EC_T_DWORD ecatGetSlaveInpVarInfoEx (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wNumOfVarsToRead,
    EC_T_PROCESS_VAR_INFO_EX *pSlaveProcVarInfoEntries,
    EC_T_WORD *pwReadEntries
)
EC_T_DWORD emGetSlaveInpVarInfoEx (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wNumOfVarsToRead,
    EC_T_PROCESS_VAR_INFO_EX *pSlaveProcVarInfoEntriesEx,
    EC_T_WORD *pwReadEntries
)
```

) Gets the input process variable extended information entries of a specific slave.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wNumOfVarsToRead** – [in] Number process variable entries that have been stored in pSlaveProcVarInfoEntries
- **pSlaveProcVarInfoEntriesEx** – [out] The read process variable extended information entries
- **pwReadEntries** – [out] The number of read process variable information entries

#### Returns

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master isn't initialized
- **EC\_E\_INVALIDPARAM** if dwInstanceID is out of range or the output pointer is EC\_NULL
- **EC\_E\_NOTFOUND** if no slave matching bFixedAddressing / wSlaveAddress can be found

struct **EC\_T\_PROCESS\_VAR\_INFO\_EX**

#### Public Members

**EC\_T\_CHAR szName[MAX\_PROCESS\_VAR\_NAME\_LEN\_EX]**  
 [out] Name of the found process variable

**EC\_T\_WORD wDataType**  
 [out] Data type of the found process variable (according to ETG.1000, section 5). See also EcCommon.h, DEFTYPE\_BOOLEAN

**EC\_T\_WORD wFixedAddr**  
 [out] Station address of the slave that is owner of this variable

**EC\_T\_INT nBitSize**

[out] Size in bit of the found process variable

**EC\_T\_INT nBitOffs**

[out] Bit offset in the process data image

**EC\_T\_BOOL bIsInputData**

[out] Determines whether the found process variable is an input variable or an output variable

**EC\_T\_WORD wIndex**

[out] Object index

**EC\_T\_WORD wSubIndex**

[out] Object sub index

**EC\_T\_WORD wPdoIndex**

[out] Index of PDO (process data object)

**EC\_T\_WORD wWkcStateDiagOffs**

[out] Bit offset in the diagnostic image (emGetDiagnosisImagePtr)

**EC\_T\_WORD wMasterSyncUnit**

[out] Master Sync Unit ID (ENI: Slave/ProcessData/RxPdo[1..4]@Su, Slave/ProcessData/TxPdo[1..4]@Su, comment at Cyclic/Frame/Cmd)

**EC\_T\_CYC\_COPY\_INFO CopyInfo**

[out] Copy Info if applied to the variable

**MAX\_PROCESS\_VAR\_NAME\_LEN\_EX**

Maximum length of a extended process variable name: 127 characters

### **emGetSlaveInpVarInfoEx() Example**

```
EC_T_PROCESS_VAR_INFO_EX aSlaveInpVarInfoNumOf[4];
EC_T_WORD wReadEntries = 0;
EC_T_WORD wNumOfVarsToRead = 4;
emGetSlaveInpVarInfoNumOf(dwInstanceId, EC_TRUE, 1004, &wNumOfVarsToRead);
/* get extended process variable information entries of specific slave */
dwRes = emGetSlaveInpVarInfoEx(dwInstanceId, EC_TRUE, 1001,
wNumOfVarsToRead, aSlaveInpVarInfoNumOf, &wReadEntries);
```

### **6.4.15 emGetSlaveOutpVarInfoNumOf**

```
static EC_T_DWORD ecatGetSlaveOutpVarInfoNumOf (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD *pwSlaveOutpVarInfoNumOf
)
EC_T_DWORD emGetSlaveOutpVarInfoNumOf (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD *pwSlaveOutpVarInfoNumOf
)
```

Gets the number of output variables of a specific slave.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pwSlaveOutpVarInfoNumOf** – [out] Number of found process variables

#### Returns

*EC\_E\_NOERROR* or error code

#### **emGetSlaveOutpVarInfoNumOf()** Example

```
/* get number of output variables of specific slave */
EC_T_WORD numOfVars = 0;
dwRes = emGetSlaveOutpVarInfoNumOf(dwInstanceId, EC_TRUE, &numOfVars);
```

#### See also:

- [emGetSlaveOutpVarInfo\(\)](#)
- [emGetSlaveOutpVarInfoEx\(\)](#)

### 6.4.16 emGetSlaveOutpVarInfo

```
static 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
)
EC_T_DWORD emGetSlaveOutpVarInfo (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wNumOfVarsToRead,
    EC_T_PROCESS_VAR_INFO *pSlaveProcVarInfoEntries,
    EC_T_WORD *pwReadEntries
)
```

Gets the output process variable information entries of a specific slave.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wNumOfVarsToRead** – [in] Number of found process variable entries
- **pSlaveProcVarInfoEntries** – [out] The read process variable information entries
- **pwReadEntries** – [out] The number of read process variable information entries

**Returns***EC\_E\_NOERROR* or error code**emGetSlaveOutpVarInfo() Example**

```
/* get general output process variable information entries of specific slave */
EC_T_PROCESS_VAR_INFO aoSlaveOutpVarInfo[4]; /* see emGetSlaveOutpVarInfoNumOf() */
EC_T_WORD wSlaveOutpVarInfoCnt = 0;
dwRes = emGetSlaveOutpVarInfo(dwInstanceId, EC_TRUE, 1002,
    4 /* see emGetSlaveOutpVarInfoNumOf() */, aoSlaveOutpVarInfo, &
    ↪wSlaveOutpVarInfoCnt);
```

**See also:***EC\_T\_PROCESS\_VAR\_INFO***6.4.17 emGetSlaveOutpVarInfoEx**

```
static EC_T_DWORD ecatGetSlaveOutpVarInfoEx (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wNumOfVarsToRead,
    EC_T_PROCESS_VAR_INFO_EX *pSlaveProcVarInfoEntries,
    EC_T_WORD *pwReadEntries
)
EC_T_DWORD emGetSlaveOutpVarInfoEx (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wNumOfVarsToRead,
    EC_T_PROCESS_VAR_INFO_EX *pSlaveProcVarInfoEntriesEx,
    EC_T_WORD *pwReadEntries
)
```

) Gets the output process variable extended information entries of a specific slave.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wNumOfVarsToRead** – [in] Number of process variable information entries
- **pSlaveProcVarInfoEntriesEx** – [out] The read process extended variable entries
- **pwReadEntries** – [out] The number of read process variable information entries

**Returns***EC\_E\_NOERROR* or error code

### **emGetSlaveOutpVarInfoEx() Example**

```
/* get extended output process variable information entries of specific slave */
EC_T_PROCESS_VAR_INFO_EX aoSlaveOutpVarInfo[4]; /* see emGetSlaveOutpVarInfoNumOf() */
/* */
EC_T_WORD wSlaveOutpVarInfoCnt = 0;
dwRes = emGetSlaveOutpVarInfoEx(dwInstanceId, EC_TRUE, 1002,
    4 /* see emGetSlaveOutpVarInfoNumOf() */, aoSlaveOutpVarInfo, &
    wSlaveOutpVarInfoCnt);
```

**See also:**

*EC\_T\_PROCESS\_VAR\_INFO\_EX*

### **6.4.18 emGetSlaveInpVarByObjectEx**

```
static EC_T_DWORD ecatGetSlaveInpVarByObjectEx (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wIndex,
    EC_T_WORD wSubIndex,
    EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)
EC_T_DWORD emGetSlaveInpVarByObjectEx (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wIndex,
    EC_T_WORD wSubIndex,
    EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)
```

Gets the input process variable extended information entry by object index, subindex of a specific slave.

#### **Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wIndex** – [in] Object Index
- **wSubIndex** – [in] Object SubIndex
- **pProcessVarInfoEntry** – [out] Process variable extended information entry

#### **Returns**

*EC\_E\_NOERROR* or error code

**emGetSlaveInpVarByObjectEx() Example**

```
/* get input process variable extended information entry by object index,
   subindex of specific slave */
EC_T_PROCESS_VAR_INFO_EX processVarInfoEntry;
OsMemset(&processVarInfoEntry, 0, sizeof(EC_T_PROCESS_VAR_INFO_EX));
dwRes = emGetSlaveInpVarByObjectEx(dwInstanceId, EC_TRUE,
    1004, 0x6000 /* Index */, 1 /* SubIndex */, &processVarInfoEntry);
```

**See also:***EC\_T\_PROCESS\_VAR\_INFO\_EX***6.4.19 emGetSlaveOutpVarByObjectEx**

```
static EC_T_DWORD ecatGetSlaveOutpVarByObjectEx (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wIndex,
    EC_T_WORD wSubIndex,
    EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)
EC_T_DWORD emGetSlaveOutpVarByObjectEx (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wIndex,
    EC_T_WORD wSubIndex,
    EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)
```

Gets the input process variable extended information entry by object index, subindex of a specific slave.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wIndex** – [in] Object Index
- **wSubIndex** – [in] Object SubIndex
- **pProcessVarInfoEntry** – [out] Process variable extended information entry

**Returns***EC\_E\_NOERROR* or error code

**emGetSlaveOutpVarByObjectEx() Example**

```
/* get output process variable extended information entry
   by object index, subindex of specific slave. */
EC_T_PROCESS_VAR_INFO_EX oSlaveOutpVarInfo;
OsMemset(&oSlaveOutpVarInfo, 0, sizeof(EC_T_PROCESS_VAR_INFO_EX));
dwRes = emGetSlaveOutpVarByObjectEx(dwInstanceId, EC_TRUE,
    1002, 0x3001 /* Index */, 1 /* SubIndex */, &oSlaveOutpVarInfo);
```

**See also:***EC\_T\_PROCESS\_VAR\_INFO\_EX***6.4.20 emFindInpVarByName**

```
static EC_T_DWORD ecatFindInpVarByName (
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO *pProcessVarInfoEntry
)
EC_T_DWORD emFindInpVarByName (
    EC_T_DWORD dwInstanceId,
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO *pProcessVarInfoEntry
)
```

Finds an input process variable information entry by the variable name.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szVariableName** – [in] Variable name
- **pProcessVarInfoEntry** – [out] Process variable information entry

**Returns***EC\_E\_NOERROR* or error code**emFindInpVarByName() Example**

```
/* get general information about Process Data variable by name */
EC_T_PROCESS_VAR_INFO oProcessVarInfoEntry;
OsMemset(&oProcessVarInfoEntry, 0, sizeof(EC_T_PROCESS_VAR_INFO));
dwRes = emFindInpVarByName(dwInstanceId, "Slave_1004 [EL1014].Channel 1.Input", &
    oProcessVarInfoEntry);
```

**See also:***EC\_T\_PROCESS\_VAR\_INFO*

### 6.4.21 emFindInpVarByNameEx

```
static EC_T_DWORD ecatFindInpVarByNameEx (
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)
EC_T_DWORD emFindInpVarByNameEx (
    EC_T_DWORD dwInstanceId,
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)
Finds an input process variable extended information entry by the variable name.
```

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szVariableName** – [in] Variable name
- **pProcessVarInfoEntry** – [out] Process variable extended information entry

#### Returns

*EC\_E\_NOERROR* or error code

### emFindInpVarByNameEx() Example

```
/* get extended information about Process Data variable by name */
EC_T_PROCESS_VAR_INFO_EX oProcessVarInfoEntry;
OsMemset(&oProcessVarInfoEntry, 0, sizeof(EC_T_PROCESS_VAR_INFO_EX));
dwRes = emFindInpVarByNameEx(dwInstanceId, "Slave_1004 [EL1014].Channel 1.Input", &
    ↪oProcessVarInfoEntry);
```

#### See also:

*EC\_T\_PROCESS\_VAR\_INFO\_EX*

### 6.4.22 emFindOutpVarByName

```
static EC_T_DWORD ecatFindOutpVarByName (
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO *pProcessVarInfoEntry
)
EC_T_DWORD emFindOutpVarByName (
    EC_T_DWORD dwInstanceId,
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO *pProcessVarInfoEntry
)
Finds an output process variable information entry by the variable name.
```

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szVariableName** – [in] Variable name
- **pProcessVarInfoEntry** – [out] Process variable information entry

#### Returns

*EC\_E\_NOERROR* or error code

**emFindOutpVarByName() Example**

```
/* get general information about Process Data variable by name */
EC_T_PROCESS_VAR_INFO oProcessVarInfoEntry;
OsMemset(&oProcessVarInfoEntry, 0, sizeof(EC_T_PROCESS_VAR_INFO));
dwRes = emFindOutpVarByName(dwInstanceId, "Slave_1002 [EL2004].channel 1.Output", &
    ↪oProcessVarInfoEntry);
```

**See also:***EC\_T\_PROCESS\_VAR\_INFO***6.4.23 emFindOutpVarByNameEx**

```
static EC_T_DWORD ecatFindOutpVarByNameEx (
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)
EC_T_DWORD emFindOutpVarByNameEx (
    EC_T_DWORD dwInstanceId,
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO_EX *pProcessVarInfoEntry
)
```

Finds an output process variable extended information entry by the variable name.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **szVariableName** – [in] Variable name
- **pProcessVarInfoEntry** – [out] Process variable extended information entry

**Returns***EC\_E\_NOERROR* or error code**emFindOutpVarByName() Example**

```
/* get extended information about Process Data variable by name */
EC_T_PROCESS_VAR_INFO_EX oProcessVarInfoEntry;
OsMemset(&oProcessVarInfoEntry, 0, sizeof(EC_T_PROCESS_VAR_INFO_EX));
dwRes = emFindOutpVarByNameEx(dwInstanceId, "Slave_1002 [EL2004].Channel 1.Output",
    ↪ &oProcessVarInfoEntry);
```

**See also:***EC\_T\_PROCESS\_VAR\_INFO\_EX*

### 6.4.24 emTraceDataConfig

```
static EC_T_DWORD ecatTraceDataConfig (EC_T_WORD wTraceDataSize)
```

```
EC_T_DWORD emTraceDataConfig (
    EC_T_DWORD dwInstanceID,
    EC_T_WORD wTraceDataSize
)
```

) Configures a trace data buffer and enables it for transmission.

Must be called after initialization and before configuration.

---

**Note:** If wTraceDataSize is too large, configuration will fail with return code *EC\_E\_XML\_CYCCMDS\_SIZEISMATCH*.

---

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **wTraceDataSize** – [in] Size of Trace Data in bytes

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range
- *EC\_E\_NOTSUPPORTED* if eCycFrameLayout\_FIXED is configured

### 6.4.25 emTraceDataGetInfo

```
static EC_T_DWORD ecatTraceDataGetInfo (EC_T_TRACE_DATA_INFO *pTraceDataInfo)
```

```
EC_T_DWORD emTraceDataGetInfo (
    EC_T_DWORD dwInstanceID,
    EC_T_TRACE_DATA_INFO *pTraceDataInfo
)
```

) Get information about the offset and size of trace data.

The trace data buffer is locate in *EC\_T\_TRACE\_DATA\_INFO.pbyData* at the byte offset *EC\_T\_TRACE\_DATA\_INFO.dwOffset*.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pTraceDataInfo** – [out] Information about trace data

#### Returns

*EC\_E\_NOERROR* or error code

```
struct EC_T_TRACE_DATA_INFO
```

## Public Members

**EC\_T\_BYTE \*pbyData**  
 [out] Process data output buffer, containing trace data

**EC\_T\_DWORD dwOffset**  
 [out] Trace data offset in bytes

**EC\_T\_WORD wSize**  
 [out] Trace data size in bytes

### emTraceDataGetInfo() Example

```
/* get trace data offset and size */
EC_T_TRACE_DATA_INFO traceDataInfo;
OsMemset(&traceDataInfo, 0, sizeof(EC_T_TRACE_DATA_INFO));
dwRes = emTraceDataGetInfo(dwInstanceId, &traceDataInfo);
```

## 6.4.26 EC\_COPYBITS

**EC\_COPYBITS** (pbyDst, nDstBitOffs, pbySrc, nSrcBitOffs, nBitSize)  
 Copies a block of bits from a source buffer to a destination buffer.

---

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

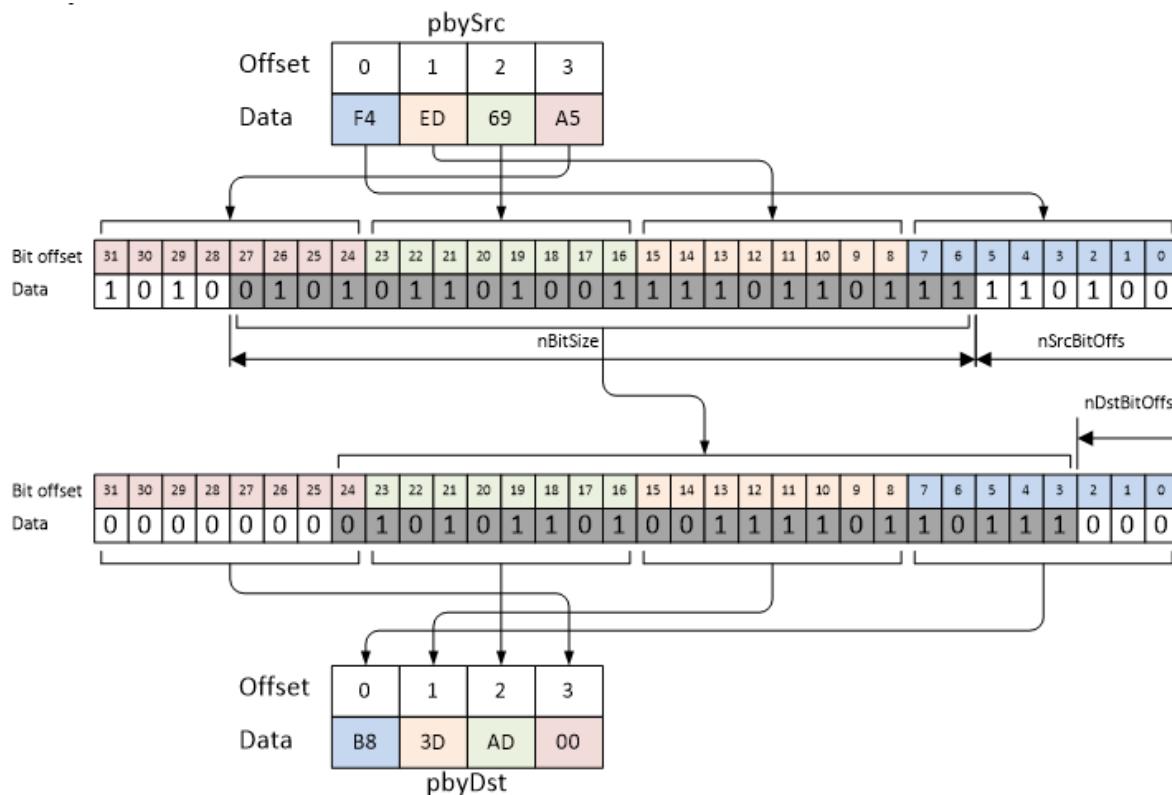
---

### Parameters

- **pbyDst** – [out] Destination buffer
- **nDstBitOffs** – [in] Bit offset within destination buffer
- **pbySrc** – [in] Source buffer
- **nSrcBitOffs** – [in] Bit offset within source buffer
- **nBitSize** – [in] Block size in bits

### See also:

- [EC\\_SETBITS](#)
- [EC\\_GETBITS](#)



```
EC_T_BYTE pbySrc[] = {0xF4, 0xED, 0x69, 0xA5};
EC_T_BYTE pbyDst[] = {0x00, 0x00, 0x00, 0x00};
EC_COPYBITS(pbyDst, 3, pbySrc, 6, 22);

/* pbyDst now contains 0xB8 0x3D 0xAD 0x00 */
```

## 6.4.27 EC\_COMPAREBITS

**EC\_COMPAREBITS** (pbyBuf1, nBitOffs1, pbyBuf2, nBitOffs2, nBitSize)  
Compares a block of bits of two buffers.

**Note:** The buffers are compared bitwise with the same semantic as `memcmp()`. The buffers are not checked for overrun.

### Parameters

- **pbyBuf1** – [in] Buffer 1
- **nBitOffs1** – [in] Bit offset within buffer 1
- **pbyBuf2** – [in] Buffer 2
- **nBitOffs2** – [in] Bit offset within buffer 2
- **nBitSize** – [in] Bit size to compare

### Returns

Integral value indicating the relationship between the content of the memory blocks:

- 0: The blocks of bits within the two buffers are equal

- <0: Binary value from block of bits in buffer 1 has a lower value than binary value from block of bits in buffer 2
- >0: Binary value from block of bits in buffer 1 has a greater value than binary value from block of bits in buffer 2

**See also:**

- [EC\\_COPYBITS](#)

```
EC_T_BYTE pbyBuf1[] = {0xB8, 0x3D, 0xAD, 0x00};
EC_T_BYTE pbyBuf2[] = {0xF4, 0xED, 0x69, 0xA5};
assert(0 == EC_COMPAREBITS(pbyBuf1, 3, pbyBuf2, 6, 22));
```

## 6.4.28 EC\_GET\_FRM\_WORD

**EC\_GET\_FRM\_WORD (ptr)**

Reads a value of type EC\_T\_WORD (16 bit) at given pointer. The value is swapped on big endian systems.

**Parameters**

- **ptr** – [in] Source buffer

**Returns**

EC\_T\_WORD value (16 bit) from buffer.

```
EC_T_BYTE byFrame[] = {0x01, 0xF4, 0xDD, 0x85, 0x03, 0x00, 0x60, 0xC1, 0x00};
EC_T_WORD wResult = 0;

wResult = EC_GET_FRM_WORD(byFrame);
/* wResult is 0xF401 on little endian systems */

wResult = EC_GET_FRM_WORD(byFrame + 5);
/* wResult is 0x6000 on little endian systems */

wResult = EC_GET_FRM_WORD(byFrame + 2);
/* wResult is 0x85DD on little endian systems */
```

## 6.4.29 EC\_GET\_FRM\_DWORD

**EC\_GET\_FRM\_DWORD (ptr)**

Reads a value of type EC\_T\_DWORD (32 bit) at given pointer. The value is swapped on big endian systems.

**Parameters**

- **ptr** – [in] Source buffer

**Returns**

EC\_T\_DWORD value (32 bit) from buffer.

```
EC_T_BYTE byFrame[] = {0x01, 0xF4, 0xDD, 0x85, 0x03, 0x00, 0x60, 0xC1, 0x00};
EC_T_DWORD dwResult = 0;

dwResult = EC_GET_FRM_DWORD(byFrame);
/* dwResult is 0x85DDF401 on little endian systems */

dwResult = EC_GET_FRM_DWORD(byFrame + 5);
/* dwResult is 0x00C16000 on little endian systems */
```

(continues on next page)

(continued from previous page)

```
dwResult = EC_GET_FRM_DWORD(byFrame + 2);
/* dwResult is 0x000385DD on little endian systems */
```

### 6.4.30 EC\_GET\_FRM\_QWORD

#### **EC\_GET\_FRM\_QWORD** (ptr)

Reads a value of type EC\_T\_QWORD (64 bit) at given pointer. The value is swapped on big endian systems.

##### Parameters

- **ptr** – [in] Source buffer

##### Returns

EC\_T\_QWORD value (64 bit) from buffer.

```
EC_T_BYTE byFrame[] = {0x01, 0xF4, 0xDD, 0x85, 0x03, 0x00, 0x60, 0xC1, 0x00};
EC_T_UINT64 ui64Result = 0;

ui64Result = EC_GET_FRM_QWORD(byFrame + 1);
/* wResult is 0x00C160000385DDF4 on little endian systems */
```

### 6.4.31 EC\_SET\_FRM\_WORD

#### **EC\_SET\_FRM\_WORD** (ptr, w)

Writes a value of type EC\_T\_WORD (16 bit) at given pointer. The value is swapped on big endian systems.

##### Parameters

- **ptr** – [in] Destination buffer
- **w** – [in] 16 bit value

```
EC_T_BYTE byFrame[32];

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

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

### 6.4.32 EC\_SET\_FRM\_DWORD

#### **EC\_SET\_FRM\_DWORD** (ptr, dw)

Writes a value of type EC\_T\_DWORD (32 bit) at given pointer. The value is swapped on big endian systems.

##### Parameters

- **ptr** – [in] Destination buffer
- **dw** – [in] 32 bit value

```
EC_T_BYTE byFrame[32];

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

(continues on next page)

(continued from previous page)

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

### 6.4.33 EC\_SET\_FRM\_QWORD

#### **EC\_SET\_FRM\_QWORD** (ptr, qw)

Writes a value of type EC\_T\_QWORD (64 bit) at given pointer. The value is swapped on big endian systems.

##### Parameters

- **ptr** – [in] Destination buffer
- **qw** – [in] 64 bit value

```
EC_T_BYTE byFrame[32];

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

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

### 6.4.34 EC\_GETBITS

#### **EC\_GETBITS** (pbySrcBuf, pbyDstData, nSrcBitOffs, nBitSize)

Reads a given number of bits from source buffer starting at given bit offset to destination buffer.

---

**Note:** This function should be only used to get bit-aligned data. For byte-aligned data the corresponding functions should be used.

---

##### Parameters

- **pbySrcBuf** – [in] Source buffer to be copied
- **pbyDstData** – [out] Destination buffer where data is copied to
- **nSrcBitOffs** – [in] Source bit offset where data is copied from
- **nBitSize** – [in] Bit count to be copied

##### See also:

- [EC\\_GET\\_FRM\\_WORD](#)
- [EC\\_GET\\_FRM\\_DWORD](#)
- [EC\\_GET\\_FRM\\_QWORD](#)

### 6.4.35 EC\_SETBITS

**EC\_SETBITS** (pbyDstBuf, pbySrcData, nDstBitOffs, nBitSize)

Writes a given number of bits from source data starting at first bit to destination buffer at given bit offset.

---

**Note:** This function should be only used to set bit-aligned data. For byte-aligned data the corresponding functions should be used.

---

#### Parameters

- **pbyDstBuf** – [out] Destination buffer where data is copied to
- **pbySrcData** – [in] Source buffer to be copied, starting with first bit
- **nDstBitOffs** – [in] Destination bit offset where data is copied to
- **nBitSize** – [in] Bit count to be copied

See also:

- [EC\\_SET\\_FRM\\_WORD](#)
- [EC\\_SET\\_FRM\\_DWORD](#)
- [EC\\_SET\\_FRM\\_QWORD](#)

## 6.5 Generic notification interface

One of the parameters the client has to set when registering with the EtherCAT master is a generic notification callback function (`emNotify()`). The master calls this function every time a event (for example an error 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.

This callback function is usually called in the context of the EtherCAT master timer thread or the EtherCAT Real-time Ethernet Driver 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.

### 6.5.1 Notification callback: emNotify

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 `emNotify()` 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.

`typedef EC_T_DWORD (*EC_PF_NOTIFY)(EC_T_DWORD dwCode, EC_T_NOTIFYPARAMS *pParms)`

**struct EC\_T\_NOTIFYPARAMS**

Detailed information about the according notification (EC\_NOTIFY\_...)

**Public Members****EC\_T\_VOID \*pCallerData**

[in] Parameter arbitrarily defined by the application at client registration with the EtherCAT stack

**EC\_T\_BYTE \*pbyInBuf**

[in] Notification input parameters

**EC\_T\_DWORD dwInBufSize**

[in] Size of the input parameter buffer

**EC\_T\_BYTE \*pbyOutBuf**

[out] Notification output (result)

**EC\_T\_DWORD dwOutBufSize**

[in] Size of the output buffer

**EC\_T\_DWORD \*pdwNumOutData**

[out] Actually used buffer size of the output buffer

## 6.5.2 emNotify - EC\_NOTIFY\_STATECHANGED

Notification about a change in the master's operational state. This notification is enabled by default.

**emNotify - EC\_NOTIFY\_STATECHANGED****Parameter**

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

**struct EC\_T\_STATECHANGE****Public Members****EC\_T\_STATE oldState**

old operational state

**EC\_T\_STATE newState**

new operational state

**See also:**

[emIoControl - EC\\_IOCTL\\_SET\\_NOTIFICATION\\_ENABLED](#) for how to control the deactivation

### 6.5.3 emNotify - EC\_NOTIFY\_XXXX

Notification about an error.

#### **emNotify - EC\_NOTIFY\_XXXX**

##### **Parameter**

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

##### **See also:**

*Diagnosis, error detection, error notifications*

### 6.5.4 Feature Pack Master Redundancy Notifications

##### **See also:**

Feature Pack “Master Redundancy”

### 6.5.5 emNotifyApp

By calling this function the generic notification callback function setup by `emRegisterClient()` is called for all clients including RAS.

static EC\_T\_DWORD **ecatNotifyApp** (EC\_T\_DWORD dwCode, *EC\_T\_NOTIFYPARMS* \*pParms)

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

Calls the notification callback functions of all registered clients.

---

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

---

##### **Parameters**

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

##### **Returns**

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

The maximum value for dwCode is defined by EC\_NOTIFY\_APP\_MAX\_CODE

## 6.5.6 emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED

The following notifications can be enabled or disabled.

- *emNotify* - EC\_NOTIFY\_SLAVE\_STATECHANGED (default Off)
- *emNotify* - EC\_NOTIFY\_SLAVES\_STATECHANGED (default Off)
- *emNotify* - EC\_NOTIFY\_SLAVE\_UNEXPECTED\_STATE (default On)
- *emNotify* - EC\_NOTIFY\_SLAVES\_UNEXPECTED\_STATE (default Off)
- *emNotify* - EC\_NOTIFY\_SLAVE\_PRESENCE (default On)
- *emNotify* - EC\_NOTIFY\_SLAVES\_PRESENCE (default Off)
- *emNotify* - EC\_NOTIFY\_SLAVE\_ERROR\_STATUS\_INFO (default On)
- *emNotify* - EC\_NOTIFY\_SLAVES\_ERROR\_STATUS (default Off)
- *emNotify* - EC\_NOTIFY\_NOT\_ALL\_DEVICES\_OPERATIONAL (default On)
- *emNotify* - EC\_NOTIFY\_CYCCMD\_WKC\_ERROR (default On)
- *emNotify* - EC\_NOTIFY\_SB\_MISMATCH (default On)
- *emNotify* - EC\_NOTIFY\_SB\_STATUS (default On)
- *emNotify* - EC\_NOTIFY\_STATUS\_SLAVE\_ERROR (default On)
- *emNotify* - EC\_NOTIFY\_FRAME\_RESPONSE\_ERROR (default On)
- *emNotify* - EC\_NOTIFY\_HC\_TOPOCHGDONE (default On)
- *emNotify* - EC\_NOTIFY\_STATECHANGED (default On)
- *emNotify* - EC\_NOTIFY\_COE\_INIT\_CMD (default Off)
- EC\_NOTIFY\_JUNCTION\_RED\_CHANGE (default Off)
- *emNotify* - 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)
- *emNotify* - EC\_NOTIFY\_SLAVE\_INITCMD\_RESPONSE\_ERROR (default On)
- EC\_NOTIFY\_REF\_CLOCK\_PRESENCE (default Off)
- EC\_NOTIFY\_DCX\_SYNC (default On)
- EC\_NOTIFY\_HC\_DETECTADDGROUPS (default On)
- *emNotify* - EC\_NOTIFY\_FRAMELOSS\_AFTER\_SLAVE (default On)
- *emNotify* - EC\_NOTIFY\_ETH\_LINK\_NOT\_CONNECTED (default On)
- *emNotify* - EC\_NOTIFY\_S2SMBX\_ERROR (default On)
- *emNotify* - EC\_NOTIFY\_SLAVE\_INITCMD\_WKC\_ERROR (default On)
- *emNotify* - EC\_NOTIFY\_BAD\_CONNECTION (default On)

### **emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED**

#### **Parameter**

- pbyInBuf: [in] pointer to EC\_T\_SET\_NOTIFICATION\_ENABLED\_PARMS.
- dwInBufSize: [in] size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

struct **EC\_T\_SET\_NOTIFICATION\_ENABLED\_PARMS****Public Members****EC\_T\_DWORD dwClientId**

[in] Client ID, 0: Master

**EC\_T\_DWORD dwCode**

[in] Notification code or EC\_ALL\_NOTIFICATIONS

**EC\_T\_DWORD dwEnabled**

[in] Enable, disable or reset to default notification. See EC\_NOTIFICATION\_flags

Notifications are given to clients if enabled for dwClientId = 0 AND corresponding dwClientId.

### **6.5.7 emIoControl - EC\_IOCTL\_GET\_NOTIFICATION\_ENABLED**

The enabled state of notifications can be retrieved using *emIoControl - EC\_IOCTL\_GET\_NOTIFICATION\_ENABLED*.**emIoControl - EC\_IOCTL\_GET\_NOTIFICATION\_ENABLED****Parameter**

- pbyInBuf: [in] pointer to EC\_T\_GET\_NOTIFICATION\_ENABLED\_PARMS.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Pointer to EC\_T\_BOOL to carry out current enable set.
- dwOutBufSize: [in] Size of the output buffer provided at pbyOutBuf in bytes.
- pdwNumOutData: [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer.

**Return**

EC\_E\_NOERROR or error code

struct **EC\_T\_GET\_NOTIFICATION\_ENABLED\_PARMS**

## Public Members

**EC\_T\_DWORD dwClientId**  
 [in] Client ID, 0: Master

**EC\_T\_DWORD dwCode**  
 [in] Notification code

### See also:

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED*

## 6.6 Slave control and status functions

### 6.6.1 emGetNumConfiguredSlaves

static EC\_T\_DWORD **ecatGetNumConfiguredSlaves** (EC\_T\_VOID)

**EC\_T\_DWORD emGetNumConfiguredSlaves** (EC\_T\_DWORD dwInstanceID)

Returns number of slaves which are configured in the ENI.

#### Parameters

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

#### Returns

Number of slaves

#### emGetNumConfiguredSlaves() Example

```
EC_T_DWORD dwSlaveCnt = emGetNumConfiguredSlaves(dwInstanceId);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
  " Slave Count: %d", dwSlaveCnt));
```

### 6.6.2 emGetNumConnectedSlaves

static EC\_T\_DWORD **ecatGetNumConnectedSlaves** (EC\_T\_VOID)

**EC\_T\_DWORD emGetNumConnectedSlaves** (EC\_T\_DWORD dwInstanceID)

Get amount of currently connected slaves.

#### Parameters

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

#### Returns

Number of connected slaves

**emGetNumConnectedSlaves() Example**

```
EC_T_DWORD dwSlaveCnt = emGetNumConnectedSlaves(dwInstanceId);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
    " Slave Count: %d", dwSlaveCnt));
```

**6.6.3 emGetSlaveId**

static EC\_T\_DWORD **ecatGetSlaveId** (EC\_T\_WORD wStationAddress)

**EC\_T\_DWORD emGetSlaveId** (EC\_T\_DWORD dwInstanceId, EC\_T\_WORD wStationAddress)

Determines the slave ID using the slave station address.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **wStationAddress** – [in] Station address of the slave

**Returns**

Slave ID or INVALID\_SLAVE\_ID if the slave could not be found or stack is not initialized

**emGetSlaveId() Example**

```
/* get slave ID using slave station address */
EC_T_WORD wStationAddress = 1002;
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, wStationAddress);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
    " Slave ID: %d", dwSlaveId));
```

**6.6.4 emGetSlaveIdAtPosition**

static EC\_T\_DWORD **ecatGetSlaveIdAtPosition** (EC\_T\_WORD wAutoIncAddress)

**EC\_T\_DWORD emGetSlaveIdAtPosition** (  
     EC\_T\_DWORD dwInstanceId,  
     EC\_T\_WORD wAutoIncAddress  
 )

Determines the slave ID using the slave auto increment address.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **wAutoIncAddress** – [in] Auto increment address of the slave

**Returns**

Slave ID or INVALID\_SLAVE\_ID if no slave matching wAutoIncAddress can be found

### emGetSlaveIdAtPosition() Example

```
/* get slave ID using configured auto increment address */
EC_T_WORD wAutoIncAdress = 0xFFFFB;
EC_T_DWORD dwSlavePos = emGetSlaveIdAtPosition(dwInstanceId, wAutoIncAdress);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
    " Slave Position: %d", dwSlavePos));
```

## 6.6.5 emSetSlaveState

```
static EC_T_DWORD ecatSetSlaveState (
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wDeviceState,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emSetSlaveState (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wDeviceState,
    EC_T_DWORD dwTimeout
)
```

Set a specified slave into the requested EtherCAT state.

The requested state shall not be higher than the overall operational state. DEVICE\_STATE\_BOOTSTRAP can only be requested if the slave's state is INIT. This function may not be called from within the JobTask's context.

If the function is called with EC\_NOWAIT, the client may wait for reaching the requested state using the notification callback (EC\_NOTIFY\_SLAVE\_STATECHANGED).

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wDeviceState** – [in] Requested device state. See DEVICE\_STATE\_...
- **dwTimeout** – [in] Timeout [ms] This function will block until the requested state is reached or the timeout elapsed. If the timeout value is set to EC\_NOWAIT the function will return immediately.

#### Returns

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master isn't initialized or denies the requested state, see comments below
- **EC\_E\_INVALIDPARM** if dwInstanceId is out of range or BOOTSTRAP was requested for a slave that does not support it
- **EC\_E\_NOTFOUND** if no slave matching dwSlaveId can be found
- **EC\_E\_TIMEOUT** if dwTimeout elapsed during the API call
- **EC\_E\_BUSY** if the master cannot execute the request at this time, the function has to be called at a later time
- **EC\_E\_NOTREADY** if the working counter was not set when requesting the slave's state (slave may not be connected or did not respond)

- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive

**DEVICE\_STATE\_UNKNOWN**

Slave in unknown state

**DEVICE\_STATE\_INIT**

Slave in INIT state

**DEVICE\_STATE\_PREOP**

Slave in PREOP state

**DEVICE\_STATE\_BOOTSTRAP**

Slave in BOOTSTRAP state

**DEVICE\_STATE\_SAFEOP**

Slave in SAFEOP state

**DEVICE\_STATE\_OP**

Slave in OP state

**DEVICE\_STATE\_ERROR**

Slave in error state

**emSetSlaveState() Example**

```
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1001);
EC_T_WORD wDeviceState = DEVICE_STATE_PREOP;
dwRes = emSetSlaveState(dwInstanceId, dwSlaveId, wDeviceState, 5000 /* timeout */);
```

**See also:**[emGetSlaveId\(\)](#)**6.6.6 emSetSlaveStateReq**

```
static EC_T_DWORD ecatSetSlaveStateReq (
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wDeviceState,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emSetSlaveStateReq (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wDeviceState,
    EC_T_DWORD dwTimeout
)
```

Request to set a specified slave into the requested EtherCAT state and return immediately.

The requested state shall not be higher than the overall operational state. DEVICE\_STATE\_BOOTSTRAP can only be requested if the slave's state is INIT.

**Parameters**

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

- **dwSlaveId** – [in] Slave ID
- **wDeviceState** – [in] Requested device state. See DEVICE\_STATE\_...
- **dwTimeout** – [in] Timeout [ms]

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized or denies the requested state, see comments below
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or BOOTSTRAP was requested for a slave that does not support it
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_BUSY* if the master cannot execute the request at this time, the function has to be called at a later time
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive

**emSetSlaveStateReq() Example**

```
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1001);
EC_T_WORD wDeviceState = DEVICE_STATE_PREOP;
dwRes = emSetSlaveStateReq(dwInstanceId, dwSlaveId, wDeviceState, 5000 /* timeout
→ */);
```

**See also:**[emSetSlaveState\(\)](#)**See also:**[emGetSlaveId\(\)](#)**6.6.7 emGetSlaveState**

```
static EC_T_DWORD ecatGetSlaveState (
    EC_T_DWORD dwSlaveId,
    EC_T_WORD *pwCurrDevState,
    EC_T_WORD *pwReqDevState
)
EC_T_DWORD emGetSlaveState (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD *pwCurrDevState,
    EC_T_WORD *pwReqDevState
)
```

Get the slave state.

The slave state is always read automatically from the AL\_STATUS register whenever necessary. It is not forced by calling this function. This function may be called from within the JobTask's context.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID

- **pwCurrDevState** – [out] Current slave state.
- **pwReqDevState** – [out] Requested slave state

**Returns**

- *EC\_E\_NOERROR* if successful.
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or the output pointers are EC\_NULL
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present.
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found

**emGetSlaveState() Example**

```
/* get slave state */
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1002);
EC_T_WORD pwCurrDevState = 0;
EC_T_WORD wReqDevState = 0;
dwRes = emGetSlaveState(dwInstanceId, dwSlaveId, &pwCurrDevState, &wReqDevState);
```

**See also:**

- [emGetSlaveId\(\)](#)
- [emSetSlaveState\(\)](#)

**6.6.8 emIsSlavePresent**

This function may be called from within the JobTask. Since Slave Id is a parameter, valid response only can be retrieved after calling [emConfigureNetwork\(\)](#).

```
static EC_T_DWORD ecatIsSlavePresent (
    EC_T_DWORD dwSlaveId,
    EC_T_BOOL *pbPresence
)
```

```
EC_T_DWORD emIsSlavePresent (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwSlaveId,
    EC_T_BOOL *pbPresence
)
```

) Returns whether a specific slave is currently connected to the Bus.

This function may be called from within the JobTask.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **pbPresence** – [out] EC\_TRUE if slave is currently connected to the bus, EC\_FALSE if not.

**Returns**

- *EC\_E\_NOERROR* if successful

- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found

### **emIsSlavePresent() Example**

```
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1002);
EC_T_BOOL bPresence = EC_FALSE;

/* returns whether a specific slave is currently connected to network */
dwRes = emIsSlavePresent(dwInstanceId, dwSlaveId, &bPresence);
```

#### See also:

[emGetSlaveId\(\)](#)

### **6.6.9 emGetSlaveProp**

```
static EC_T_BOOL ecatGetSlaveProp(
    EC_T_DWORD dwSlaveId,
    EC_T_SLAVE_PROP *pSlaveProp
)
```

```
EC_T_BOOL emGetSlaveProp(
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwSlaveId,
    EC_T_SLAVE_PROP *pSlaveProp
)
```

) Determines the properties of the slave device.

*Deprecated:*

Use [emGetCfgSlaveInfo](#) instead

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **pSlaveProp** – [out] Slave properties

#### Returns

EC\_TRUE if the slave exists, EC\_FALSE if no slave matching dwSlaveId can be found

### **emGetSlaveProp() Example**

```
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1002);
EC_T_SLAVE_PROP oSlaveProp;
OsMemset(&oSlaveProp, 0, sizeof(EC_T_SLAVE_PROP));
EC_T_BOOL bSlaveProp = emGetSlaveProp(dwInstanceId, dwSlaveId, &oSlaveProp);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO, " Slave ID: %d",  
    bSlaveProp));
```

#### See also:

[emGetSlaveId\(\)](#)

## 6.6.10 emSlaveSerializeMbxTfers

static EC\_T\_DWORD **ecatSlaveSerializeMbxTfers** (EC\_T\_DWORD dwSlaveId)

```
EC_T_DWORD emSlaveSerializeMbxTfers (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId
)
```

) Serializes all mailbox transfers to the specified slave.

The parallel (overlapped) usage of more than one protocol (CoE, EoE, FoE, etc.) will be disabled. By default parallel mailbox transfers are enabled.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID

### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave does not support mailbox transfers

## emSlaveSerializeMbxTfers() Example

```
/* serialize all mailbox transfers to specified slave */
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1001);
dwRes = emSlaveSerializeMbxTfers(dwInstanceId, dwSlaveId);
```

### See also:

[emGetSlaveId\(\)](#)

## 6.6.11 emSlaveParallelMbxTfers

static EC\_T\_DWORD **ecatSlaveParallelMbxTfers** (EC\_T\_DWORD dwSlaveId)

```
EC_T_DWORD emSlaveParallelMbxTfers (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId
)
```

) Re-enable the parallel mailbox transfers to the specified slave.

Allows parallel (overlapped) usage of more than one protocol (CoE, EoE, FoE, etc.). By default parallel mailbox transfers are enabled.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave does not support mailbox transfers

**emSlaveParallelMbxTfers() Example**

```
/* re-enable parallel mailbox transfers to specified slave */
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1001);
dwRes = emSlaveParallelMbxTfers(dwInstanceId, dwSlaveId);
```

**See also:**

[emGetSlaveId\(\)](#)

**6.6.12 emIoControl - EC\_IOCTL\_SET\_MBX\_RETRYACCESS\_PERIOD**

Sets the mailbox retry access period in milliseconds for a specific slave. If a slave rejects a mailbox access because of a busy state, the master restarts mailbox access after that period of time.

**emIoControl - EC\_IOCTL\_SET\_MBX\_RETRYACCESS\_PERIOD****Parameter**

- pbyInBuf: [in] Pointer to a size 6 byte array. The first 4 bytes must contain the slave id (EC\_T\_DWORD), the last 2 bytes the new retry access period in milliseconds(EC\_T\_WORD).
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

*EC\_E\_NOERROR* or error code

By default, the retry access period is set to 25 milliseconds.

**6.6.13 emNotify - EC\_NOTIFY\_SLAVE\_STATECHANGED**

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

**emNotify - EC\_NOTIFY\_SLAVE\_STATECHANGED****Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_SLAVE\_STATECHANGED\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL

- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

```
struct EC_T_SLAVE_STATECHANGED_NTFY_DESC
```

### Public Members

*EC\_T\_SLAVE\_PROP* **SlaveProp**  
Slave properties

*EC\_T\_STATE* **newState**  
New slave state

#### See also:

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the activation

## 6.6.14 emNotify - EC\_NOTIFY\_SLAVES\_STATECHANGED

Collects *emNotify - EC\_NOTIFY\_SLAVE\_STATECHANGED*

This notification is disabled by default.

#### See also:

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the activation

### emNotify - EC\_NOTIFY\_SLAVES\_STATECHANGED

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_SLAVES\_STATECHANGED\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

```
struct EC_T_SLAVES_STATECHANGED_NTFY_DESC_ENTRY
```

### Public Members

*EC\_T\_WORD* **wStationAddress**  
Slave station address

*EC\_T\_BYTE* **byState**  
New slave state

## 6.6.15 emWriteSlaveRegister

```
static EC_T_DWORD ecatWriteSlaveRegister (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wRegisterOffset,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emWriteSlaveRegister (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wRegisterOffset,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen,
    EC_T_DWORD dwTimeout
)
```

Writes data into the ESC memory of a specified slave.

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

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

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wRegisterOffset** – [in] Register offset, e.g. use 0x0120 to write to the AL Control register.
- **pbyData** – [in] Buffer containing transferred data
- **wLen** – [in] Number of bytes to send
- **dwTimeout** – [in] Timeout [ms]

### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC\_NOWAIT
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC\_E\_TIMEOUT* if dwTimeout elapsed during the API call
- *EC\_E\_BUSY* another transfer request is already pending or the master or the corresponding slave is currently changing its operational state

- *EC\_E\_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC\_E\_INVALIDSIZE* if the size of the complete command does not fit into a single Ethernet frame. The maximum amount of data to transfer must not exceed 1486 bytes

### **emWriteSlaveRegister() Example**

```
dwRes = emWriteSlaveRegister(dwInstanceId, EC_TRUE,
 1001, wRegisterOffset, abyData, wLen, 5000 /* timeout */);
```

## 6.6.16 emWriteSlaveRegisterReq

```
static EC_T_DWORD ecatWriteSlaveRegisterReq(
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wRegisterOffset,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen
)
EC_T_DWORD emWriteSlaveRegisterReq(
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wRegisterOffset,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen
)
```

) Requests a data write transfer into the ESC memory of a specified slave and returns immediately.

A notification *EC\_NOTIFY\_SLAVE\_REGISTER\_TRANSFER* is given on completion. This function may be called from within the JobTask's context.

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

### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTferId** – [in] Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within *EC\_T\_SLAVEREGISTER\_TRANSFER\_NTFY\_DESC*
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing

- **wRegisterOffset** – [in] Register offset. I.e. use 0x0120 to write to the AL Control register
- **pbyData** – [in] Buffer containing transferred data
- **wLen** – [in] Number of bytes to send

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC\_NOWAIT
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC\_E\_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

**emWriteSlaveRegisterReq() Example**

```
EC_T_DWORD dwTferId = 1234; /* arbitrary unique ID from application */
/* assigned by application. should be unique for each transfer */

/* write data into slave's ESC memory */
dwRes = emWriteSlaveRegisterReq(dwInstanceId, dwClientId, dwTferId,
    EC_TRUE, 1001, wRegisterOffset, abyData, wLen);
```

**See also:**

*emNotify - EC\_NOTIFY\_SLAVE\_REGISTER\_TRANSFER*

**6.6.17 emReadSlaveRegister**

```
static EC_T_DWORD ecatReadSlaveRegister(
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wRegisterOffset,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emReadSlaveRegister (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wRegisterOffset,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen,
    EC_T_DWORD dwTimeout
)
```

Reads data from the ESC memory of a specified slave.

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

**Parameters**

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

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC\_NOWAIT
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC\_E\_TIMEOUT* if dwTimeout elapsed during the API call
- *EC\_E\_BUSY* another transfer request is already pending or the master or the corresponding slave is currently changing its operational state
- *EC\_E\_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC\_E\_INVALIDSIZE* if the size of the complete command does not fit into a single Ethernet frame. The maximum amount of data to transfer must not exceed 1486 bytes

**emReadSlaveRegister() Example**

```
/* read ESC memory */
EC_T_BYTE abyData[2] = {0, 0};

dwRes = emReadSlaveRegister(dwInstanceId, EC_TRUE, 1001,
    0 /* ADO */, abyData, 2, 5000 /* timeout */);
```

**6.6.18 emReadSlaveRegisterReq**

```
static EC_T_DWORD ecatReadSlaveRegisterReq(
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wRegisterOffset,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen
)
```

```
EC_T_DWORD emReadSlaveRegisterReq(
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wRegisterOffset,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen
)
```

) Requests data read transfer from the ESC memory of a specified slave and returns immediately.

A notification EC\_NOTIFY\_SLAVE\_REGISTER\_TRANSFER is given on completion. This function may be called from within the JobTask's context.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTferId** – [in] Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within [EC\\_T\\_SLAVEREGISTER\\_TRANSFER\\_NTFY\\_DESC](#)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wRegisterOffset** – [in] Register offset, e.g. use 0x0130 to read the AL Status register.
- **pbyData** – [out] Buffer receiving transferred data
- **wLen** – [in] Number of bytes to receive

#### Returns

- [EC\\_E\\_NOERROR](#) if successful.
- [EC\\_E\\_INVALIDSTATE](#) if master isn't initialized
- [EC\\_E\\_INVALIDPARM](#) if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC\_NOWAIT
- [EC\\_E\\_SLAVE\\_NOT\\_PRESENT](#) if slave not present.
- [EC\\_E\\_NOTFOUND](#) if no slave matching bFixedAddressing / wSlaveAddress can be found
- [EC\\_E\\_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.

**emReadSlaveRegisterReq() Example**

```

/* read ESC memory (non-blocking) */
EC_T_BYTE abyData[2] = {0, 0};
EC_T_DWORD dwTferId = 1234; /* arbitrary unique ID from application */

/* get data read transfer from ESC memory of specified slave */
dwRes = emReadSlaveRegisterReq(dwInstanceId, dwClientId, dwTferId,
    EC_TRUE, 1001, 0 /* ADO */, abyData, 2);

```

**See also:***emNotify - EC\_NOTIFY\_SLAVE\_REGISTER\_TRANSFER***6.6.19 emNotify - EC\_NOTIFY\_SLAVE\_REGISTER\_TRANSFER**

This notification is given, when a slave register transfer is completed.

**emNotify - EC\_NOTIFY\_SLAVE\_REGISTER\_TRANSFER****Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_SLAVEREGISTER\_TRANSFER\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

```
struct EC_T_SLAVEREGISTER_TRANSFER_NTFY_DESC
```

**Public Members****EC\_T\_DWORD dwTferId**

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

**EC\_T\_DWORD dwResult**

Result of Slave register transfer

**EC\_T\_BOOL bRead**

EC\_TRUE: Read register, EC\_FALSE: Write register transfer

**EC\_T\_WORD wFixedAddr**

Station address of slave

**EC\_T\_WORD wRegisterOffset**

Register offset

**EC\_T\_WORD wLen**

Length of slave register transfer

**EC\_T\_BYTEx\*pbyData**

Pointer to the data read

**EC\_T\_WORD wWkc**  
Received working counter

## 6.6.20 emReadSlaveEEProm

```
static 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
)
EC_T_DWORD emReadSlaveEEProm (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wEEPROMStartOffset,
    EC_T_WORD *pwReadData,
    EC_T_DWORD dwReadLen,
    EC_T_DWORD *pdwNumOutData,
    EC_T_DWORD dwTimeout
)
) Read EEPROM data from slave.
```

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

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wEEPROMStartOffset** – [in] Word address to start EEPROM read from
- **pwReadData** – [in] Pointer to EC\_T\_WORD array to carry the read data
- **dwReadLen** – [in] Size of the EC\_T\_WORD array provided at pwReadData (in EC\_T\_WORDS)
- **pdwNumOutData** – [out] Pointer to EC\_T\_DWORD carrying actually read data (in EC\_T\_WORDS) after completion
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC\_NOWAIT.

### Returns

*EC\_E\_NOERROR* or error code

### emReadSlaveEEPROM() Example

```
/* read EEPROM data from slave */
EC_T_WORD awData[16];
EC_T_DWORD dwNumOutData = 0;
OsMemset(awData, 0, sizeof(awData));

dwRes = emReadSlaveEEPROM(dwInstanceId, EC_TRUE, 1001,
    /* WORD offset */, awData, EC_NUMOFELEMENTS(awData),
    &dwNumOutData, 5000 /* timeout */);
```

### 6.6.21 emReadSlaveEEPROMReq

```
static EC_T_DWORD ecatReadSlaveEEPROMReq(
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    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
)
EC_T_DWORD emReadSlaveEEPROMReq(
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    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
)
```

Requests a EEPROM data read operation from slave and returns immediately.

A EC\_NOTIFY\_EEPROM\_OPERATION is given on completion or timeout. This function may be called from within the JobTask's context.

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTferId** – [in] Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within [EC\\_T\\_EEPROM\\_OPERATION\\_NTFY\\_DESC](#)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wEEPROMStartOffset** – [in] Word address to start EEPROM read from
- **pwReadData** – [out] Pointer to EC\_T\_WORD array to carry the read data, must be valid until the operation complete

- **dwReadLen** – [in] Size of the EC\_T\_WORD array provided at pwReadData (in EC\_T\_WORDS)
- **pdwNumOutData** – [out] Pointer to EC\_T\_DWORD carrying actually read data (in EC\_T\_WORDS) after completion
- **dwTimeout** – [in] Timeout [ms]

**Returns***EC\_E\_NOERROR* or error code**emReadSlaveEEPROM() Example**

```
EC_T_DWORD dwTferId = 1234; /* arbitrary unique ID from application */
EC_T_WORD awData[16];
EC_T_DWORD dwNumOutData = 0;
OsMemset(awData, 0, sizeof(awData));

/* requests EEPROM data read operation from slave */
dwRes = emReadSlaveEEPROMReq(dwInstanceId, dwClientId,
    dwTferId, EC_TRUE, 1001, 7 /* WORD offset */, awData,
    EC_NUMOFELEMENTS(awData), &dwNumOutData, 5000 /* timeout */);
```

**See also:***emNotify - EC\_NOTIFY\_EEPROM\_OPERATION***6.6.22 emWriteSlaveEEPROM**

```
static 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
)
EC_T_DWORD emWriteSlaveEEPROM (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wEEPROMStartOffset,
    EC_T_WORD *pwWriteData,
    EC_T_DWORD dwWriteLen,
    EC_T_DWORD dwTimeout
)
```

Write EEPROM data to slave.

The EEPROM's CRC is updated automatically. *emResetSlaveController()* is needed to reload the alias address in register 0x12. This function may not be called from within the JobTask's context.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing

- **wEEPromStartOffset** – [in] Word address to start EEPROM Write from
- **pwWriteData** – [in] Pointer to WORD array carrying the write data
- **dwWriteLen** – [in] Size of Write Data WORD array (in WORDS)
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC\_NOWAIT.

**Returns***EC\_E\_NOERROR* or error code**emWriteSlaveEEPROM() Example**

```
dwRes = emWriteSlaveEEPROM(dwInstanceId, EC_TRUE,
    1001, 0 /* offset */, awData, dwWriteLen, 5000 /* timeout */);
```

**See also:***emResetSlaveController()***6.6.23 emWriteSlaveEEPROMReq**

```
static EC_T_DWORD ecatWriteSlaveEEPROMReq(
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wEEPromStartOffset,
    EC_T_WORD *pwWriteData,
    EC_T_DWORD dwWriteLen,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emWriteSlaveEEPROMReq(
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wEEPromStartOffset,
    EC_T_WORD *pwWriteData,
    EC_T_DWORD dwWriteLen,
    EC_T_DWORD dwTimeout
)
```

) Requests a EEPROM data write operation from slave and returns immediately.

The EEPROM's CRC is updated automatically. A reset of the slave controller is needed to reload the alias address in register 0x12. A *EC\_NOTIFY\_EEPROM\_OPERATION* is given on completion or timeout. This function may be called from within the JobTask's context.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTferId** – [in] Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within *EC\_T\_EEPROM\_OPERATION\_NTFY\_DESC*

- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wEEPROMStartOffset** – [in] Word address to start EEPROM Write from.
- **pwWriteData** – [in] Pointer to WORD array carrying the write data, must be valid until operation complete
- **dwWriteLen** – [in] Size of Write Data WORD array (in WORDS)
- **dwTimeout** – [in] Timeout [ms]

**Returns***EC\_E\_NOERROR* or error code**emWriteSlaveEEPROMReq() Example**

```
EC_T_DWORD dwTferId = 1234; /* arbitrary unique ID from application */
dwRes = emWriteSlaveEEPROMReq(dwInstanceId, dwClientId, dwTferId,
    EC_TRUE, 1001, 0 /* offset */, awData, dwWriteLen, 5000 /* timeout */);
```

**See also:**

- *emResetSlaveController()*
- *emNotify - EC\_NOTIFY\_EEPROM\_OPERATION*

**6.6.24 emAssignSlaveEEPROM**

```
static EC_T_DWORD ecatAssignSlaveEEPROM (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL bSlavePDIAccessEnable,
    EC_T_BOOL bForceAssign,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emAssignSlaveEEPROM (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL bSlavePDIAccessEnable,
    EC_T_BOOL bForceAssign,
    EC_T_DWORD dwTimeout
)
```

Set EEPROM Assignment to PDI or EtherCAT Master.

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

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **bSlavePDIAccessEnable** – [in] EC\_TRUE: EEPROM assigned to slave PDI application, EC\_FALSE: EEPROM assigned to EC-Master

- **bForceAssign** – [in] Force Assignment of EEPROM (only for ECat Master Assignment)
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC\_NOWAIT.

**Returns***EC\_E\_NOERROR* or error code**emAssignSlaveEEPROM() Example**

```
/* set EEPROM access to PDI */
dwRes = emAssignSlaveEEPROM(dwInstanceId, EC_TRUE,
    1001, EC_TRUE /* PDI access */, EC_TRUE /* force */, 5000 /* timeout */);
```

**6.6.25 emAssignSlaveEEPROMReq**

```
static EC_T_DWORD ecatAssignSlaveEEPROMReq(
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL bSlavePDIAccessEnable,
    EC_T_BOOL bForceAssign,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emAssignSlaveEEPROMReq (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL bSlavePDIAccessEnable,
    EC_T_BOOL bForceAssign,
    EC_T_DWORD dwTimeout
)
    Requests EEPROM Assignment to PDI or EtherCAT Master operation and return immediately.
    EC_NOTIFY_EEPROM_OPERATION is given on completion or timeout. This function may be called from
    within the JobTask's context.
```

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTferId** – [in] Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within *EC\_T\_EEPROM\_OPERATION\_NTFY\_DESC*
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **bSlavePDIAccessEnable** – [in] EC\_TRUE: EEPROM assigned to slave PDI application, EC\_FALSE: EEPROM assigned to EC-Master

- **bForceAssign** – [in] Force Assignment of EEPROM (only for ECat Master Assignment)
- **dwTimeout** – [in] Timeout [ms]

**Returns***EC\_E\_NOERROR* or error code**emAssignSlaveEEPROMReq() Example**

```
/* set EEPROM access to PDI (non-blocking) */
EC_T_DWORD dwTferId = 1234; /* arbitrary unique ID from application */
/* requests EEPROM Assignment to PDI or EtherCAT Master operation and return
↳ immediately */
dwRes = emAssignSlaveEEPROMReq(dwInstanceId, dwClientId, dwTferId,
    EC_TRUE, 1001, EC_TRUE /* PDI access */, EC_TRUE /* force */, 5000 /* timeout
↳ */);
```

**See also:***emNotify - EC\_NOTIFY\_EEPROM\_OPERATION***6.6.26 emActiveSlaveEEPROM**

```
static EC_T_DWORD ecatActiveSlaveEEPROM (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL *pbSlavePDIAccessActive,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emActiveSlaveEEPROM (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL *pbSlavePDIAccessActive,
    EC_T_DWORD dwTimeout
)
    Check whether EEPROM is marked access active by Slave PDI application.
```

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

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pbSlavePDIAccessActive** – [out] Pointer to Boolean value: EC\_TRUE: EEPROM active by PDI application, EC\_FALSE: EEPROM not active
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC\_NOWAIT

**Returns***EC\_E\_NOERROR* or error code

**emActiveSlaveEEPROM() Example**

```
/* check whether EEPROM is marked access active by PDI */
EC_T_BOOL bSlavePDIAccessActive = EC_FALSE;
dwRes = emActiveSlaveEEPROM(dwInstanceId, EC_TRUE, 1001,
    &bSlavePDIAccessActive, 5000 /* timeout */);
```

**6.6.27 emActiveSlaveEEPROMReq**

```
static EC_T_DWORD ecatActiveSlaveEEPROMReq(
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL *pbSlavePDIAccessActive,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emActiveSlaveEEPROMReq (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL *pbSlavePDIAccessActive,
    EC_T_DWORD dwTimeout
)
```

Requests EEPROM is marked access active by Slave PDI application check and returns immediately.

A EC\_NOTIFY\_EEPROM\_OPERATION is given on completion or timeout. This function may be called from within the JobTask's context.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTferId** – [in] Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within [EC\\_T\\_EEPROM\\_OPERATION\\_NTFY\\_DESC](#)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pbSlavePDIAccessActive** – [out] Pointer to Boolean value: EC\_TRUE: EEPROM active by PDI application, EC\_FALSE: EEPROM not active. Must be valid until operation complete
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC\_NOWAIT.

**Returns**

[EC\\_E\\_NOERROR](#) or error code

**emActiveSlaveEEPROMReq() Example**

```
/* check whether EEPROM is marked access active by PDI (non-blocking) */
EC_T_DWORD dwTferId = 1234; /* arbitrary unique ID from application */
EC_T_BOOL bSlavePDIAccessActive = EC_FALSE;
dwRes = emActiveSlaveEEPROMReq(dwInstanceId, dwClientId, dwTferId,
    EC_TRUE, 1001, &bSlavePDIAccessActive, 5000 /* timeout */);
```

**See also:***emNotify - EC\_NOTIFY\_EEPROM\_OPERATION***6.6.28 emReloadSlaveEEPROM**

```
static EC_T_DWORD ecatReloadSlaveEEPROM (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emReloadSlaveEEPROM (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD dwTimeout
)
```

Causes a slave to reload its EEPROM values to ESC registers.

Alias address at 0x12 is not reloaded through this command, this is prevented by the slave hardware. The slave controller must be reset to reload the alias address. This function may not be called from within the JobTask's context.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC\_NOWAIT

**Returns***EC\_E\_NOERROR* or error code**emReloadSlaveEEPROM() Example**

```
dwRes = emReloadSlaveEEPROM(dwInstanceId, EC_TRUE, 1001, 5000/* timeout */);
```

**See also:***emResetSlaveController()*

## 6.6.29 emReloadSlaveEEPROMReq

```
static EC_T_DWORD ecatReloadSlaveEEPROMReq (
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emReloadSlaveEEPROMReq (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD dwTimeout
)
```

Request a slave to reload its EEPROM values to ESC registers, and returns immediately.

Alias address at 0x12 is not reloaded through this command, this is prevented by the slave hardware. The slave controller must be reset to reload the alias address. A EC\_NOTIFY\_EEPROM\_OPERATION is given on completion or timeout. This function may be called from within the JobTask's context.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTferId** – [in] Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within [EC\\_T\\_EEPROM\\_OPERATION\\_NTFY\\_DESC](#)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **dwTimeout** – [in] Timeout [ms]

### Returns

[EC\\_E\\_NOERROR](#) or error code

### emReloadSlaveEEPROMReq() Example

```
EC_T_DWORD dwTferId = 1234; /* arbitrary unique ID from application */
dwRes = emReloadSlaveEEPROMReq(dwInstanceID, dwClientId,
    dwTferId, EC_TRUE, 1001, 5000 /* timeout */);
```

### See also:

- [emNotify - EC\\_NOTIFY\\_EEPROM\\_OPERATION](#)
- [emResetSlaveController\(\)](#)

## 6.6.30 emNotify - EC\_NOTIFY\_EEPROM\_OPERATION

This notification is given, when a slave EEPROM operation is completed.

### emNotify - EC\_NOTIFY\_EEPROM\_OPERATION

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_EEPROM\_OPERATION\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

```
struct EC_T_EEPROM_OPERATION_NTFY_DESC
```

#### Public Members

##### **EC\_T\_DWORD dwTransferId**

Transfer ID. For every new EEPROM operation a unique ID has to be assigned. This ID can be used after completion to identify the transfer

##### **EC\_T\_EEPROM\_OPERATION\_TYPE eType**

Type of EEPROM operation

##### **EC\_T\_DWORD dwResult**

Result of EEPROM operation

##### **EC\_T\_SLAVE\_PROP SlaveProp**

Slave properties

```
union _EC_T_EEPROM_OPERATION_NTFY_DESC_RESULT
```

```
struct _EC_T_EEPROM_OPERATION_NTFY_DESC_RESULT_ACTIVE
```

#### Public Members

##### **EC\_T\_BOOL bSlavePDIAccessActive**

EC\_TRUE: EEPROM active by PDI application, EC\_FALSE: EEPROM not active

```
struct _EC_T_EEPROM_OPERATION_NTFY_DESC_RESULT_READ
```

**Public Members****EC\_T\_WORD wEEPROMStartOffset**

Start address of EEPROM operation. Given by API

**EC\_T\_WORD \*pwData**

Pointer to WORD array contains the data. Given by API

**EC\_T\_DWORD dwReadLen**

Number of Words to be read. Given by API

**EC\_T\_DWORD dwNumOutData**

Number of Words actually read from EEPROM

struct **\_EC\_T\_EEPROM\_OPERATION\_NTFY\_DESC\_RESULT\_WRITE****Public Members****EC\_T\_WORD wEEPROMStartOffset**

Start address of EEPROM operation. Given by API

**EC\_T\_WORD \*pwData**

Pointer to WORD array contains the data. Given by API

**EC\_T\_DWORD dwWriteLen**

Number of Words to be written. Given by API

enum **EC\_T\_EEPROM\_OPERATION\_TYPE***Values:*enumerator **eEEPROMOp\_Unknown**

Unknown EEPROM operation, only for internal use

enumerator **eEEPROMOp\_Assign**

Assign slave EEPROM operation, used by emAssignSlaveEEPROMReq

enumerator **eEEPROMOp\_Active**

Active slave EEPROM operation, used by emActiveSlaveEEPROMReq

enumerator **eEEPROMOp\_Read**

Read slave EEPROM operation, used by emReadSlaveEEPROMReq

enumerator **eEEPROMOp\_Write**

Write slave EEPROM operation, used by emWriteSlaveEEPROMReq

enumerator **eEEPROMOp\_Reload**

Reload slave EEPROM operation, used by emReloadSlaveEEPROMReq

enumerator **eEEPROMOp\_Reset**

Reset slave EEPROM operation, used by emResetSlaveController

### 6.6.31 emResetSlaveController

```
static EC_T_DWORD ecatResetSlaveController (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emResetSlaveController (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD dwTimeout
)
Reset EtherCAT slave controller (ESC)
```

A special sequence of three independent and consecutive frames/commands is sent to the slave (reset register ECAT 0x0040 or PDI 0x0041), after which the slave resets. If that fails, the reset sequence is repeated until it succeeds or the timeout expires. The ESC must support resetting and the slave state should be INIT when calling this function. The number of acyclic frames per cycle *EC\_T\_INIT\_MASTER\_PARMS.dwMaxAcycFramesPerCycle* must be at least 3, otherwise an error is returned. This function may not be called from within the JobTask's context.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC\_NOWAIT

#### Returns

- *EC\_E\_NOERROR* or error code
- *EC\_E\_NOTSUPPORTED* if *EC\_T\_INIT\_MASTER\_PARMS.dwMaxAcycFramesPerCycle* is less than 3
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive

#### emResetSlaveController() Example

```
/* reset EtherCAT slave controller */
dwRes = emResetSlaveController(dwInstanceId, EC_TRUE, 1001, 5000 /* timeout */);
```

## 6.6.32 emIoControl - EC\_IOCTL\_ALL\_SLAVES\_MUST\_REACH\_MASTER\_STATE

Specifies if all the slaves must reach the requested master state.

### **emIoControl - EC\_IOCTL\_ALL\_SLAVES\_MUST\_REACH\_MASTER\_STATE**

#### Parameter

- 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: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

*EC\_E\_NOERROR* or error code

Missing mandatory slaves will be signalized by *emNotify - EC\_NOTIFY\_SLAVE\_PRESENCE*. Slaves who cannot reach the requested master state will be signalized by *emNotify - EC\_NOTIFY\_SLAVE\_UNEXPECTED\_STATE*. *emNotify - EC\_NOTIFY\_NOT\_ALL\_DEVICES\_OPERATIONAL* will not be generated anymore if this IOCTL is called with EC\_FALSE, *emNotify - EC\_NOTIFY\_CYCCMD\_WKC\_ERROR* will be still generated.

## 6.6.33 emGetCfgSlaveInfo

```
static EC_T_DWORD ecatGetCfgSlaveInfo (
    EC_T_BOOL bStationAddress,
    EC_T_WORD wSlaveAddress,
    EC_T_CFG_SLAVE_INFO *pSlaveInfo
)
EC_T_DWORD emGetCfgSlaveInfo (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_CFG_SLAVE_INFO *pSlaveInfo
)
```

Return information about a configured slave from the ENI file.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pSlaveInfo** – [out] Information about the slave.

#### Returns

*EC\_E\_NOERROR* or error code

struct **EC\_T\_CFG\_SLAVE\_INFO**

## Public Members

**EC\_T\_DWORD dwSlaveId**  
[out] Slave's ID to bind bus slave and config slave information

**EC\_T\_CHAR abyDeviceName[ECAT\_DEVICE\_NAMESIZE]**  
[out] Slave's configured name (80 Byte) (from ENI file)

**EC\_T\_DWORD dwHCGroupIdx**  
[out] Index of Hot Connect group, 0 for mandatory

**EC\_T\_BOOL bIsPresent**  
[out] Slave present on bus

**EC\_T\_BOOL bIsHCGroupPresent**  
[out] Slave's Hot Connect group present on bus

**EC\_T\_DWORD dwVendorId**  
[out] Vendor identification (from ENI file)

**EC\_T\_DWORD dwProductCode**  
[out] Product code (from ENI file)

**EC\_T\_DWORD dwRevisionNumber**  
[out] Revision number (from ENI file)

**EC\_T\_DWORD dwSerialNumber**  
[out] Serial number (from ENI file)

**EC\_T\_WORD wStationAddress**  
[out] Slave's configured station address (from ENI file)

**EC\_T\_WORD wAutoIncAddress**  
[out] Slave's auto increment address (may differ from ENI file)

**EC\_T\_DWORD dwPdOffsIn**  
[out] Process input data bit offset (from ENI file)

**EC\_T\_DWORD dwPdSizeIn**  
[out] Process input data bit size (from ENI file)

**EC\_T\_DWORD dwPdOffsOut**  
[out] Process output data bit offset (from ENI file)

**EC\_T\_DWORD dwPdSizeOut**  
[out] Process output data bit size (from ENI file)

**EC\_T\_DWORD dwPdOffsIn2**  
[out] 2nd sync unit process input data bit offset (from ENI file)

**EC\_T\_DWORD dwPdSizeIn2**  
[out] 2nd sync unit process input data bit size (from ENI file)

**EC\_T\_DWORD dwPdOffsOut2**  
[out] 2nd sync unit process output data bit offset (from ENI file)

**EC\_T\_DWORD dwPdSizeOut2**

[out] 2nd sync unit process output data bit size (from ENI file)

**EC\_T\_DWORD dwPdOffsetIn3**

[out] 3rd sync unit process input data bit offset (from ENI file)

**EC\_T\_DWORD dwPdSizeIn3**

[out] 3rd sync unit process input data bit size (from ENI file)

**EC\_T\_DWORD dwPdOffsetOut3**

[out] 3rd sync unit process output data bit offset (from ENI file)

**EC\_T\_DWORD dwPdSizeOut3**

[out] 3rd sync unit process output data bit size (from ENI file)

**EC\_T\_DWORD dwPdOffsetIn4**

[out] 4th sync unit process input data bit offset (from ENI file)

**EC\_T\_DWORD dwPdSizeIn4**

[out] 4th sync unit process input data bit size (from ENI file)

**EC\_T\_DWORD dwPdOffsetOut4**

[out] 4th sync unit process output data bit offset (from ENI file)

**EC\_T\_DWORD dwPdSizeOut4**

[out] 4th sync unit process output data bit size (from ENI file)

**EC\_T\_DWORD dwMbxSupportedProtocols**

[out] Mailbox protocols supported by the slave (from ENI file). Combination of *Supported mailbox protocols* flags

**EC\_T\_DWORD dwMbxOutSize**

[out] Mailbox output byte size (from ENI file)

**EC\_T\_DWORD dwMbxInSize**

[out] Mailbox input byte size (from ENI file)

**EC\_T\_DWORD dwMbxOutSize2**

[out] Bootstrap mailbox output byte size (from ENI file)

**EC\_T\_DWORD dwMbxInSize2**

[out] Bootstrap mailbox input byte size (from ENI file)

**EC\_T\_BOOL bDcSupport**

[out] Slave supports DC (from ENI file)

**EC\_T\_WORD wNumProcessVarsInp**

[out] Number of input process data variables (from ENI file)

**EC\_T\_WORD wNumProcessVarsOutp**

[out] Number of output process data variables (from ENI file)

**EC\_T\_WORD wPrevStationAddress**

[out] Station address of the previous slave (from ENI file)

**EC\_T\_WORD wPrevPort**

[out] Connected port of the previous slave (from ENI file)

**EC\_T\_WORD wIdentifyAdo**

[out] ADO used for identification command (from ENI file)

**EC\_T\_WORD wIdentifyData**

[out] Identification value to be validated (from ENI file)

**EC\_T\_BYTE byPortDescriptor**

[out] Port descriptor (ESC register 0x0007) (from ENI file)

**EC\_T\_WORD wWkcStateDiagOffsIn[EC\_CFG\_SLAVE\_PD\_SECTIONS]**

[out] Offset of WkcState bit in diagnosis image (ENI: ProcessData/Recv[1..4]/BitStart): 0xFFFFFFFF  
= offset not available. WkcState bit values: 0 = Data valid, 1 = Data invalid

**EC\_T\_WORD wWkcStateDiagOffsOut[EC\_CFG\_SLAVE\_PD\_SECTIONS]**

[out] Offset of WkcState bit in diagnosis image (ENI: ProcessData/Send[1..4]/BitStart): 0xFFFFFFFF  
= offset not available. WkcState bit values: 0 = Data valid, 1 = Data invalid

**EC\_T\_WORD awMasterSyncUnitIn[EC\_CFG\_SLAVE\_PD\_SECTIONS]**

[out] Sync Unit (ENI: ProcessData/TxPdo[1..4]@Su)

**EC\_T\_WORD awMasterSyncUnitOut[EC\_CFG\_SLAVE\_PD\_SECTIONS]**

[out] Sync Unit (ENI: ProcessData/RxPdo[1..4]@Su)

**EC\_T\_BOOL bDisabled**

[out] Slave disabled by API (emSetSlaveDisabled / emSetSlavesDisabled).

**EC\_T\_BOOL bDisconnected**

[out] Slave disconnected by API (emSetSlaveDisconnected / emSetSlavesDisconnected).

**EC\_T\_BOOL bExtended**

[out] Slave generated by emConfigExtend

**EC\_T\_BOOL bDcReferenceClock**

[out] Slave is reference clock (from ENI file)

**EC\_T\_BOOL bDcPotentialRefClock**

[out] Slave can be used as a reference clock (from ENI file)

**EC\_T\_DWORD dwDcCycleTime0**

[out] Cycle time of Sync0 event in ns (from ENI file)

**EC\_T\_DWORD dwDcCycleTime1**

[out] Calculated value dwDcCycleTime1 [ns] = Cycle time of Sync1 event - Cycle time of Sync0 event  
+ Shift time of Sync0 event (from ENI file)

**EC\_T\_INT nDcShiftTime**

[out] Shift time of Sync0 event in ns (from ENI file)

**Supported mailbox protocols flags**

**EC\_MBX\_PROTOCOL\_AOE**

**EC\_MBX\_PROTOCOL\_EOE**

**EC\_MBX\_PROTOCOL\_COE**  
**EC\_MBX\_PROTOCOL\_FOE**  
**EC\_MBX\_PROTOCOL\_SOE**  
**EC\_MBX\_PROTOCOL\_VOE**

### emGetCfgSlaveInfo() Example

```
/* get information about slave configured in ENI file */
EC_T_CFG_SLAVE_INFO oSlaveInfo;
OsMemset(&oSlaveInfo, 0, sizeof(EC_T_CFG_SLAVE_INFO));
dwRes = emGetCfgSlaveInfo(dwInstanceId, EC_TRUE, 1001, &oSlaveInfo);
```

### 6.6.34 emGetCfgSlaveEoeInfo

```
static EC_T_DWORD ecatGetCfgSlaveEoeInfo (
    EC_T_BOOL bStationAddress,
    EC_T_WORD wSlaveAddress,
    EC_T_CFG_SLAVE_EOE_INFO *pSlaveEoeInfo
)
EC_T_DWORD emGetCfgSlaveEoeInfo (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_CFG_SLAVE_EOE_INFO *pSlaveEoeInfo
)
```

Return EoE information about a configured slave from the ENI file.

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pSlaveEoeInfo** – [out] Information about the slave

#### Returns

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master isn't initialized
- **EC\_E\_INVALIDPARM** if dwInstanceId is out of range
- **EC\_E\_NOTFOUND** if no slave matching bFixedAddressing / wSlaveAddress can be found
- **EC\_E\_NO\_MBX\_SUPPORT** if the slave does not support mailbox communication
- **EC\_E\_NO\_EOE\_SUPPORT** if the slave supports mailbox communication, but not EoE

struct **EC\_T\_CFG\_SLAVE\_EOE\_INFO**

**Public Members**

**EC\_T\_DWORD dwSlaveId**  
 [out] Slave ID

**EC\_T\_BOOL bMacAddr**  
 [out] Indicates whether the MAC address could be read and is valid

**EC\_T\_BYTEx abyMacAddr[6]**  
 [out] MAC address

**EC\_T\_BOOL bIpAddr**  
 [out] Indicates whether the IP address could be read and is valid

**EC\_T\_IPADDR oIpAddr**  
 [out] IP address

**EC\_T\_BOOL bSubnetMask**  
 [out] Indicates whether the subnet mask could be read and is valid

**EC\_T\_IPADDR oSubnetMask**  
 [out] Subnet mask

**EC\_T\_BOOL bDefaultGateway**  
 [out] Indicates whether the default gateway could be read and is valid

**EC\_T\_IPADDR oDefaultGateway**  
 [out] Default gateway

**EC\_T\_BOOL bDnsServer**  
 [out] Indicates whether the DNS server could be read and is valid

**EC\_T\_IPADDR oDnsServer**  
 [out] DNS server

**EC\_T\_BOOL bDnsName**  
 [out] Indicates whether the DNS name could be read and is valid

**EC\_T\_CHAR szDnsName[32]**  
 [out] DNS name

**emGetCfgSlaveEoeInfo() Example**

```
/* get EoE information about slave configured in ENI file */
EC_T_CFG_SLAVE_EOE_INFO oSlaveInfo;
OsMemset(&oSlaveInfo, 0, sizeof(EC_T_CFG_SLAVE_EOE_INFO));
dwRes = emGetCfgSlaveEoeInfo(dwInstanceId, EC_TRUE, 1001, &oSlaveInfo);
```

### 6.6.35 emGetCfgSlaveSmInfo

```
static EC_T_DWORD ecatGetCfgSlaveSmInfo (
    EC_T_BOOL bStationAddress,
    EC_T_WORD wSlaveAddress,
    EC_T_CFG_SLAVE_SM_INFO *pSlaveSmInfo
)
EC_T_DWORD emGetCfgSlaveSmInfo (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_CFG_SLAVE_SM_INFO *pSlaveSmInfo
)
) Return information about Sync Master of a configured slave from the ENI file.
```

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pSlaveSmInfo** – [out] Information about the slave.

#### Returns

*EC\_E\_NOERROR* or error code

struct **EC\_T\_CFG\_SLAVE\_SM\_ENTRY**

#### Public Members

**EC\_T\_WORD wPhysAddr**  
[out] ESC (0x800 + y \* 8)

**EC\_T\_WORD wLength**  
[out] ESC (0x802 + y \* 8)

**EC\_T\_BYTEx byOpMode**  
[out] Bits 0..1 ESC (0x804 + y \* 8)

**EC\_T\_BYTEx byDirection**  
[out] Bits 2..3 ESC (0x804 + y \* 8)

**EC\_T\_DWORD dwPdBitOffs**  
[out] Process input data bit offset (from ENI file)

**EC\_T\_DWORD dwPdBitSize**  
[out] Process input data bit size (from ENI file)

**EC\_T\_WORD wWkcStateDiagBitOffs**  
[out] Offset of WkcState bit in diagnosis image

**EC\_T\_WORD wMasterSyncUnit**  
[out] Sync Unit (ENI: ProcessData/TxPdo[1..4]@Su)

struct **EC\_T\_CFG\_SLAVE\_SM\_INFO**

#### Public Members

**EC\_T\_DWORD dwSlaveId**

[out] Slave ID

**EC\_T\_DWORD dwSmInfoNumOf**

[out] Number of available sync managers

**EC\_T\_CFG\_SLAVE\_SM\_ENTRY aoSmInfos[ECREG\_SYNCMANAGER\_MAX\_NUMOF]**

[out] Sync managers info

#### **emGetCfgSlaveSmInfo() Example**

```
/* get information about slave's sync managers configured in ENI file */
EC_T_CFG_SLAVE_SM_INFO oSlaveSmInfo;
OsMemset(&oSlaveSmInfo, 0, sizeof(EC_T_CFG_SLAVE_SM_INFO));
dwRes = emGetCfgSlaveSmInfo(dwInstanceId, EC_TRUE, 1001, &oSlaveSmInfo);
```

### 6.6.36 **emGetBusSlaveInfo**

```
static EC_T_DWORD ecatGetBusSlaveInfo (
    EC_T_BOOL bStationAddress,
    EC_T_WORD wSlaveAddress,
    EC_T_BUS_SLAVE_INFO *pSlaveInfo
)
EC_T_DWORD emGetBusSlaveInfo (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BUS_SLAVE_INFO *pSlaveInfo
)
```

Return information about a slave connected to the EtherCAT bus.

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pSlaveInfo** – [out] Information from the slave.

#### Returns

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master isn't initialized
- **EC\_E\_INVALIDPARAM** if dwInstanceId is out of range
- **EC\_E\_NOTFOUND** if no slave matching bFixedAddressing / wSlaveAddress can be found

---

```
struct EC_T_BUS_SLAVE_INFO
```

### Public Members

#### **EC\_T\_DWORD dwSlaveId**

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

#### **EC\_T\_DWORD adwPortSlaveIds[ESC\_PORT\_COUNT]**

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

#### **EC\_T\_WORD wPortState**

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

wwww : Signal detected 1=yes, 0=no

xxxx : Loop closed 1=yes, 0=no

yyyy : Link established 1=yes, 0=no

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

#### **EC\_T\_WORD wAutoIncAddress**

[out] The slave's auto increment address

#### **EC\_T\_BOOL bDcSupport**

[out] Slave supports DC (Bus Topology Scan)

#### **EC\_T\_BOOL bDc64Support**

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

#### **EC\_T\_DWORD dwVendorId**

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

#### **EC\_T\_DWORD dwProductCode**

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

#### **EC\_T\_DWORD dwRevisionNumber**

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

#### **EC\_T\_DWORD dwSerialNumber**

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

#### **EC\_T\_BYTExbyESCType**

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

#### **EC\_T\_BYTExbyESCRevision**

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

#### **EC\_T\_WORD wESCBuild**

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

#### **EC\_T\_BYTExbyPortDescriptor**

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

**EC\_T\_WORD wFeaturesSupported**

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

**EC\_T\_WORD wStationAddress**

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

**EC\_T\_WORD wAliasAddress**

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

**EC\_T\_WORD wALStatus**

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

**EC\_T\_WORD wALStatusCode**

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

**EC\_T\_DWORD dwSystemTimeDifference**

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

**EC\_T\_WORD wMbxSupportedProtocols**

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

**EC\_T\_WORD wDLStatus**

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

**EC\_T\_WORD wPrevPort**

[out] Connected port of the previous slave

**EC\_T\_WORD wIdentifyData**

[out] Last read identification value see *EC\_T\_CFG\_SLAVE\_INFO.wIdentifyAdo*

**EC\_T\_BOOL bLineCrossed**

[out] Line crossed was detected at this slave

**EC\_T\_DWORD dwSlaveDelay**

[out] Delay behind slave [ns]. This value is only valid if a DC configuration is used

**EC\_T\_DWORD dwPropagDelay**

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

**EC\_T\_BOOL bIsRefClock**

[out] Slave is reference clock

**EC\_T\_BOOL bIsDeviceEmulation**

[out] Slave without Firmware. ESC register 0x0141, enabled by EEPROM offset 0x0000.8.

**EC\_T\_WORD wLineCrossedFlags**

[out] Combination of *Line crossed flags*

**EC\_T\_DWORD dwCyclicWkcErrorCnt**

[out] Counter for Cyclic WC Error

**EC\_T\_DWORD dwSlaveAbsentCnt**

[out] Counter for Absent/Not Present Slaves

**EC\_T\_DWORD dwUnexpectedStateCnt**  
 [out] Counter for Abnormal State Change

#### Port Slave ID's

```
MASTER_SLAVE_ID
SIMULATOR_SLAVE_ID
MASTER_RED_SLAVE_ID
EL9010_SLAVE_ID
FRAMELOSS_SLAVE_ID
JUNCTION_RED_FLAG
EC_LINECROSSED_flags

EC_LINECROSSED_NOT_CONNECTED_PORTA
EC_LINECROSSED_UNEXPECTED_INPUT_PORT
EC_LINECROSSED_UNEXPECTED_JUNCTION_RED
EC_LINECROSSED_UNRESOLVED_PORT_CONNECTION
EC_LINECROSSED_HIDDEN_SLAVE_CONNECTED
EC_LINECROSSED_PHYSIC_MISMATCH
EC_LINECROSSED_INVALID_PORT_CONNECTION
```

#### emGetBusSlaveInfo() Example

```
/* get information about slave connected to EtherCAT bus */
EC_T_BUS_SLAVE_INFO oSlaveInfo;
OsMemset(&oSlaveInfo, 0, sizeof(EC_T_BUS_SLAVE_INFO));
dwRes = emGetBusSlaveInfo(dwInstanceId, EC_TRUE, 1001, &oSlaveInfo);
```

### 6.6.37 emReadSlaveIdentification

```
static EC_T_DWORD ecatReadSlaveIdentification(
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wAdo,
    EC_T_WORD *pwValue,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emReadSlaveIdentification(
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wAdo,
    EC_T_WORD *pwValue,
    EC_T_DWORD dwTimeout
)
```

Read identification value from slave.

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

## Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wAdo** – [in] ADO used for identification command
- **pwValue** – [out] Pointer to Word value containing the Identification value
- **dwTimeout** – [in] Timeout [ms]

## Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC\_NOWAIT
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC\_E\_TIMEOUT* if dwTimeout elapsed during the API call
- *EC\_E\_BUSY* another transfer request is already pending or the master or the corresponding slave is currently changing its operational state
- *EC\_E\_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC\_E\_ADO\_NOT\_SUPPORTED* if the slave does not support requesting ID mechanism
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive

## emReadSlaveIdentification() Example

```
/* get identification value from slave */
EC_T_WORD wValue = 0;
dwRes = emReadSlaveIdentification(dwInstanceId, EC_TRUE, 1001,
    0x0134 /* explicit device ID */, &wValue, 5000 /* timeout */);
```

## 6.6.38 emReadSlaveIdentificationReq

```
static EC_T_DWORD ecatReadSlaveIdentificationReq(
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wAdo,
    EC_T_WORD *pwValue,
    EC_T_DWORD dwTimeout
)
```

```
EC_T_DWORD emReadSlaveIdentificationReq(
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wAdo,
    EC_T_WORD *pwValue,
    EC_T_DWORD dwTimeout
)
```

) Request the identification value from a slave and returns immediately.

A notification EC\_NOTIFY\_SLAVE\_IDENTIFICATION is given on completion or timeout. This function may be called from within the JobTask's context.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTferId** – [in] Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within [EC\\_T\\_SLAVE\\_IDENTIFICATION\\_NTFY\\_DESC](#)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wAdo** – [in] ADO used for identification command
- **pwValue** – [out] Pointer to Word value containing the Identification value, must be valid until the request complete.
- **dwTimeout** – [in] Timeout [ms]

#### Returns

- [EC\\_E\\_NOERROR](#) if successful
- [EC\\_E\\_INVALIDSTATE](#) if master isn't initialized
- [EC\\_E\\_INVALIDPARM](#) if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC\_NOWAIT
- [EC\\_E\\_SLAVE\\_NOT\\_PRESENT](#) if slave not present
- [EC\\_E\\_NOTFOUND](#) if no slave matching bFixedAddressing / wSlaveAddress can be found
- [EC\\_E\\_ADO\\_NOT\\_SUPPORTED](#) if the slave does not support requesting ID mechanism

**emReadSlaveIdentificationReq() Example**

```
/* get identification value from slave (non-blocking) */
EC_T_WORD wValue = 0;
EC_T_DWORD dwTferId = 1234; /* arbitrary unique ID from application */
dwRes = emReadSlaveIdentificationReq(dwInstanceId, dwClientId,
    dwTferId, EC_TRUE, 1001, 0x0134 /* explicit device ID */, &wValue, 5000 /*
→timeout */);
```

**See also:***emNotify - EC\_NOTIFY\_SLAVE\_IDENTIFICATION***6.6.39 emNotify - EC\_NOTIFY\_SLAVE\_IDENTIFICATION**

This notification is given, when the read slave identification request is completed.

**emNotify - EC\_NOTIFY\_SLAVE\_IDENTIFICATION****Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_SLAVE\_IDENTIFICATION\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**struct EC\_T\_SLAVE\_IDENTIFICATION\_NTFY\_DESC****Public Members****EC\_T\_DWORD dwTferId**

Transfer ID. For every new port operation a unique ID has to be assigned. This ID can be used after completion to identify the transfer

**EC\_T\_DWORD dwResult**

Result of request

***EC\_T\_SLAVE\_PROP* SlaveProp**

Slave properties

**EC\_T\_WORD wAdo**

Slave address offset used for identification. Given by API

**EC\_T\_WORD wValue**

Slave identification value. Given by API

## 6.6.40 emIoControl - EC\_IOCTL\_SET\_AUTO\_ACK\_AL\_STATUS\_ERROR\_ENABLED

Specifies if slave errors must be automatically acknowledged

### emIoControl - EC\_IOCTL\_SET\_AUTO\_ACK\_AL\_STATUS\_ERROR\_ENABLED

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_BOOL variable. If set to EC\_TRUE slave errors must be automatically acknowledged, if set to EC\_FALSE the application must acknowledge slave errors explicitly
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

EC\_E\_NOERROR or error code

The pending slave error will be acknowledged during the next `emSetSlaveState()` call.

## 6.6.41 emIoControl - EC\_IOCTL\_SET\_AUTO\_ADJUST\_CYCCMD\_WKC\_ENABLED

Specifies if the cyclic commands expected WKC must be automatically adjusted according the state and the presence of the slaves.

### emIoControl - EC\_IOCTL\_SET\_AUTO\_ADJUST\_CYCCMD\_WKC\_ENABLED

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_BOOL variable. If set to EC\_TRUE cyclic commands expected WKC must be automatically adjusted, if set to EC\_FALSE the cyclic commands expected WKC stay unchanged
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

EC\_E\_NOERROR or error code

If TRUE, the notification `emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR` is only generated if a slave doesn't increment the WKC although it should. AUTO\_ADJUST\_CYCCMD\_WKC is disabled by default.

#### See also:

- *Cyclic cmd WKC validation*
- *Working Counter (WKC) State in Diagnosis Image*

## 6.6.42 emSetSlaveDisabled

Before using this function, please check if the following patents has to be taken into consideration for your application and use case:

- JP2014146077: CONTROL DEVICE AND OPERATION METHOD FOR CONTROL DEVICE
- JP2014146070: CONTROL DEVICE, CONTROL METHOD, AND PROGRAM
- JP2014120884: INFORMATION PROCESSING APPARATUS, INFORMATION ROCESSING PROGRAM, AND INFORMATION PROCESSING METHOD

```
static EC_T_DWORD ecatSetSlaveDisabled(
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL bDisabled
)
EC_T_DWORD emSetSlaveDisabled(
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL bDisabled
)
Enable or disable a specific slave.
```

The EtherCAT state of disabled slaves can not be set higher than PREOP. If the state is higher than PREOP at the time this function is called. The state will be automatically change to PREOP. The information about the last requested state is lost and is set to PREOP too.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **bDisabled** – [in] EC\_TRUE: Disable slave, EC\_FALSE: Enable slave

### Returns

*EC\_E\_NOERROR* or error code

### emSetSlaveDisabled() Example

```
/* enable or disable specific slave */
dwRes = emSetSlaveDisabled(dwInstanceId, EC_TRUE, 1002, EC_TRUE /* disabled */);
```

## 6.6.43 emIoControl - EC\_IOCTL\_SET\_SLAVE\_MAX\_STATE

Specifies maximum state for specific slave.

### emIoControl - EC\_IOCTL\_SET\_SLAVE\_MAX\_STATE

#### Parameter

- **pbyInBuf**: [in] Pointer to EC\_T\_SLAVE\_MAX\_STATE\_DESC. Specifies maximum state for the specific slave.
- **dwInBufSize**: [in] Size of the input buffer provided at pbyInBuf in bytes.

- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return***EC\_E\_NOERROR or error code*

### 6.6.44 emSetSlaveDisconnected

Before using this function, please check if the following patents has to be taken into consideration for your application and use case:

- JP2014146077: CONTROL DEVICE AND OPERATION METHOD FOR CONTROL DEVICE
- JP2014146070: CONTROL DEVICE, CONTROL METHOD, AND PROGRAM
- JP2014120884: INFORMATION PROCESSING APPARATUS, INFORMATION ROCESSING PROGRAM, AND INFORMATION PROCESSING METHOD

```
static EC_T_DWORD ecatSetSlaveDisconnected(
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL bDisconnected
)
EC_T_DWORD emSetSlaveDisconnected(
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_BOOL bDisconnected
)
```

Mark specific slave for connection or disconnection.

The EtherCAT state of disconnected slaves can not be set higher than INIT. If the state is higher than INIT at the time this function is called, the state will be automatically change to INIT. The information about the last requested state is lost and is set to INIT too.

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **bDisconnected** – [in] EC\_TRUE: Mark slave for disconnection, EC\_FALSE: Mark slave for (re-)connection

**Returns***EC\_E\_NOERROR or error code*

### **emSetSlaveDisconnected() Example**

```
/*Connect or disconnect specific slave*/
dwRes = emSetSlaveDisconnected(dwInstanceId, EC_TRUE, 1002,
    EC_TRUE /* disconnected */);
```

### **6.6.45 emSetSlavesDisconnected**

Before using this function, please check if the following patents has to be taken into consideration for your application and use case:

- JP2014146077: CONTROL DEVICE AND OPERATION METHOD FOR CONTROL DEVICE
- JP2014146070: CONTROL DEVICE, CONTROL METHOD, AND PROGRAM
- JP2014120884: INFORMATION PROCESSING APPARATUS, INFORMATION ROCESSING PROGRAM, AND INFORMATION PROCESSING METHOD

```
static EC_T_DWORD ecatSetSlavesDisconnected(
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_SLAVE_SELECTION eSlaveSelection,
    EC_T_BOOL bDisconnected
)
EC_T_DWORD emSetSlavesDisconnected (
    EC_T_DWORD dwInstanceId,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_SLAVE_SELECTION eSlaveSelection,
    EC_T_BOOL bDisconnected
)
Mark a specific group of slaves for connection or disconnection.
```

#### **Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **eSlaveSelection** – [in] Slave selection criteria
- **bDisconnected** – [in] EC\_TRUE: mark slaves for disconnection, EC\_FALSE: mark slaves for connection

#### **Returns**

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master isn't initialized
- **EC\_E\_INVALIDPARM** if dwInstanceId is out of range
- **EC\_E\_NOTFOUND** if no slave matching bFixedAddressing / wSlaveAddress can be found

**emSetSlavesDisconnected() Example**

```
/* set specific group of slaves for connection or disconnection */
dwRes = emSetSlavesDisconnected(dwInstanceId, EC_TRUE, 1001,
    eSlaveSelectionTopoFollowers, EC_TRUE /* disconnected*/);
```

**See also:***emSetSlaveDisconnected()***6.6.46 emGetSlavePortState**

```
static EC_T_DWORD ecatGetSlavePortState (
    EC_T_DWORD dwSlaveId,
    EC_T_WORD *pwPortState
)
EC_T_DWORD emGetSlavePortState (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD *pwPortState
)
```

Returns the state of the slave ports.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **pwPortState** – [out] Slave port state.

Format: wwww xxxx yyyy zzzz (each nibble : port 3210)

wwww : Signal detected 1=yes, 0=no (ESC Register 0x110 Bit 9, 11, 13, 15)

xxxx : Loop closed 1=yes, 0=no (ESC Register 0x110 Bit 8, 10, 12, 14)

yyyy : Link established 1=yes, 0=no (ESC Register 0x110 Bit 4, 5, 6, 7)

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

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceId is out of range or the output pointer is EC\_NULL
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found

**emGetSlavePortState() Example**

```
EC_T_DWORD dwSlaveId = 2;
EC_T_WORD wPortState = 0;
dwRes = emGetSlavePortState(dwInstanceId, dwSlaveId, &wPortState);
```

**See also:**[emGetSlaveId\(\)](#)**6.6.47 emSetSlavePortState**

```
static EC_T_DWORD ecatSetSlavePortState(
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wPort,
    EC_T_BOOL bClose,
    EC_T_BOOL bForce,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emSetSlavePortState(
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wPort,
    EC_T_BOOL bClose,
    EC_T_BOOL bForce,
    EC_T_DWORD dwTimeout
)
```

Open or close slave port.

This function allows to open or close a specific slave port in different ways. It also can be used to re-open ports closed by a rescue scan.

**Parameters**

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wPort** – [in] Port to open or close. Can be ESC\_PORT\_A, ESC\_PORT\_B, ESC\_PORT\_C, ESC\_PORT\_D
- **bClose** – [in] EC\_TRUE: close port, EC\_FALSE: open port
- **bForce** – [in] EC\_TRUE: port will be closed or open, EC\_FALSE: port will be set in AutoClose mode
- **dwTimeout** – [in] Timeout [ms]

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceId is out of range
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive

### **emSetSlavePortState() Example**

```
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1001);
dwRes = emSetSlavePortState(dwInstanceId, dwSlaveId, ESC_PORT_B,
    EC_TRUE /* close */ , EC_TRUE /* force */ , 5000 /* timeout */ );
```

#### See also:

- *emRescueScan()*
- *emGetSlaveId()*

### **6.6.48 emSetSlavePortStateReq**

```
static EC_T_DWORD ecatSetSlavePortStateReq(
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wPort,
    EC_T_BOOL bClose,
    EC_T_BOOL bForce,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emSetSlavePortStateReq(
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwClientId,
    EC_T_DWORD dwTferId,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wPort,
    EC_T_BOOL bClose,
    EC_T_BOOL bForce,
    EC_T_DWORD dwTimeout
)
```

Requests Open or close slave port operation and returns immediately.

A *EC\_T\_PORT\_OPERATION\_NTFY\_DESC* is given on completion. This function can be called to re-open ports closed by a rescue scan.

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClientId** – [in] Client ID returned by RegisterClient (0 if all registered clients shall be notified).
- **dwTferId** – [in] Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within *EC\_T\_PORT\_OPERATION\_NTFY\_DESC*
- **dwSlaveId** – [in] Slave ID
- **wPort** – [in] Port to open or close. Can be ESC\_PORT\_A, ESC\_PORT\_B, ESC\_PORT\_C, ESC\_PORT\_D
- **bClose** – [in] EC\_TRUE: close port, EC\_FALSE: open port
- **bForce** – [in] EC\_TRUE: port will be closed or open, EC\_FALSE: port will be set in AutoClose mode
- **dwTimeout** – [in] Timeout [ms]

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found

### **emSetSlavePortStateReq() Example**

```
EC_T_DWORD dwTferId = 1234; /* arbitrary unique ID from application */
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1001);
dwRes = emSetSlavePortStateReq(dwInstanceId, dwTferId, dwClientId,
    dwSlaveId, ESC_PORT_B, EC_TRUE /* close */, EC_TRUE /* force */, 5000 /*
    ↪timeout */);
```

#### See also:

- *emNotify - EC\_NOTIFY\_PORT\_OPERATION*
- *emRescueScan()*
- *emGetSlaveId()*

### **6.6.49 emNotify - EC\_NOTIFY\_PORT\_OPERATION**

This notification is given, when the port operation request is completed.

#### **emNotify - EC\_NOTIFY\_PORT\_OPERATION**

##### **Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_PORT\_OPERATION\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

struct **EC\_T\_PORT\_OPERATION\_NTFY\_DESC**

##### **Public Members**

###### **EC\_T\_DWORD dwTferId**

Transfer ID. For every new port operation a unique ID has to be assigned. This ID can be used after completion to identify the transfer

###### **EC\_T\_DWORD dwResult**

Result of request

###### **EC\_T\_SLAVE\_PROP SlaveProp**

Slave properties

**EC\_T\_WORD wPortStateOld**  
Old state of the slave ports

**EC\_T\_WORD wPortStateNew**  
New state of the slave ports

#### See also:

`emGetSlavePortState ()`

## 6.6.50 emIoControl - EC\_IOCTL\_SET\_NEW\_BUSSLAVES\_TO\_INIT

Force state change to INIT for all new slaves in network after detection.

### emIoControl - EC\_IOCTL\_SET\_NEW\_BUSSLAVES\_TO\_INIT

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_BOOL. EC\_TRUE: Force state change, EC\_FALSE: No state change.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

EC\_E\_NOERROR or error code

Default: No state change after detection

## 6.7 Diagnosis, error detection, error notifications

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 [emNotify - 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:

1. Slave unrelated errors
2. 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 [emNotify - 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 [emNotify - EC\\_NOTIFY\\_CYCCMD\\_WKC\\_ERROR](#) error notification will be generated.

#### 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 [emNotify - 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 `emNotify - EC_NOTIFY_SLAVE_PRESENCE`.

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

#### **Master calls `emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR`**

- Application gets informed
- WKC State in Diagnosis Image changes

#### **See also:**

*Working Counter (WKC) State in Diagnosis Image*

#### **Use cases**

##### **1. Slave is disconnected or powered off:**

- Master detects a DL status event interrupt and performs a bus scan.
- Master calls `emNotify - EC_NOTIFY_SLAVE_PRESENCE`
- Application gets informed and could set the whole master into a lower state, e.g. `eEcat-State_INIT`

##### **2. Slave state is not OPERATIONAL anymore**

- Master calls `emNotify - EC_NOTIFY_SLAVE_UNEXPECTED_STATE`
- Application gets informed and could either set the whole master into lower state (e.g. `eEcat-State_PREOP`), or calls `emSetSlaveState()` to repair the failed slave.

##### **3. Slave is re-connected or powered on:**

- Master detects a DL status event interrupt and performs a bus scan.
- Master calls `emNotify - EC_NOTIFY_SLAVE_PRESENCE`.
- Application could wait until all slaves are re-connected by calling the functions `emGetNumConnectedSlaves()` and `emGetNumConfiguredSlaves()`.
- After all slaves are re-connected the application could either set the whole master to `eEcat-State_INIT` and afterwards to `:cpp:enumerator:`eEcatState_OP``, or the application uses `emSetSlaveState()` to repair only the failed slaves.

### **6.7.1 `emSetLogParms`**

```
static EC_T_DWORD ecatSetLogParms (EC_T_LOG_PARMS *pLogParms)
```

```
EC_T_DWORD emSetLogParms (EC_T_DWORD dwInstanceID, EC_T_LOG_PARMS *pLogParms)
```

Sets log parameters. Used to change the parameters provided by `emInitMaster()`.

#### **Parameters**

**pLogParms** – [in] New Log parameters

## 6.7.2 emEthDbgMsg

```
static EC_T_DWORD ecatEthDbgMsg (
    EC_T_BYT byEthTypeByte0,
    EC_T_BYT byEthTypeByte1,
    EC_T_CHAR *szMsg
)
EC_T_DWORD emEthDbgMsg (
    EC_T_DWORD dwInstanceID,
    EC_T_BYT byEthTypeByte0,
    EC_T_BYT byEthTypeByte1,
    EC_T_CHAR *szMsg
)
Send a debug message to the EtherCAT Link Layer.
```

This feature can be used for debugging purposes.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **byEthTypeByte0** – [in] Ethernet type byte 0
- **byEthTypeByte1** – [in] Ethernet type byte 1
- **szMsg** – [in] Message to send to link layer

### Returns

*EC\_E\_NOERROR* or error code

## emEthDbgMsg() Example

```
EC_T_CHAR* szMsg = (EC_T_CHAR*) "Hello World";
EC_T_BYT byEthTypeByte0 = 6;
EC_T_BYT byEthTypeByte1 = 6;
/* send message as frame, validate given EtherType:
   >= 1536: EtherType (supported as parameter),
   <= 1500: size of payload (not supported as parameter),
   1501-1535: undefined */
/* send debug message to EtherCAT Link Layer */
dwRes = emEthDbgMsg(dwInstanceId, byEthTypeByte0, byEthTypeByte1, szMsg);
```

## 6.7.3 emIoControl - EC\_IOCTL\_GET\_SLVSTATISTICS

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

### emIoControl - EC\_IOCTL\_GET\_SLVSTATISTICS

#### Parameter

- **pbyInBuf:** [in] Pointer to a EC\_T\_DWORD type variable containing the slave id.
- **dwInBufSize:** [in] Size of the input buffer provided at pbyInBuf in bytes.
- **pbyOutBuf:** [out] Pointer to struct EC\_T\_SLVSTATISTICS\_DESC
- **dwOutBufSize:** [in] Size of the output buffer provided at pbyOutBuf in bytes.
- **pdwNumOutData:** [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer.

**Return**

EC\_E\_NOERROR or error code

struct **EC\_T\_SLVSTATISTICS\_DESC****Public Members**EC\_T\_BYTE **abyInvalidFrameCnt**[ESC\_PORT\_COUNT]  
[out] Invalid Frame Counters per Slave PortEC\_T\_BYTE **abyRxErrorCnt**[ESC\_PORT\_COUNT]  
[out] RX Error Counters per Slave PortEC\_T\_BYTE **abyFwdRxErrorCnt**[ESC\_PORT\_COUNT]  
[out] Forwarded RX Error Counters per Slave PortEC\_T\_BYTE **byProcessingUnitErrorCnt**  
[out] Processing Unit Error CounterEC\_T\_BYTE **byPdiErrorCnt**  
[out] PDI Error CounterEC\_T\_WORD **wAlStatusCode**  
[out] AL Status CodeEC\_T\_BYTE **abyLostLinkCnt**[ESC\_PORT\_COUNT]  
[out] Lost Link Counters per Slave PortEC\_T\_UINT64 **qwReadTime**  
[out] Timestamp of the last read [ns]EC\_T\_UINT64 **qwChangeTime**  
[out] Timestamp of the last counter change [ns]**See also:**

- *emIoControl - EC\_IOCTL\_SET\_SLVSTAT\_PERIOD*
- *emIoControl - EC\_IOCTL\_CLR\_SLVSTATISTICS*

**6.7.4 emGetSlaveStatistics**

```
static EC_T_DWORD ecatGetSlaveStatistics (
    EC_T_DWORD dwSlaveId,
    EC_T_SLVSTATISTICS_DESC *pSlaveStatisticsDesc
)
EC_T_DWORD emGetSlaveStatistics (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_SLVSTATISTICS_DESC *pSlaveStatisticsDesc
)
    Get Slave's statistics counter.
```

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave id
- **pSlaveStatisticsDesc** – [out] Pointer to structure *EC\_T\_SLVSTATISTICS\_DESC*

**Returns***EC\_E\_NOERROR* or error code**emGetSlaveStatistics() Example**

```
/* get slave's statistics counters */
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1002);
EC_T_SLVSTATISTICS_DESC oSlaveStatisticsDesc;
OsMemset(&oSlaveStatisticsDesc, 0, sizeof(EC_T_SLVSTATISTICS_DESC));
dwRes = dwSlaveId;
dwRes = emGetSlaveStatistics(dwInstanceId, dwSlaveId, &oSlaveStatisticsDesc);
```

**See also:**

- *emIoControl - EC\_IOCTL\_GET\_SLVSTATISTICS*
- *emGetSlaveId()*

**6.7.5 emIoControl - EC\_IOCTL\_CLR\_SLVSTATISTICS**

Clear all error registers in all slaves

**emIoControl - EC\_IOCTL\_CLR\_SLVSTATISTICS****Parameter**

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return***EC\_E\_NOERROR* or error code**6.7.6 emClearSlaveStatistics**static EC\_T\_DWORD **ecatClearSlaveStatistics** (EC\_T\_DWORD dwSlaveId)

```
EC_T_DWORD emClearSlaveStatistics (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwSlaveId
)
) Clears all error registers of a slave.
```

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave Id, INVALID\_SLAVE\_ID clears all slaves

**Returns***EC\_E\_NOERROR or error code***emClearSlaveStatistics() Example**

```
/* clear all error counters at slave */
EC_T_DWORD dwSlaveId = emGetSlaveId(dwInstanceId, 1002);
dwRes = emClearSlaveStatistics(dwInstanceId, dwSlaveId);
```

**See also:***emGetSlaveId()***6.7.7 emIoControl - EC\_IOCTL\_GET\_SLVSTAT\_PERIOD**

Get Slave Statistics collection period. Period of 0: automatic slave statistics collection disabled.

**emIoControl - EC\_IOCTL\_GET\_SLVSTAT\_PERIOD****Parameter**

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Pointer to a EC\_T\_DWORD type variable containing the slave statistics collection period [ms] to get.
- dwOutBufSize: [in] Size of the output buffer provided at pbyOutBuf in bytes.
- pdwNumOutData: [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer.

**Return***EC\_E\_NOERROR or error code***6.7.8 emIoControl - EC\_IOCTL\_SET\_SLVSTAT\_PERIOD**

Update Slave Statistics collection period. It implicitly forces an immediate collection of slave statistics if performed successfully.

A Period of 0 disables automatic slave statistics collection, otherwise it set the time between the read sequences.

**emIoControl - EC\_IOCTL\_SET\_SLVSTAT\_PERIOD****Parameter**

- 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: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return***EC\_E\_NOERROR or error code*

## 6.7.9 emIoControl - EC\_IOCTL\_FORCE\_SLVSTAT\_COLLECTION

Sends datagrams to collect slave statistics counters.

### emIoControl - EC\_IOCTL\_FORCE\_SLVSTAT\_COLLECTION

#### Parameter

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

EC\_E\_NOERROR or error code

## 6.7.10 emIoControl - EC\_IOCTL\_CLEAR\_MASTER\_INFO\_COUNTERS

Reset Master Info Counters according to given bit masks

### emIoControl - EC\_IOCTL\_CLEAR\_MASTER\_INFO\_COUNTERS

#### Parameter

- pbyInBuf: [in] Pointer to value of EC\_T\_CLEAR\_MASTER\_INFO\_COUNTERS\_PARMS.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

EC\_E\_NOERROR or error code

struct **EC\_T\_CLEAR\_MASTER\_INFO\_COUNTERS\_PARMS**

#### Public Members

##### **EC\_T\_DWORD dwClearBusDiagnosisCounters**

[in] Bit 0..7: Clear corresponding Counter ID:

- Bit 0: Clear all Counters
- Bit 1: Clear Tx Frame Counter
- Bit 2: Clear Rx Frame Counter
- Bit 3: Clear Lost Frame Counter
- Bit 4: Clear Cyclic Frame Counter
- Bit 5: Clear Cyclic Datagram Counter
- Bit 6: Clear Acyclic Frame Counter

- Bit 7: Clear Acyclic DataGram Counter
- Bit 8: Clear Cyclic Lost Frame Counter
- Bit 9: Clear Acyclic Lost Frame Counter

**EC\_T\_UINT64 `qwMailboxStatisticsClearCounters`**

[in] Bit 0..56: Clear corresponding Counter ID.

- Bit 0..7: Clear AoE statistics
  - Bit 0: Total Read Transfer Count
  - Bit 1: Read Transfer Count Last Second
  - Bit 2: Total Bytes Read
  - Bit 3: Bytes Read Last Second
  - Bit 4: Total Write Transfer Count
  - Bit 5: Write Transfer Count Last Second
  - Bit 6: Total Bytes Write
  - Bit 7: Bytes Write Last Second
- Bit 8..15: Clear CoE statistics (same ordering as Bit 0..7, AoE)
- Bit 16..23: Clear EoE statistics (same ordering as Bit 0..7, AoE)
- Bit 24..31: Clear FoE statistics (same ordering as Bit 0..7, AoE)
- Bit 32..39: Clear SoE statistics (same ordering as Bit 0..7, AoE)
- Bit 40..47: Clear VoE statistics (same ordering as Bit 0..7, AoE)
- Bit 48..55: Clear RawMbx statistics (same ordering as Bit 0..7, AoE)

```
qwMailboxStatisticsClearCounters = 0x0000000100; //Clear CoE Total Read Transfer
                                         ↪Count.
```

### 6.7.11 emIoControl - EC\_IOCTL\_SET\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK

Sets a bit mask to enable or disable the generation of specific error notifications of frame response errors. The application then can decide to suppress those error messages. By default all errors, expect [EC\\_FRAME\\_RESPONSE\\_ERROR\\_NOTIFY\\_MASK\\_NON\\_ECAT\\_FRAME](#) are enabled (the notification mask is set to [EC\\_FRAME\\_RESPONSE\\_ERROR\\_NOTIFY\\_MASK\\_DEFAULT](#)).

#### emIoControl - EC\_IOCTL\_SET\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK

##### Parameter

- 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: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

##### Return

EC\_E\_NOERROR or error code

The following frame response error notification mask values exist:

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_UNDEFINED**  
Mask for eRspErr\_UNDEFINED notifications

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_NO\_RESPONSE**  
Mask for eRspErr\_NO\_RESPONSE notifications

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_WRONG\_IDX**  
Mask for eRspErr\_WRONG\_IDX notifications

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_UNEXPECTED**  
Mask for eRspErr\_UNEXPECTED notifications

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_FRAME\_RETRY**  
Mask for eRspErr\_FRAME\_RETRY notifications

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_RETRY\_FAIL**  
Mask for eRspErr\_RETRY\_FAIL notifications

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_FOREIGN\_SRC\_MAC**  
Mask for eRspErr\_FOREIGN\_SRC\_MAC notifications

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_NON\_ECAT\_FRAME**  
Mask for eRspErr\_NON\_ECAT\_FRAME notifications

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_ALL**  
Mask for all notifications enabled except eRspErr\_NON\_ECAT\_FRAME

**EC\_FRAME\_RESPONSE\_ERROR\_NOTIFY\_MASK\_DEFAULT**  
Mask for all frame response error notifications

#### See also:

*emNotify - EC\_NOTIFY\_FRAME\_RESPONSE\_ERROR*

### 6.7.12 emIoControl - EC\_IOCTL\_SET\_FRAME\_LOSS\_SIMULATION

This IO Control enables the application to simulate the loss of sent and/or received EtherCAT frames for testing purposes. Three modes of operation are possible: Random, periodic or random periodic frame loss simulation.

- Random frame loss simulation: For each frame the dwFrameLossLikelihoodPpm parameter determines whether the frame will be discarded.
- Periodic frame loss simulation: After dwFixedLossNumLostFrames discarded frames, dwFixedLossNumGoodFrames frames will be processed.
- Random periodic frame loss simulation: The dwFrameLossLikelihoodPpm parameter determines whether a periodic frame loss sequence is triggered.

---

**Important:** Do not activate this on shipped releases. Frameloss has significant influence on performance and reliability of the application!

---

#### emIoControl - EC\_IOCTL\_SET\_FRAME\_LOSS\_SIMULATION

##### Parameter

- pbyInBuf: [in] Array of four EC\_T\_DWORDs (arrDword)

- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

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

**6.7.13 emIoControl - EC\_IOCTL\_SET\_RXFRAME\_LOSS\_SIMULATION**

Same as *emIoControl - EC\_IOCTL\_SET\_FRAME\_LOSS\_SIMULATION* but only enables receive direction frame losses.

**emIoControl - EC\_IOCTL\_SET\_RXFRAME\_LOSS\_SIMULATION****Parameter**

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

**6.7.14 emIoControl - EC\_IOCTL\_SET\_TXFRAME\_LOSS\_SIMULATION**

Same as *emIoControl - EC\_IOCTL\_SET\_FRAME\_LOSS\_SIMULATION* but only enables transmit direction frame losses.

**emIoControl - EC\_IOCTL\_SET\_TXFRAME\_LOSS\_SIMULATION****Parameter**

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0

- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

### 6.7.15 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 emNotify() is called. In addition to this notification code the second parameter given to emNotify() contains a pointer to an error notification descriptor of type [\*EC\\_T\\_ERROR\\_NOTIFICATION\\_DESC\*](#). This error notification descriptor contains detailed information about the error.

```
struct EC_T_ERROR_NOTIFICATION_DESC
```

**Public Members**

**EC\_T\_DWORD dwNotifyErrorCode**

Error ID (same value as the notification code)

**EC\_T\_CHAR achErrorInfo[MAX\_ERRINFO\_STRLEN]**

Additional error string (may be empty)

```
union _EC_T_ERROR_NOTIFICATION_PARM
```

**Public Members**

***EC\_T\_WKCERR\_DESC* WkcErrDesc**

WKC error descriptor

***EC\_T\_FRAME\_RSPERR\_DESC* FrameRspErrDesc**

Frame response error descriptor

***EC\_T\_INITCMD\_ERR\_DESC* InitCmdErrDesc**

Master/Slave init command error descriptor

***EC\_T\_SLAVE\_ERROR\_INFO\_DESC* SlaveErrInfoDesc**

Slave Error Info Descriptor

***EC\_T\_SLAVES\_ERROR\_DESC* SlavesErrDesc**

Slaves Error Descriptor

***EC\_T\_MBOX\_SDO\_ABORT\_DESC* SdoAbortDesc**

SDO Abort

***EC\_T\_RED\_CHANGE\_DESC* RedChangeDesc**

Redundancy Descriptor

***EC\_T\_MBOX\_FOE\_ABORT\_DESC* FoeErrorDesc**

FoE error code and string

***EC\_T\_MBXRcv\_INVALID\_DATA\_DESC* MbxRcvInvalidDataDesc**

Invalid mailbox data received descriptor

*EC\_T\_PDIWATCHDOG\_DESC* **PdiWatchdogDesc**  
PDI watchdog expired

*EC\_T\_SLAVE\_NOTSUPPORTED\_DESC* **SlaveNotSupportedDesc**  
Slave not supported

*EC\_T\_SLAVE\_UNEXPECTED\_STATE\_DESC* **SlaveUnexpectedStateDesc**  
Slave in unexpected state

*EC\_T\_SLAVES\_UNEXPECTED\_STATE\_DESC* **SlavesUnexpectedStateDesc**  
Slaves in unexpected state

*EC\_T\_EEPROM\_CHECKSUM\_ERROR\_DESC* **EEPROMChecksumErrorDesc**  
EEPROM checksum error

*EC\_T\_JUNCTION\_RED\_CHANGE\_DESC* **JunctionRedChangeDesc**  
Junction redundancy change descriptor

*EC\_T\_FRAMELOSS\_AFTER\_SLAVE\_NTFY\_DESC* **FrameLossAfterSlaveDesc**  
Frameloss after Slave descriptor

*EC\_T\_S2SMBX\_ERROR\_DESC* **S2SMBxErrorDesc**  
S2S Mailbox Error descriptor

*EC\_T\_BAD\_CONNECTION\_NTFY\_DESC* **BadConnectionDesc**  
Bad connection descriptor

*EC\_T\_COMMUNICATION\_TIMEOUT\_NTFY\_DESC* **CommunicationTimeoutDesc**  
Communication timeout descriptor

*EC\_T\_TAP\_LINK\_STATUS\_NTFY\_DESC* **TapLinkStatusDesc**  
Tap link status

If the pointer to this descriptor exists (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 `emNotify()` 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 `emIoControl - EC_IOCTL_SET_NOTIFICATION_ENABLED` it is possible to determine which errors shall be signalled and which not.

## 6.7.16 emNotify - 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 `emNotify - 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 `EC_T_ERROR_NOTIFICATION_DESC`.

This notification is enabled by default.

```
struct EC_T_WKCERR_DESC
```

## Public Members

### **`EC_T_SLAVE_PROP SlaveProp`**

Slave properties, content is undefined in case of cyclic WKC\_ERROR

### **`EC_T_BYTE byCmd`**

EtherCAT command type

### **`EC_T_DWORD dwAddr`**

Logical address or physical address (ADP/ADO)

### **`EC_T_WORD wWkcSet`**

Working counter set value

### **`EC_T_WORD wWkcAct`**

Working counter actual value

### **`EC_T_DWORD dwTaskId`**

Cyclic Task ID (ENI: Cyclic/TaskId)

### **`EC_T_WORD wMsuId`**

Master Sync Unit ID (ENI: Slave/ProcessData/RxPdo[1..4]@Su, Slave/ProcessData/TxPdo[1..4]@Su, comment at Cyclic/Frame/Cmd)

```
struct EC_T_SLAVE_PROP
```

## Public Members

### **`EC_T_WORD wStationAddress`**

Configured station address or INVALID\_FIXED\_ADDR

### **`EC_T_WORD wAutoIncAddr`**

Configured auto increment address or INVALID\_AUTO\_INC\_ADDR

### **`EC_T_CHAR achName[MAX_STD_STRLEN]`**

Configured name of the slave device (NULL terminated string)

## See also:

- *emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the deactivation.
- *Cyclic cmd WKC validation*
- *Working Counter ( WKC ) State in Diagnosis Image*
- *emIoControl - EC\_IOCTL\_SET\_AUTO\_ADJUST\_CYCCMD\_WKC\_ENABLED*

### 6.7.17 emNotify - 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 [\*EC\\_T\\_ERROR\\_NOTIFICATION\\_DESC\*](#).

### 6.7.18 emNotify - 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 [\*EC\\_T\\_ERROR\\_NOTIFICATION\\_DESC\*](#). The structure member SlaveProp contains information about the corresponding slave device.

### 6.7.19 emNotify - EC\_NOTIFY\_FOE\_MBSLAVE\_ERROR

This error will be indicated in case a slave notifies an error over FoE.

### 6.7.20 emNotify - 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 [\*EC\\_T\\_ERROR\\_NOTIFICATION\\_DESC\*](#). The structure member SlaveProp contains information about the corresponding slave device.

### 6.7.21 emNotify - 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 [\*EC\\_T\\_ERROR\\_NOTIFICATION\\_DESC\*](#). The structure member SlaveProp contains information about the corresponding slave device.

### 6.7.22 emNotify - 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.

### 6.7.23 emNotify - 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 [EC\\_T\\_ERROR\\_NOTIFICATION\\_DESC](#). The structure member SlaveProp contains information about the corresponding slave device.

### 6.7.24 emNotify - EC\_NOTIFY\_S2SMBX\_ERROR

This error will be indicated in case a Slave-To-Slave mailbox transfer fails.

**See also:**

[EC\\_E\\_S2SMBX\\_NOT\\_CONFIGURED](#)

### 6.7.25 emNotify - 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.

This notification is enabled by default.

**See also:**

[emIoControl - EC\\_IOCTL\\_SET\\_NOTIFICATION\\_ENABLED](#) for how to control the deactivation.

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:**

[emInitMaster \(\)](#)

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:

**1. 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".

**2. 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 dwMaxAcycCmdsPerCycle -> shorter master timer cycle, i.e. shorter period between two calls to

`emExecJob (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:

1. the frame was not received (due to bus problems)
2. **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 dwMaxAcycCmdsPerCycle
  - higher cyclic timer period, i.e. less calls to `emExecJob()` (`eUsrJob_SendAllCycFrames`)

### 3. non-deterministic sending of acyclic frames.

Sending acyclic frames by calling `emExecJob()` (`eUsrJob_SendAcycFrames`) have to be properly scheduled with sending cyclic frames by calling `emExecJob()` (`eUsrJob_SendAllCycFrames`).

Using the control interface `emIoControl - EC_IOCTL_SET_FRAME_RESPONSE_ERROR_NOTIFY_MASK` it is possible to determine which response errors shall be signalled and which not.

Detailed error information is stored in structure `EC_T_FRAME_RSPERR_DESC` of `EC_T_ERROR_NOTIFICATION_DESC`.

```
struct EC_T_FRAME_RSPERR_DESC
```

## Public Members

### `EC_T_BOOL bIsCyclicFrame`

Indicates whether the lost frame was a cyclic frame

### `EC_T_FRAME_RSPERR_TYPE EErrorType`

Frame response error type

### `EC_T_BYTEx byEcCmdHeaderIdxSet`

Expected IDX value, this value is valid only for acyclic frames in case EErrorType is not equal to eRspErr\_UNEXPECTED

### `EC_T_BYTEx byEcCmdHeaderIdxAct`

Actually received IDX value, this value is only valid for acyclic frames in case of EErrorType is equal to: eRspErr\_WRONG\_IDX and eRspErr\_UNEXPECTED

### `EC_T_WORD wCycFrameNum`

Number of the lost cyclic frame from the ENI

### `EC_T_DWORD dwTaskId`

Cyclic Task ID (ENI: Cyclic/TaskId). Only valid if bIsCyclicFrame is set

### enum `EC_T_FRAME_RSPERR_TYPE`

Values:

#### enumerator `eRspErr_UNDEFINED`

undefined

#### enumerator `eRspErr_NO_RESPONSE`

No Ethernet frame received (timeout, frame loss)

enumerator **eRspErr\_WRONG\_IDX**  
Wrong IDX value in acyclic frame

enumerator **eRspErr\_UNEXPECTED**  
Unexpected frame was received

enumerator **eRspErr\_FRAME\_RETRY**  
Ethernet frame will be re-sent (timeout, frame loss)

enumerator **eRspErr\_RETRY\_FAIL**  
all retry mechanism fails to re-sent acyclic frames

enumerator **eRspErr\_FOREIGN\_SRC\_MAC**  
Frame with MAC from other Master received

enumerator **eRspErr\_NON\_ECAT\_FRAME**  
Non EtherCAT frame received

enumerator **eRspErr\_CRC**  
Ethernet frame with CRC error received

## 6.7.26 emNotify - 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.

This notification is enabled by default.

Detailed error information is stored in structure *EC\_T\_INITCMD\_ERR\_DESC* of *EC\_T\_ERROR\_NOTIFICATION\_DESC*.

```
struct EC_T_INITCMD_ERR_DESC
```

### Public Members

*EC\_T\_SLAVE\_PROP* **SlaveProp**  
Slave properties

*EC\_T\_CHAR* **achStateChangeName**[MAX\_SHORT\_STRLEN]  
State change description when the error occurred

*EC\_T\_INITCMD\_ERR\_TYPE* **EErrorType**  
Init command error type

*EC\_T\_CHAR* **szComment**[MAX\_STD\_STRLEN]  
Comment (ENI)

enum **EC\_T\_INITCMD\_ERR\_TYPE**  
*Values:*

enumerator **eInitCmdErr\_NO\_ERROR**  
No error

enumerator **eInitCmdErr\_NO\_RESPONSE**  
No Ethernet frame received (timeout)

enumerator **eInitCmdErr\_VALIDATION\_ERR**  
Validation error (invalid slave command response)

enumerator **eInitCmdErr\_FAILED**  
Init commands failed (state could not be reached)

enumerator **eInitCmdErr\_NOT\_PRESENT**  
Slave not present on the bus

enumerator **eInitCmdErr\_ALSTATUS\_ERROR**  
Error in AL Status Register

enumerator **eInitCmdErr\_MBXSLAVE\_ERROR**  
Error at Mailbox Init Command

enumerator **eInitCmdErr\_PDI\_WATCHDOG**  
PDI watchdog has been detected

#### See also:

[emIoControl - EC\\_IOCTL\\_SET\\_NOTIFICATION\\_ENABLED](#) for how to control the deactivation

### 6.7.27 emNotify - EC\_NOTIFY\_MBSLAVE\_INITCMD\_TIMEOUT

This error is identical to error code [emNotify - 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.

### 6.7.28 emNotify - 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 [\*EC\\_T\\_ERROR\\_NOTIFICATION\\_DESC\*](#).

## 6.7.29 emNotify - EC\_NOTIFY\_NOT\_ALL\_DEVICES\_OPERATIONAL

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

## 6.7.30 emNotify - EC\_NOTIFY\_ALL\_DEVICES\_OPERATIONAL

When processing cyclic frames the EtherCAT master checks whether all slaves are still in OPERATIONAL state. This will be notified after *emNotify - EC\_NOTIFY\_NOT\_ALL\_DEVICES\_OPERATIONAL* and all the slaves are back in OPERATIONAL state.

## 6.7.31 emNotify - 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 *emNotify - 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.

This notification is enabled by default.

### See also:

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the deactivation

## 6.7.32 emNotify - 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 *EC\_T\_ERROR\_NOTIFICATION\_DESC*. This notification is enabled by default.

struct **EC\_T\_SLAVE\_ERROR\_INFO\_DESC**

### Public Members

**EC\_T\_SLAVE\_PROP SlaveProp**  
Slave properties

**EC\_T\_WORD wStatus**  
Slave Status (AL Status)

**EC\_T\_WORD wStatusCode**  
Error status code (AL STATUS CODE)

### See also:

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the deactivation

### 6.7.33 emNotify - EC\_NOTIFY\_SLAVES\_ERROR\_STATUS

This notification collects notifications of type *emNotify - 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.

```
struct EC_T_SLAVES_ERROR_DESC
```

#### Public Members

**EC\_T\_WORD wCount**

Number of slave errors

*EC\_T\_SLAVES\_ERROR\_DESC\_ENTRY* **SlaveError**[MAX\_SLAVES\_ERROR\_NTFY\_ENTRIES]

Slave error descriptions

```
struct EC_T_SLAVES_ERROR_DESC_ENTRY
```

#### Public Members

**EC\_T\_WORD wStationAddress**

Slave station address

**EC\_T\_WORD wStatus**

Slave status (AL Status)

**EC\_T\_WORD wStatusCode**

Slave status code (AL Control Status)

#### See also:

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the activation

### 6.7.34 emNotify - 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 *EC\_T\_ERROR\_NOTIFICATION\_DESC*. This notification is enabled by default.

```
struct EC_T_SLAVE_UNEXPECTED_STATE_DESC
```

#### Public Members

*EC\_T\_SLAVE\_PROP* **SlaveProp**

Slave properties

*EC\_T\_STATE* **curState**

Current state

*EC\_T\_STATE* **expState**

Expected state

**See also:**

[emIoControl - EC\\_IOCTL\\_SET\\_NOTIFICATION\\_ENABLED](#) for how to control the deactivation

### 6.7.35 emNotify - EC\_NOTIFY\_SLAVES\_UNEXPECTED\_STATE

This notification collects notifications of type *emNotify - 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.

```
struct EC_T_SLAVES_UNEXPECTED_STATE_DESC
```

#### Public Members

##### EC\_T\_WORD wCount

Number of unexpected slave state changes

##### *EC\_T\_SLAVES\_UNEXPECTED\_STATE\_DESC\_ENTRY*

```
SlaveStates[MAX_SLAVES_UNEXPECTED_STATE_NTFY_ENTRIES]
```

Slave state change descriptions

```
struct EC_T_SLAVES_UNEXPECTED_STATE_DESC_ENTRY
```

#### Public Members

##### EC\_T\_WORD wStationAddress

Slave station address

##### *EC\_T\_STATE curState*

Current state

##### *EC\_T\_STATE expState*

Expected state

**See also:**

[emIoControl - EC\\_IOCTL\\_SET\\_NOTIFICATION\\_ENABLED](#) for how to control the activation

### 6.7.36 emNotify - EC\_NOTIFY\_ETH\_LINK\_NOT\_CONNECTED

This notification will be indicated if the Ethernet link is disconnected. This error is never indicated if the Real-time Ethernet Driver does not support detection of a missing link cable.

In case of permanent frame loss no slaves can be found although the slaves are connected. This does not affect link connection detection therefore this notification will be not indicated on permanent frame loss.

### 6.7.37 emNotify - EC\_NOTIFY\_ETH\_LINK\_CONNECTED

This notification will be indicated if the Ethernet link is reconnected after a disconnect. This notification is never indicated if the Real-time Ethernet Driver does not support detection of a missing link cable.

### 6.7.38 emNotify - EC\_NOTIFY\_CLIENTREGISTRATION\_DROPPED

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

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

### 6.7.39 emNotify - 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 `EC_T_ERROR_NOTIFICATION_DESC`.

```
struct EC_T_EEPROM_CHECKSUM_ERROR_DESC
```

#### Public Members

`EC_T_SLAVE_PROP SlaveProp`  
Slave properties

### 6.7.40 emNotify - EC\_NOTIFY\_MBXRCV\_INVALID\_DATA

This error is signalized when invalid mailbox data have been received from slave. Detailed error information is stored in structure `EC_T_MBXRCV_INVALID_DATA_DESC` of `EC_T_ERROR_NOTIFICATION_DESC`.

```
struct EC_T_MBXRCV_INVALID_DATA_DESC
```

#### Public Members

`EC_T_SLAVE_PROP SlaveProp`  
Slave properties

### 6.7.41 emNotify - 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 `EC_T_ERROR_NOTIFICATION_DESC`.

```
struct EC_T_PDIWATCHDOG_DESC
```

## Public Members

`EC_T_SLAVE_PROP SlaveProp`  
Slave properties

### 6.7.42 `ecatGetText`

const EC\_T\_CHAR \***ecatGetText** (EC\_T\_DWORD dwTextId)

### 6.7.43 `emLogFrameEnable`

```
static EC_T_DWORD ecatLogFrameEnable (
    EC_T_PFLGFRAME_CB pvLogFrameCallBack,
    EC_T_VOID *pvContext
)
EC_T_DWORD emLogFrameEnable (
    EC_T_DWORD dwInstanceID,
    EC_T_PFLGFRAME_CB pvLogFrameCallBack,
    EC_T_VOID *pvContext
)
Setup a callback function to log the EtherCAT network traffic.
```

The callback function is called by the cyclic task. Therefore the code inside the callback has to be fast and non-blocking. The callback parameter dwLogFlags can be used as a filter to log just specific frames. The master discards the frame if the callback function modifies the Ethernet frame type at byte offset 12.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pvLogFrameCallBack** – [in] Pointer to frame logging callback function
- **pvContext** – [in] Pointer to function specific context

#### Returns

`EC_E_NOERROR` or error code

typedef EC\_T\_VOID (\***EC\_T\_PFLGFRAME\_CB**)(EC\_T\_VOID \*pvContext, EC\_T\_DWORD dwLogFlags,  
EC\_T\_DWORD dwFrameSize, EC\_T\_BYTEx \*pbyFrame)

**Note:** The master discards the frame if the callback function modifies the Ethernet frame type at byte offset 12.

#### Parameters

- **pvContext** – [in] Arbitrarily application-defined parameter passed to callback
- **dwLogFlags** – [in] Frame logging flags, EC\_LOG\_FRAME\_FLAG\_...
- **dwFrameSize** – [in] Size of frame in bytes
- **pbyFrame** – [in] Pointer to frame data

#### `EC_LOG_FRAME_FLAG_MASTERSTATE_MASK`

Bit 0 to 15: Master state mask

**EC\_LOG\_FRAME\_FLAG\_ACYC\_FRAME**

Bit 16 (0x00010000): 0=cyclic frame, 1=acyclic frame

**EC\_LOG\_FRAME\_FLAG\_DBG\_FRAME**

Bit 17 (0x00020000): 0=EtherCAT frame, 1=debug frame

**EC\_LOG\_FRAME\_FLAG\_RED\_FRAME**

Bit 18 (0x00040000): 0=main frame, 1=red frame

**EC\_LOG\_FRAME\_FLAG\_RX\_FRAME**

Bit 19 (0x00080000): 0=TX frame, 1=RX frame

**EC\_LOG\_FRAME\_FLAG\_MASTER\_RED\_FRAME**

Bit 20 (0x00100000): 0=slave frame, 1=MasterMaster frame

```
/****************************************************************************
 ** \brief Handler to log frames.
 *
 * CAUTION: Called by cyclic task!!! Do not consume too much CPU time!!!
 */
EC_T_VOID LogFrameHandler(EC_T_VOID* pvContext, EC_T_DWORD dwLogFlags, EC_T_DWORD
                           dwFrameSize, EC_T_BYTE* pbyFrame)
{
    EC_T_STATE eMasterState;

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

    /* skip tx frame */
    if ((S_dwLogFrameLevel == 3) && !(dwLogFlags & EC_LOG_FRAME_FLAG_RX_FRAME))
        return;

    /* skip cyclic frame */
    if ((S_dwLogFrameLevel == 2) && !(dwLogFlags & EC_LOG_FRAME_FLAG_ACYC_FRAME))
        return;

    /* skip red frame */
    if (dwLogFlags & EC_LOG_FRAME_FLAG_RED_FRAME)
        return;

    /* do something with pbyFrame ... */
}
```

## 6.7.44 emLogFrameDisable

static EC\_T\_DWORD **ecatLogFrameDisable** (EC\_T\_VOID)  
 EC\_T\_DWORD **emLogFrameDisable** (EC\_T\_DWORD dwInstanceID)  
 Disable the frame logging callback.

**Parameters**

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

**Returns**

*EC\_E\_NOERROR* or error code

## 6.7.45 emGetMasterInfo

static EC\_T\_DWORD **ecatGetMasterInfo** (*EC\_T\_MASTER\_INFO* \*pMasterInfo)

```
EC_T_DWORD emGetMasterInfo (
    EC_T_DWORD dwInstanceID,
    EC_T_MASTER_INFO *pMasterInfo
)
```

) Get generic information about the Master.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMasterInfo** – [out] Master information

### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or pParms is NULL or contains values out of range
- *EC\_E\_ADS\_IS\_RUNNING* if ADS server is running

struct **EC\_T\_MASTER\_INFO**

### Public Members

**EC\_T\_DWORD dwMasterVersion**  
Master version

*EC\_T\_BUS\_DIAGNOSIS\_INFO* **BusDiagnosisInfo**  
Bus diagnostics

*EC\_T\_MAILBOX\_STATISTICS* **MailboxStatistics**  
Mailbox statistics

**EC\_T\_REDUNDANCY\_DIAGNOSIS\_INFO RedundancyDiagnosisInfo**  
Redundancy diagnosis info

**EC\_T\_DWORD dwMasterStateSummary**  
Master state summary

**EC\_T\_DWORD dwMasterVersionType**  
Master version type (EC\_VERSION\_TYPE\_...)

**EC\_T\_WORD wMasterStateSummaryDiagBitOffset**  
Bit offset of Master state summary in diagnosis image

**EC\_T\_WORD wMasterStateSummaryDiagBitSize**  
Bit offset size of Master state summary in diagnosis image

struct **EC\_T\_BUS\_DIAGNOSIS\_INFO**

**Public Members**

**EC\_T\_DWORD dwCRC32ConfigChecksum**  
CRC32 checksum of the loaded configuration

**EC\_T\_DWORD dwNumSlavesFound**  
Number of slaves connected

**EC\_T\_DWORD dwNumDCSlavesFound**  
Number of slaves with DC enabled connected

**EC\_T\_DWORD dwNumCfgSlaves**  
Number of slaves in ENI

**EC\_T\_DWORD dwNumMbxSlaves**  
Number of slaves in ENI with mailbox support

**EC\_T\_DWORD dwTxFrames**  
Number of frames sent

**EC\_T\_DWORD dwRXFrames**  
Number of frames received

**EC\_T\_DWORD dwLostFrames**  
Number of lost frames

**EC\_T\_DWORD dwCyclicFrames**  
Number of cyclic frames sent

**EC\_T\_DWORD dwCyclicDatagrams**  
Number of cyclic datagrams / EtherCAT commands sent

**EC\_T\_DWORD dwAcyclicFrames**  
Number of acyclic frames sent

**EC\_T\_DWORD dwAcyclicDatagrams**  
Number of acyclic datagrams / EtherCAT commands sent

**EC\_T\_DWORD dwClearCounters**  
Clear frame / datagram counter bit field

**EC\_T\_DWORD dwCyclicLostFrames**  
Number of cyclic lost frames

**EC\_T\_DWORD dwAcyclicLostFrames**  
Number of acyclic lost frames

struct **EC\_T\_MAILBOX\_STATISTICS**

**Public Members**

***EC\_T\_STATISTIC\_TRANSFER\_DUPLEX AoE***  
AoE mailbox transfer statistics

***EC\_T\_STATISTIC\_TRANSFER\_DUPLEX CoE***  
CoE mailbox transfer statistics

***EC\_T\_STATISTIC\_TRANSFER\_DUPLEX EoE***  
EoE mailbox transfer statistics

***EC\_T\_STATISTIC\_TRANSFER\_DUPLEX FoE***  
FoE mailbox transfer statistics

***EC\_T\_STATISTIC\_TRANSFER\_DUPLEX SoE***  
SoE mailbox transfer statistics

***EC\_T\_STATISTIC\_TRANSFER\_DUPLEX VoE***  
VoE mailbox transfer statistics

***EC\_T\_STATISTIC\_TRANSFER\_DUPLEX RawMbx***  
Raw mailbox transfer statistics

struct **EC\_T\_STATISTIC\_TRANSFER\_DUPLEX**

**Public Members**

***EC\_T\_STATISTIC\_TRANSFER Read***  
Number of read transfers

***EC\_T\_STATISTIC\_TRANSFER Write***  
Number of write transfers

struct **EC\_T\_STATISTIC\_TRANSFER**

**Public Members**

***EC\_T\_STATISTIC Cnt***  
Number of transfers

***EC\_T\_STATISTIC Bytes***  
Number of bytes transferred

struct **EC\_T\_STATISTIC**

## Public Members

**EC\_T\_DWORD dwTotal**  
Total

**EC\_T\_DWORD dwLast**  
Last

### emGetMasterInfo() Example

```
EC_T_MASTER_INFO oMasterInfo;
OsMemset(&oMasterInfo, 0, sizeof(EC_T_MASTER_INFO));
dwRes = emGetMasterInfo(dwInstanceId, &oMasterInfo);
```

## 6.7.46 emGetMemoryUsage

```
static EC_T_DWORD ecatGetMemoryUsage (
    EC_T_DWORD *pdwCurrentUsage,
    EC_T_DWORD *pdwMaxUsage
)
```

```
EC_T_DWORD emGetMemoryUsage (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD *pdwCurrentUsage,
    EC_T_DWORD *pdwMaxUsage
)
```

Returns information about memory usage.

All calls to malloc/free and new/delete are monitored.

#### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pdwCurrentUsage** – [out] Current memory usage in Bytes at the time where this function is called
- **pdwMaxUsage** – [out] Maximum memory usage in Bytes since initialization at the time where this function is called

#### Returns

*EC\_E\_NOERROR* or error code

### emGetMemoryUsage() Example

```
EC_T_DWORD dwCurrentUsage = EC_NULL;
EC_T_DWORD dwMaxUsage = EC_NULL;
dwRes = emGetMemoryUsage(dwInstanceId, &dwCurrentUsage, &dwMaxUsage);
```

## 6.7.47 emGetMasterDump

```
static EC_T_DWORD ecatGetMasterDump (
    EC_T_BYTE *pbyBuffer,
    EC_T_DWORD dwBufferSize,
    EC_T_DWORD *pdwDumpSize
)
EC_T_DWORD emGetMasterDump (
    EC_T_DWORD dwInstanceId,
    EC_T_BYTE *pbyBuffer,
    EC_T_DWORD dwBufferSize,
    EC_T_DWORD *pdwDumpSize
)
```

The dump contains relevant information about the master and slave status.

The dump is only intended for internal troubleshooting at acontis. Amongst others it contains the following descriptors:

- *EC\_T\_INIT\_MASTER\_PARMS*
- *EC\_T\_BUS\_DIAGNOSIS\_INFO*
- *EC\_T\_MAILBOX\_STATISTICS*
- *EC\_T\_CFG\_SLAVE\_INFO*
- *EC\_T\_BUS\_SLAVE\_INFO*
- *EC\_T\_SLVSTATISTICS\_DESC*

The buffer is written until all relevant data have been dumped or buffer size has been exceeded.

### Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pbyBuffer** – [in] Preallocated buffer to dump log data
- **dwBufferSize** – [in] Size of preallocated buffer
- **pdwDumpSize** – [out] Size of master dump

### Returns

- *EC\_E\_NOERROR*
- *EC\_E\_NOMEMORY* if buffer too small

### emGetMasterDump() Example

```
EC_T_DWORD dwBufferSize = 8192;
EC_T_BYTE* byBuffer = (EC_T_BYTE*)OsMalloc(dwBufferSize);
EC_T_DWORD dwDumpSize = dwBufferSize;
dwRes = emGetMasterDump(dwInstanceId, byBuffer, dwBufferSize, &dwDumpSize);
```

## 6.7.48 emGetMasterSyncUnitInfoNumOf

```
static EC_T_DWORD ecatGetMasterSyncUnitInfoNumOf (EC_T_VOID)
EC_T_DWORD emGetMasterSyncUnitInfoNumOf (EC_T_DWORD dwInstanceID)
    Get number of Master Sync Units info entries.
```

### Parameters

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

### Returns

Number of Master Sync Units info entries

### emGetMasterSyncUnitInfoNumOf() Example

```
/* get Master Sync Units info entries count */
EC_T_DWORD dwSyncedUnitsCount = emGetMasterSyncUnitInfoNumOf(dwInstanceId);
EcLogMsg(EC_LOG_LEVEL_INFO, (pEcLogContext, EC_LOG_LEVEL_INFO,
    "Units count: %d", dwSyncedUnitsCount));
```

## 6.7.49 emGetMasterSyncUnitInfo

```
static EC_T_DWORD ecatGetMasterSyncUnitInfo (
    EC_T_WORD wMsuId,
    EC_T_MSU_INFO *pMsuInfo
)
EC_T_DWORD emGetMasterSyncUnitInfo (
    EC_T_DWORD dwInstanceID,
    EC_T_WORD wMsuId,
    EC_T_MSU_INFO *pMsuInfo
)
    Get information about specific Master Sync Unit.
```

### Parameters

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

### Returns

**EC\_E\_NOERROR** or error code

MSU\_ID\_ALL\_INFO\_ENTRIES retrieves the information from all master sync units at once. The application must ensure that pMsuInfo is capable for all entries.

struct **EC\_T\_MSU\_INFO**

**Public Members**

**EC\_T\_WORD wMsuId**  
 [out] Master Sync Unit ID (ENI: Slave/ProcessData/RxPdo[1..4]@Su,  
 Slave/ProcessData/TxPdo[1..4]@Su, comment at Cyclic/Frame/Cmd)

**EC\_T\_DWORD dwBitOffsIn**  
 [out] Process Data Image INPUTs bit offset

**EC\_T\_DWORD dwBitSizeIn**  
 [out] Process Data Image INPUTs bit length

**EC\_T\_DWORD dwBitOffsOut**  
 [out] Process Data Image OUTPUTs bit offset

**EC\_T\_DWORD dwBitSizeOut**  
 [out] Process Data Image OUTPUTs bit length

**EC\_T\_WORD wWkcStateDiagOffsIn**  
 [out] INPUTs WkcState bit offset in Diagnosis Image. (Bit values: 0 = Process Data valid, 1 = Process Data invalid)

**EC\_T\_WORD wWkcStateDiagOffsOut**  
 [out] OUTPUTs WkcState bit offset in Diagnosis Image. (Bit values: 0 = Process Data valid, 1 = Process Data invalid)

**EC\_T\_DWORD adwReserved[16]**  
 reserved

**emGetMasterSyncUnitInfo() Example**

```
/* get information about specific Master Sync Unit */
EC_T_WORD wMsuId = 0;
EC_T_MSU_INFO oMsuInfo;
OsMemset(&oMsuInfo, 0, sizeof(EC_T_MSU_INFO));
dwRes = emGetMasterSyncUnitInfo(dwInstanceId, wMsuId, &oMsuInfo);
```

**See also:**[emGetMasterSyncUnitInfoNumOf\(\)](#)**6.7.50 emBadConnectionsDetect**

```
static EC_T_DWORD ecatBadConnectionsDetect (
    EC_T_BOOL bRefreshSlaveStatistics,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emBadConnectionsDetect (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bRefreshSlaveStatistics,
    EC_T_DWORD dwTimeout
)
    Detects bad connections.
```

Analyzes the slave ESC error counters:

- Invalid Frame Counter (0x0300),
- RX Error Counter (0x0301),
- Lost Link Counter (0x0310),

whether there is a problem in the area PHY - connector - cable - connector - PHY. If one of the above error counters shows a value not equal to zero, an EC\_NOTIFY\_BAD\_CONNECTION is generated, which contains the exact position of the faulty connection. It is recommended to call emBadConnectionsReset() on startup of EC-Master to ensure that all error counters of all slaves are in a defined state.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bRefreshSlaveStatistics** – [in] EC\_TRUE: refresh ESC error counters, EC\_FALSE: process current ESC error counters
- **dwTimeout** – [in] Timeout [ms] May not be EC\_NOWAIT!

#### Returns

*EC\_E\_NOERROR* or error code

#### emBadConnectionsDetect() Example

```
dwRes = emBadConnectionsDetect(dwInstanceId, EC_TRUE, 5000 /* timeout */);
```

#### See also:

*emBadConnectionsReset ()*

### 6.7.51 emBadConnectionsReset

```
static EC_T_DWORD ecatBadConnectionsReset (EC_T_VOID)
static EC_T_DWORD emBadConnectionsReset (EC_T_DWORD dwInstanceId)
    Clears all error counters (0x0300 - 0x0313) of all slaves.
```

#### Parameters

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

#### emBadConnectionsReset() Example

```
dwRes = emBadConnectionsReset(dwInstanceId);
```

### 6.7.52 emNotify - EC\_NOTIFY\_BAD\_CONNECTION

This error is signalized every time a bad connection is detected within the call of *emBadConnectionsDetect ()*. It contains the exact location of the bad connection between two slaves. This notification is enabled by default.

struct **EC\_T\_BAD\_CONNECTION\_NTFY\_DESC**

**Public Members**

***EC\_T\_SLAVE\_PROP SlavePropParent***  
slave properties of parent slave

***EC\_T\_WORD wPortAtParent***  
port at parent slave

***EC\_T\_SLAVE\_PROP SlavePropChild***  
slave properties of child slave

***EC\_T\_WORD wPortAtChild***  
port at child slave

**See also:**

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the deactivation

**6.7.53 emSelfTestScan**

static EC\_T\_DWORD **ecatSelfTestScan** (*EC\_T\_SELFTESTSCAN\_PARMS* \*pParms)

```
EC_T_DWORD emSelfTestScan (
    EC_T_DWORD dwInstanceID,
    EC_T_SELFTESTSCAN_PARMS *pParms
)
) Self test scan.
```

Send a burst of numerous frames and analyze the slave connections. After deactivating the job task, frames will be sent as fast as the LinkLayer can send them. The size of the frames increases and decreases between the defined limits. Dependent on the parameters the BadConnectionsDetect API will analyze the slave connections.

**Parameters**

- ***dwInstanceID*** – [in] Instance ID (Multiple EtherCAT Network Support)
- ***pParms*** – [in] Self-test scan parameters

**Returns**

- ***EC\_E\_NOERROR*** if successful
- ***EC\_E\_INVALIDSTATE*** if master isn't initialized
- ***EC\_E\_INVALIDPARM*** if dwInstanceID is out of range or pParms is NULL or contains values out of range
- ***EC\_E\_BAD\_CONNECTION*** if bad connection was detected
- ***EC\_E\_FRAME\_LOST*** if frame(s) lost during self-test
- ***EC\_E\_NOTSUPPORTED*** if not in polling mode
- ***EC\_E\_MASTER\_RED\_STATE\_INACTIVE*** if Master Redundancy is configured and master is inactive

struct **EC\_T\_SELFTESTSCAN\_PARMS**

**Public Members****EC\_T\_DWORD dwSize**

[in] Set to sizeof(EC\_T\_SELFTESTSCAN\_PARMS)

**EC\_T\_DWORD dwTimeout**

[in] Timeout [ms], 0 or EC\_NOWAIT defaults to 500ms

**EC\_T\_DWORD dwFrameCount**

[in] Total number of frames sent during the self-test. Default value is 1500. A value of 0 let the current setting unmodified.

**EC\_T\_DWORD dwFrameSizeMin**

[in] Min frame size [bytes]. Default value is 60. A value of 0 let the current setting unmodified.

**EC\_T\_DWORD dwFrameSizeMax**

[in] Max frame size [bytes]. Default value is 1514. A value of 0 let the current setting unmodified.

**EC\_T\_DWORD dwFrameSizeStep**

[in] Size [bytes] by which the frame increases or decreases continuously during the self-test. Default value is 1. A value of 0 let the current setting unmodified.

**EC\_T\_BOOL bDetectBadConnections**

[in] Execute the bad connection detection after self-test

**EC\_T\_UINT64 qwFrameRoundtripTimeAvg**

[out] Roundtrip time average [us]. Time taken from sending to receiving the frame (master application level).

**EC\_T\_UINT64 qwFrameRoundtripTimeMin**

[out] Roundtrip time minimum [us]. Time taken from sending to receiving the frame (master application level).

**EC\_T\_UINT64 qwFrameRoundtripTimeMax**

[out] Roundtrip time maximum [us]. Time taken from sending to receiving the frame (master application level).

**EC\_T\_BOOL bMeasureRoundtripTimeForSingleFrame**

[in] Execute roundtrip time calculation for single frame

**emSelfTestScan() Example**

```
EC_T_SELFTESTSCAN_PARMS oParms;
OsMemset(&oParms, 0, sizeof(EC_T_SELFTESTSCAN_PARMS));
oParms.dwSize = sizeof(EC_T_SELFTESTSCAN_PARMS);
oParms.dwTimeout = 5000;
oParms.dwFrameCount = 1500;
oParms.dwFrameSizeMin = 60;
oParms.dwFrameSizeMax = 1514;
oParms.dwFrameSizeStep = 1;
oParms.bDetectBadConnections = EC_FALSE;
dwRes = emSelfTestScan(dwInstanceId, &oParms);
```

**See also:**

*emBadConnectionsDetect ()*

## 6.8 Performance Measurement

The acontis EC-Master software has a built-in performance measurement capability. This can be used to measure the execution times of the job functions that are called within the cyclic part of the application, as well as application specific functions. These executions times can be recorded both in form of overall statistics (min/avg/max) and in form of histograms.

### 6.8.1 Enabling performance measurements

Performance measurements need to be enabled inside the master init parms.

```
/* enable performance measurements */
oInitParms.PerfMeasInternalParms.bEnabled = EC_TRUE;

/* initialize the master */
dwRes = ecatInitMaster(&oInitParms);
```

**See also:**

[EC\\_T\\_PERF\\_MEAS\\_INTERNAL\\_PARMS](#)

### 6.8.2 Retrieving overall performance statistics (min/avg/max)

Performance measurements in the example application can be activated using the command line parameter (-perf). It enables performance measurements and performance histograms as well. The resulting measurement values are recorded every few seconds to the log file, and printed to the console in the following format:

PerfMsmt	'JOB_ProcessAllRxFrames'	(min/avg/max)	[usec]	: 12.5/ 15.9/ 25.6
PerfMsmt	'JOB_SendAllCycFrames'	(min/avg/max)	[usec]	: 3.6/ 5.7/ 14.8
PerfMsmt	'JOB_MasterTimer'	(min/avg/max)	[usec]	: 2.1/ 3.7/ 8.2
PerfMsmt	'JOB_SendAcycFrames'	(min/avg/max)	[usec]	: 0.3/ 0.6/ 2.6
PerfMsmt	'Cycle Time'	(min/avg/max)	[usec]	: 918.4/ 999.6/1067.9
PerfMsmt	'myAppWorkPd'	(min/avg/max)	[usec]	: 0.1/ 0.4/ 0.8
PerfMsmt	'JOB_Total'	(min/avg/max)	[usec]	: 19.0/ 25.9/ 39.2

In an application these values can be retrieved using the *emPerfMeasGet*/*emPerfMeasAppGet* APIs. These APIs require the index of a measurement point. Note that the index of a particular measurement point is implementation defined and not the same in all versions. It should therefore be detected at runtime using *emPerfMeasGetInfo*. The following example shows how measurements for *JOB\_ProcessAllRxFrames* can be retrieved:

```
EC_T_DWORD dwProcessAllRxFramesIdx = 0;
EC_T_DWORD dwMeasNum = 0;
dwRes = ecatPerfMeasGetNumOf(&dwMeasNum);

/* find index of perf measurement */
for (EC_T_DWORD i = 0; i < dwMeasNum; ++i)
{
    EC_T_PERF_MEAS_INFO PerfMeasInfo;
    dwRes = ecatPerfMeasGetInfo(i, &PerfMeasInfo, 1);

    if (0 == OsStrncmp("JOB_ProcessAllRxFrames", PerfMeasInfo.szName, OsStrlen(
        "JOB_ProcessAllRxFrames") + 1))
    {
        dwProcessAllRxFramesIdx = i;
        break;
    }
}
```

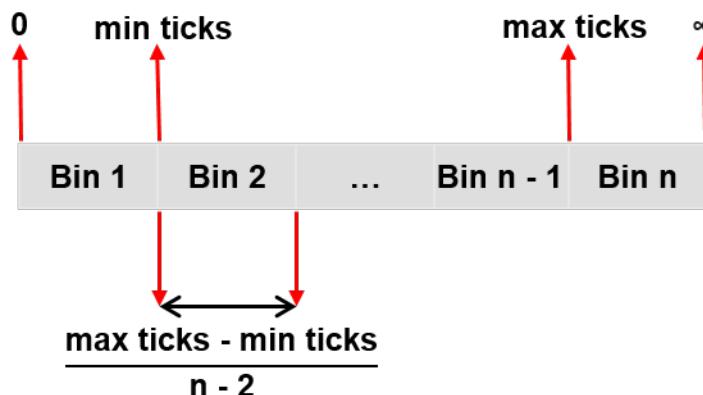
(continues on next page)

(continued from previous page)

```
/* retrieve values */
EC_T_PERF_MEAS_VAL PerfMeasVal;
dwRes = ecatPerfMeasGetRaw(dwProcessAllRxFramesIdx, &PerfMeasVal, EC_NULL, 1);
```

### 6.8.3 Recording performance histograms

In addition to the overall statistics described above it is possible to create a histogram of all results of a particular benchmark. The histogram has the following format:



```
/* enabling histograms */
EC_T_PERF_MEAS_COUNTER_PARMS* pHistParms = EC_NULL;
pHistParms = &oInitParms.PerfMeasInternalParms.HistogramParms;
/* amount of bins to use for the histogram */
pHistParms->dwBinCount = 1000;
/* range of the histograms.
 * - results below qwMinTicks are stored in the first bin
 * - results above qwMaxTicks are stored in the last bin
 *
 * a good starting point is the range 0 <-> 2 * the amount of ticks per cycle
 */
pHistParms->qwMinTicks = 0;
pHistParms->qwMaxTicks = 2 * qwTicksPerCycle;

/* initialize the master */
dwRes = ecatInitMaster(&oInitParms);
```

#### See also:

*EC\_T\_PERF\_MEAS\_HISTOGRAM\_PARMS*

Similar to the overall statistics it is possible to retrieve the histograms using *emPerfMeasGetRaw*:

```
/* retrieve values */
EC_T_PERF_MEAS_HISTOGRAM PerfMeasHist;
PerfMeasHist.aBins = (EC_T_DWORD*)OsMalloc(dwBinCount * sizeof(EC_T_DWORD));
PerfMeasHist.dwBinCount = dwBinCount;
dwRes = ecatPerfMeasGetRaw(dwProcessAllRxFramesIdx, EC_NULL, &PerfMeasHist, 1);
```

## 6.8.4 Special benchmark types

In addition to the normal benchmarks as described above, there are some special benchmark types which are flagged in `EC_T_PERF_MEAS_INFO`.

### `EC_T_PERF_MEAS_FLAG_OFFSET`

Marks the measurement as an offset benchmark, measured from the eUsrJob\_StartTask to the end of the benchmark. This flag has no influence on the measurement and is simply passed through to the application.

### `EC_T_PERF_MEAS_FLAG_LONG_TIMER`

Changes the default of qwMinTicks/qwMaxTicks selected when passing qwMinTicks=qwMinTicks=0 from 0  
– cycle time to 0.5 \* cycle time – 1.5 \* cycle time

## 6.8.5 Application benchmarks

In addition to the internal benchmarks it is possible to create application specific benchmarks using the *emPerfMeasApp* API.

```
static EC_T_PERF_MEAS_INFO_PARMS S_aPerfMeasInfos[] =
{
    {"myBench", 0}
};

#define APPL_PERF_MEAS_NUM      (sizeof(S_aPerfMeasInfos) / sizeof(S_
→aPerfMeasInfos[0]))
#define PERF_myBench            0

EC_T_PERF_MEAS_APP_PARMS oPerfMeasAppParms;
OsMemset(&oPerfMeasAppParms, 0, sizeof(EC_T_PERF_MEAS_APP_PARMS));
oPerfMeasAppParms.dwNumMeas = APPL_PERF_MEAS_NUM;
oPerfMeasAppParms.aPerfMeasInfos = S_aPerfMeasInfos;

dwRes = ecatPerfMeasAppCreate( &oPerfMeasAppParms, EC_NULL);

ecatPerfMeasAppStart(EC_NULL, PERF_myBench);
/* benchmarked work */
ecatPerfMeasAppEnd(EC_NULL, PERF_myBench);
```

## 6.8.6 API

### `emPerfMeasAppCreate`

```
static EC_T_DWORD ecatPerfMeasAppCreate (
    EC_T_PERF_MEAS_APP_PARMS *pPerfMeasAppParms,
    EC_T_VOID **ppvPerfMeas
)
EC_T_DWORD emPerfMeasAppCreate (
    EC_T_DWORD dwInstanceID,
    EC_T_PERF_MEAS_APP_PARMS *pPerfMeasAppParms,
    EC_T_VOID **ppvPerfMeas
)
```

Create PerfMeas object and bind it to the master instance.

This API can be called multiple times to create PerfMeas objects. The performance counters in each of the objects can be accessed in the following two ways:

- by passing the PerfMeas object and the index of the performance measurement. The index ranges from [0-pPerfMeasAppParms->dwNumAppMeas]

- by passing EC\_NULL instead of a PerfMeas object and an index. In this case the index works across all PerfMeas objects bound to the master instance.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pPerfMeasAppParms** – [in] Pointer to parameter definitions
- **ppvPerfMeas** – [out] Created PerfMeas object.

#### Returns

*EC\_E\_NOERROR* or an error code

struct **EC\_T\_PERF\_MEAS\_APP\_PARMS**

#### Public Members

**EC\_T\_DWORD dwNumMeas**

[in] Number of performance counters to create

**EC\_T\_PERF\_MEAS\_INFO\_PARMS \*aPerfMeasInfos**

[in] PerfMeasInfos associated with the corresponding benchmark

**EC\_T\_PERF\_MEAS\_COUNTER\_PARMS CounterParms**

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

**EC\_T\_PERF\_MEAS\_HISTOGRAM\_PARMS HistogramParms**

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

#### emPerfMeasAppDelete

static EC\_T\_DWORD **ecatPerfMeasAppDelete** (EC\_T\_VOID \*pvPerfMeas)

**EC\_T\_DWORD emPerfMeasAppDelete (**

**EC\_T\_DWORD dwInstanceID,**  
     **EC\_T\_VOID \*pvPerfMeas**

)

Delete application performance measurement and unbind it from the master instance.

Objects which are not deleted using PerfMeasAppDelete are automatically deleted when calling DeinitMaster.

---

**Note:** This invalidates the global index used when passing EC\_NULL into the other PerfMeasApp functions

---

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pvPerfMeas** – [in] PerfMeas object to delete

#### Returns

*EC\_E\_NOERROR* or an error code

## **emPerfMeasAppStart**

```
static EC_T_DWORD ecatPerfMeasAppStart (
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD dwIndex
)
EC_T_DWORD emPerfMeasAppStart (
    EC_T_DWORD dwInstanceID,
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD dwIndex
)
)
Start application performance measurement.
```

### **Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pvPerfMeas** – [in] PerfMeas object or EC\_NULL to use continuous index
- **dwIndex** – [in] Index of the performance measurement

### **Returns**

*EC\_E\_NOERROR* or an error code

## **emPerfMeasAppEnd**

```
static EC_T_DWORD ecatPerfMeasAppEnd (EC_T_VOID *pvPerfMeas, EC_T_DWORD dwIndex)

EC_T_DWORD emPerfMeasAppEnd (
    EC_T_DWORD dwInstanceID,
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD dwIndex
)
)
Stop application performance measurement.
```

### **Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pvPerfMeas** – [in] PerfMeas object or EC\_NULL to use continuous index
- **dwIndex** – [in] Index of the performance measurement

### **Returns**

*EC\_E\_NOERROR* or an error code

## **emPerfMeasAppReset**

```
static EC_T_DWORD ecatPerfMeasAppReset (
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD dwIndex
)
EC_T_DWORD emPerfMeasAppReset (
    EC_T_DWORD dwInstanceID,
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD dwIndex
)
)
Reset application performance measurement.
```

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pvPerfMeas** – [in] PerfMeas object or EC\_NULL to use continuous index
- **dwIndex** – [in] Index of the performance measurement, use EC\_PERF\_MEAS\_ALL to reset all

**Returns***EC\_E\_NOERROR* or an error code**emPerfMeasAppGetNumOf**

```
static EC_T_DWORD ecatPerfMeasAppGetNumOf (
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD *pdwNumOf
)
EC_T_DWORD emPerfMeasAppGetNumOf (
    EC_T_DWORD dwInstanceID,
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD *pdwNumOf
)
    Reset number of application performance measurement.
```

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pvPerfMeas** – [in] PerfMeas object or EC\_NULL to get the number of performance measurements in all PerfMeas objects
- **pdwNumOf** – [out] Number of performance measurements

**Returns***EC\_E\_NOERROR* or an error code**emPerfMeasAppGetInfo**

```
static EC_T_DWORD ecatPerfMeasAppGetInfo (
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD dwIndex,
    EC_T_PERF_MEAS_INFO *pPerfMeasInfo,
    EC_T_DWORD dwPerfMeasNumOf
)
EC_T_DWORD emPerfMeasAppGetInfo (
    EC_T_DWORD dwInstanceID,
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD dwIndex,
    EC_T_PERF_MEAS_INFO *pPerfMeasInfo,
    EC_T_DWORD dwPerfMeasNumOf
)
    Get general info about one/all application performance measurement.
```

**Parameters**

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

- **pvPerfMeas** – [in] PerfMeas object or EC\_NULL to use continuous index
- **dwIndex** – [in] Index of the performance measurement information, use EC\_PERF\_MEAS\_ALL to get all
- **pPerfMeasInfo** – [out] Pointer to a buffer receiving one/all performance measurement information
- **dwPerfMeasNumOf** – [in] Number of elements allocated in pPerfMeasInfo

**Returns***EC\_E\_NOERROR* or an error codestruct **EC\_T\_PERF\_MEAS\_INFO****Public Members**EC\_T\_CHAR **szName**[MAX\_STD\_STRLEN]  
Name of the benchmarkEC\_T\_UINT64 **qwFrequency**  
Frequency in Hz used by the timer*EC\_T\_USER\_JOB* **eUserJob**  
UserJob associated with the benchmarkEC\_T\_DWORD **dwBinCountHistogram**  
length of Histogram BinsEC\_T\_DWORD **dwFlags**  
Flags associated with the benchmark (See EC\_T\_PERF\_MEAS\_FLAG...)**emPerfMeasAppGetRaw**

```
static EC_T_DWORD ecatPerfMeasAppGetRaw (
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD dwIndex,
    EC_T_PERF_MEAS_VAL *pPerfMeasVal,
    EC_T_PERF_MEAS_HISTOGRAM *pPerfMeasHistogram,
    EC_T_DWORD dwPerfMeasNumOf
)
EC_T_DWORD emPerfMeasAppGetRaw (
    EC_T_DWORD dwInstanceID,
    EC_T_VOID *pvPerfMeas,
    EC_T_DWORD dwIndex,
    EC_T_PERF_MEAS_VAL *pPerfMeasVal,
    EC_T_PERF_MEAS_HISTOGRAM *pPerfMeasHistogram,
    EC_T_DWORD dwPerfMeasNumOf
)
Get raw data of one/all application performance measurement.
```

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pvPerfMeas** – [in] PerfMeas object or EC\_NULL to use continuous index

- **dwIndex** – [in] Index of the performance measurement, use EC\_PERF\_MEAS\_ALL to get all
- **pPerfMeasVal** – [out] Pointer to a buffer receiving one/all performance measurement values or EC\_NULL
- **pPerfMeasHistogram** – [out] Pointer to a buffer receiving one/all performance measurement histograms or EC\_NULL
- **dwPerfMeasNumOf** – [in] Number of elements allocated in pPerfMeasVal and pPerfMeasHistogram

**Returns***EC\_E\_NOERROR* or an error codestruct **EC\_T\_PERF\_MEAS\_VAL****Public Members**EC\_T\_UINT64 **qwCurrTicks**  
[ticks]EC\_T\_UINT64 **qwMinTicks**  
[ticks]EC\_T\_UINT64 **qwMaxTicks**  
[ticks]EC\_T\_UINT64 **qwAvgTicks**  
[ticks]struct **EC\_T\_PERF\_MEAS\_HISTOGRAM****Public Members**EC\_T\_DWORD \***aBins**

Histogram Bins:

The first bin is used for times below dwMinTicks. The last bin is used for times equal and above dwMaxTicks. All other times are stored in dwBinCount – 2 bins of equal size.

With e.g. dwBinCount = 202

qwMinTicks corresponding to 500us

qwMaxTicks corresponding to 1500us

aBins[0]: (-inf, 500us) corresponds to -inf < x < 500us

aBins[1]: [500us, 505us) corresponds to 500us <= x < 505us

aBins[2]: [505us, 510us) corresponds to 505us <= x < 510us

...

aBins[199]: [1490us, 1495us) corresponds to 1490us <= x < 1495us

aBins[200]: [1495us, 1500us) corresponds to 1495us <= x < 1500us

aBins[201]: [1500us, inf+) corresponds to 1500us <= x < inf+

**EC\_T\_DWORD dwBinCount**  
length of aBins

**EC\_T\_UINT64 qwMinTicks**  
results below qwMinTicks are stored in the first bin

**EC\_T\_UINT64 qwMaxTicks**  
results above qwMaxTicks are stored in the last bin

## emPerfMeasResetByTaskId

```
static EC_T_DWORD ecatPerfMeasResetByTaskId(
    EC_T_DWORD dwTaskId,
    EC_T_DWORD dwIndex
)
EC_T_DWORD emPerfMeasResetByTaskId(
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwTaskId,
    EC_T_DWORD dwIndex
)
    Reset internal performance measurement.
```

### Parameters

- **dwInstanceID** – [in] Master Instance ID
- **dwTaskId** – [in] Task Job ID
- **dwIndex** – [in] Index of the performance measurement, use EC\_PERF\_MEAS\_ALL to reset all

### Returns

*EC\_E\_NOERROR* or an error code

## emPerfMeasGetNumOfByTaskId

```
static EC_T_DWORD ecatPerfMeasGetNumOfByTaskId(
    EC_T_DWORD dwTaskId,
    EC_T_DWORD *pdwNumOf
)
EC_T_DWORD emPerfMeasGetNumOfByTaskId(
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwTaskId,
    EC_T_DWORD *pdwNumOf
)
    Reset number of internal performance measurement.
```

### Parameters

- **dwInstanceID** – [in] Master Instance ID
- **dwTaskId** – [in] Task Job ID
- **pdwNumOf** – [out] Number of performance measurements

### Returns

*EC\_E\_NOERROR* or an error code

**emPerfMeasGetInfoByTaskId**

```
static EC_T_DWORD ecatPerfMeasGetInfoByTaskId(
    EC_T_DWORD dwTaskId,
    EC_T_DWORD dwIndex,
    EC_T_PERF_MEAS_INFO *pPerfMeasInfo,
    EC_T_DWORD dwPerfMeasNumOf
)
EC_T_DWORD emPerfMeasGetInfoByTaskId(
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwTaskId,
    EC_T_DWORD dwIndex,
    EC_T_PERF_MEAS_INFO *pPerfMeasInfo,
    EC_T_DWORD dwPerfMeasNumOf
)
) Get general info about one/all internal performance measurement.
```

**Parameters**

- **dwInstanceID** – [in] Master Instance ID
- **dwTaskId** – [in] Task Job ID
- **dwIndex** – [in] Index of the performance measurement, use EC\_PERF\_MEAS\_ALL to get all
- **pPerfMeasInfo** – [out] Pointer to a buffer receiving one/all performance measurement infos
- **dwPerfMeasNumOf** – [in] Number of elements allocated in pPerfMeasInfo

**Returns***EC\_E\_NOERROR* or an error code**emPerfMeasGetRawByTaskId**

```
static EC_T_DWORD ecatPerfMeasGetRawByTaskId(
    EC_T_DWORD dwTaskId,
    EC_T_DWORD dwIndex,
    EC_T_PERF_MEAS_VAL *pPerfMeasVal,
    EC_T_PERF_MEAS_HISTOGRAM *pPerfMeasHistogram,
    EC_T_DWORD dwPerfMeasNumOf
)
EC_T_DWORD emPerfMeasGetRawByTaskId(
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwTaskId,
    EC_T_DWORD dwIndex,
    EC_T_PERF_MEAS_VAL *pPerfMeasVal,
    EC_T_PERF_MEAS_HISTOGRAM *pPerfMeasHistogram,
    EC_T_DWORD dwPerfMeasNumOf
)
) Get raw data of one/all application performance measurement.
```

**Parameters**

- **dwInstanceID** – [in] Master Instance ID
- **dwTaskId** – [in] Task Job ID

- **dwIndex** – [in] Index of the performance measurement, use EC\_PERF\_MEAS\_ALL to get all
- **pPerfMeasVal** – [out] Pointer to a buffer receiving one/all performance measurement values or EC\_NULL
- **pPerfMeasHistogram** – [out] Pointer to a buffer receiving one/all performance measurement histograms or EC\_NULL
- **dwPerfMeasNumOf** – [in] Number of elements allocated in pPerfMeasVal and pPerfMeasHistogram

**Returns***EC\_E\_NOERROR* or an error code

## 6.9 EtherCAT Mailbox Transfer

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

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

Typical mailbox transfer sequence:

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

```
MbxTferDesc.dwMaxDataLen = 10

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

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

2. **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.**

```
OsMemcpy (pMbxTfer->pbyMbxTferData, "0123456789", 10);

pMbxTfer->dwTferId = 1;

pMbxTfer->dwClntId = dwClntId;

pMbxTfer->dwDataLen=10;

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

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

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

**transfer object is set to TferDone prior to calling emNotify().**

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

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

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

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

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

```
emMbxTferDelete(pMbxTfer);

OsFree(MbxTferDesc.pbyMbxTferDescData);
```

### 6.9.1 Mailbox transfer object states

The following states exist for a mailbox transfer object:

enum **EC\_T\_MBXTFER\_STATUS**

*Values:*

enumerator **eMbxTferStatus\_Idle**  
Mailbox transfer object not in use

enumerator **eMbxTferStatus\_Pend**  
Mailbox transfer in process

enumerator **eMbxTferStatus\_TferDone**  
Mailbox transfer completed

enumerator **eMbxTferStatus\_TferReqError**  
Mailbox transfer request error

enumerator **eMbxTferStatus\_TferWaitingForContinue**  
Mailbox transfer waiting for continue, object owned by application

A mailbox transfer will be processed by the 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.

After completion of the mailbox transfer (with timeout and the client may finally set the transfer object into the state **EC\_T\_MBXTFER\_STATUS::eMbxTferStatus\_Idle**. New mailbox transfers can only be requested if the object is in the state **EC\_T\_MBXTFER\_STATUS::eMbxTferStatus\_Idle**.

**See also:**

[emMbxTferAbort \(\)](#)

## 6.9.2 emMbxTferCreate

static ***EC\_T\_MBXTFER*** \****ecatMbxTferCreate*** (***EC\_T\_MBXTFER\_DESC*** \****pMbxTferDesc***)

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

Creates a mailbox transfer object.

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

### Parameters

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

### Returns

- Pointer to the created mailbox transfer object if successful
- ***EC\_NULL*** on error (No memory left)

struct ***EC\_T\_MBXTFER\_DESC***

### Public Members

#### ***EC\_T\_DWORD dwMaxDataLen***

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

#### ***EC\_T\_BYT\* pbyMbxTferDescData***

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

struct ***EC\_T\_MBXTFER***

### Public Members

#### ***EC\_T\_DWORD dwClnId***

[] Client ID

#### ***EC\_T\_MBXTFER\_DESC MbxTferDesc***

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

#### ***EC\_T\_MBXTFER\_TYPE eMbxTferType***

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

#### ***EC\_T\_DWORD dwDataLen***

[] Amount of data bytes for the next mailbox transfer. If the mailbox transfer does not transfer data from

or to the slave this parameter will be ignored. This element has to be set to an appropriate value every time prior to initiate a new request. When the transfer is completed (emNotify) this value will contain the amount of data that was actually transferred

**EC\_T\_BYT<sub>E</sub> \*pbyMbxTferData**

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

**EC\_T\_MBXTFER\_STATUS eTferStatus**

[out] Transfer state. After a new transfer object is created the state will be set to eMbxTferStatus\_Idle

**EC\_T\_DWORD dwErrorCode**

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

**EC\_T\_DWORD dwTferId**

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

**EC\_T\_MBX\_DATA MbxData**

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

enum **EC\_T\_MBXTFER\_TYPE**

*Values:*

enumerator **eMbxTferType\_COE\_SDO\_DOWNLOAD**  
CoE SDO download

enumerator **eMbxTferType\_COE\_SDO\_UPLOAD**  
CoE SDO upload

enumerator **eMbxTferType\_COE\_GETODLIST**  
CoE Get object dictionary list

enumerator **eMbxTferType\_COE\_GETOBDESC**  
CoE Get object description

enumerator **eMbxTferType\_COE\_GETENTRYDESC**  
CoE Get object entry description

enumerator **eMbxTferType\_COE\_EMERGENCY**  
CoE emergency request

enumerator **eMbxTferType\_COE\_RX\_PDO**  
CoE RxPDO

enumerator **eMbxTferType\_FOE\_FILE\_UPLOAD**  
FoE upload

enumerator **eMbxTferType\_FOE\_FILE\_DOWNLOAD**  
FoE download

enumerator **eMbxTferType\_SOE\_READREQUEST**  
SoE read request

enumerator **eMbxFerType\_SOE\_READRESPONSE**  
SoE read response

enumerator **eMbxFerType\_SOE\_WRITEREQUEST**  
SoE write request

enumerator **eMbxFerType\_SOE\_WRITERESPONSE**  
SoE write response

enumerator **eMbxFerType\_SOE\_NOTIFICATION**  
SoE notification

enumerator **eMbxFerType\_SOE\_EMERGENCY**  
SoE emergency

enumerator **eMbxFerType\_VOE\_MBX\_READ**  
VoE read

enumerator **eMbxFerType\_VOE\_MBX\_WRITE**  
VoE write

enumerator **eMbxFerType\_AOE\_READ**  
AoE read

enumerator **eMbxFerType\_AOE\_WRITE**  
AoE write

enumerator **eMbxFerType\_AOE\_READWRITE**  
AoE read/write

enumerator **eMbxFerType\_AOE\_WRITECONTROL**  
AoE write control

enumerator **eMbxFerType\_RAWMBX**  
Raw mbx

enumerator **eMbxFerType\_FOE\_SEG\_DOWNLOAD**  
FoE segmented download

enumerator **eMbxFerType\_FOE\_SEG\_UPLOAD**  
FoE segmented upload

enumerator **eMbxFerType\_S2SMBX**  
S2S mbx

enumerator **eMbxFerType\_FOE\_UPLOAD\_REQ**  
FoE upload request

enumerator **eMbxFerType\_FOE\_DOWNLOAD\_REQ**  
FoE download request

enumerator **eMbxFerType\_EOE\_SEND\_FRAME**  
EoE send frame

enumerator **eMbxTferType\_EOE\_RECEIVE\_FRAME**  
EoE receive frame

enumerator **eMbxTferType\_EOE\_SET\_IP**  
EoE set IP address

union **EC\_T\_MBX\_DATA**  
*#include <EcInterfaceCommon.h>*

## Public Members

**EC\_T\_AOE\_CMD\_RESPONSE AoE\_Response**  
AoE

**EC\_T\_MBX\_DATA\_COE CoE**  
CoE

**EC\_T\_COE\_ODLIST CoE\_ODList**  
CoE Object Dictionary list

**EC\_T\_COE\_OBDESC CoE\_ObDesc**  
CoE object description

**EC\_T\_COE\_ENTRYDESC CoE\_EntryDesc**  
CoE entry description

**EC\_T\_COE\_EMERGENCY CoE\_Emergency**  
CoE emergency data

**EC\_T\_MBX\_DATA\_COE\_INITCMD CoE\_InitCmd**  
CoE InitCmd

**EC\_T\_MBX\_DATA\_FOE FoE**  
FoE

**EC\_T\_MBX\_DATA\_FOE\_REQ FoE\_Request**  
FoE request

**EC\_T\_MBX\_DATA\_SOE SoE**  
SoE

**EC\_T\_SOE\_NOTIFICATION SoE\_Notification**  
SoE notification request

**EC\_T\_SOE\_EMERGENCY SoE\_Emergency**  
SoE emergency request

### See also:

EC-Master Class A about AoE, FoE and VoE mailbox protocols.

### 6.9.3 emMbxTferAbort

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

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

#### Parameters

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

#### Returns

*EC\_E\_NOERROR* if successful

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

### 6.9.4 emMbxTferDelete

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

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

#### Parameters

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

#### Returns

*EC\_E\_NOERROR* or error code

### 6.9.5 emNotify - EC\_NOTIFY\_MBOXRCV

Indicates a mailbox transfer completion.

#### emNotify - EC\_NOTIFY\_MBOXRCV

##### Parameter

- **pbyInBuf**: [in] Pointer to a structure of type *EC\_T\_MBXTFER*, contains the corresponding mailbox transfer object.
- **dwInBufSize**: [in] Size of the transfer object provided at pbyInBuf in bytes.
- **pbyOutBuf**: [out] Should be set to *EC\_NULL*
- **dwOutBufSize**: [in] Should be set to 0
- **pdwNumOutData**: [out] Should be set to *EC\_NULL*

The element *EC\_T\_MBXTFER*::*dwClntId* contains the corresponding ID of the client that is notified, the corresponding transfer ID can be found in *EC\_T\_MBXTFER*::*dwTferId*. The transfer result is stored in *EC\_T\_MBXTFER*::*dwErrorCode*.

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

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

## 6.10 Automation Device Specification over EtherCAT (AoE)

The AoE protocol is used to access the Object dictionary of slave devices of underlying field-buses, e.g. for a CAN application protocol Slave connected to a EtherCAT-CANopen gateway device. It is also used in relation with the EtherCAT Automation Protocol (EAP).

Reference:

- ETG.1020 -> AoE

### Current limitations

- Fragmented AoE access is not yet implemented

### 6.10.1 emAoeGetSlaveNetId

```
static EC_T_DWORD ecatAoeGetSlaveNetId (
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poAoeNetId
)
EC_T_DWORD emAoeGetSlaveNetId (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poAoeNetId
)
    Retrieve the NetID of a specific EtherCAT device.
```

#### Parameters

- `dwInstanceID` – [in] Instance ID (Multiple EtherCAT Network Support)
- `dwSlaveId` – [in] Slave ID
- `poAoeNetId` – [out] AoE NetID of the corresponding slave

#### Returns

- `EC_E_NOERROR` if successful
- `EC_E_INVALIDSTATE` if master isn't initialized
- `EC_E_INVALIDPARAM` if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- `EC_E_SLAVE_NOT_PRESENT` if slave not present
- `EC_E_NOTFOUND` if no slave matching dwSlaveId can be found
- `EC_E_NO_MBX_SUPPORT` if slave has no mailbox support
- `EC_E_ADS_IS_RUNNING` if ADS server is running

struct `EC_T_AOE_NETID`

**Public Members**

**EC\_T\_BYTE aby[6]**  
AoE net id

See also:

*emGetSlaveId()*

### 6.10.2 emAoeRead

```
static EC_T_DWORD ecatAoeRead (
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwDataOutLen,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emAoeRead (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwDataOutLen,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)
```

Execute a AoE mailbox read request to an EtherCAT slave device.

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

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID.
- **wTargetPort** – [in] Target port.
- **dwIndexGroup** – [in] AoE read command index group.
- **dwIndexOffset** – [in] AoE read command index offset
- **dwDataLen** – [in] Buffer length in bytes
- **pbyData** – [out] Buffer receiving transferred data
- **pdwDataOutLen** – [out] Number of bytes read from the target device

- **pdwErrorCode** – [out] AoE response error code
- **pdwCmdResult** – [out] AoE read command result code
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

**Returns**

- *EC\_E\_NOERROR*
- *EC\_E\_AOE\_VENDOR\_SPECIFIC*: will be returned in case the AoE device has responded with an user defined error code
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive

**See also:**

- *emAoeReadReq()*
- *emGetSlaveId()*

**6.10.3 emAoeReadReq**

```
static EC_T_DWORD ecatAoeReadReq (
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emAoeReadReq (
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwTimeout
)
Execute a non-blocking AoE mailbox read request to an EtherCAT slave device.
```

If the functions returns *EC\_E\_AOE\_DEVICE\_XXXX* (well known device errors) or *EC\_E\_AOE\_VENDOR\_SPECIFIC* the original ADS error can be retrieved from the mailbox transfer object at *EC\_T\_MBXTFER.MbxData.AoE\_Response*. A unique transfer ID must be written into *EC\_T\_MBXTFER.dwTferId*.

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID.
- **wTargetPort** – [in] Target port.
- **dwIndexGroup** – [in] AoE read command index group

- **dwIndexOffset** – [in] AoE read command index offset
- **dwTimeout** – [in] Timeout [ms]

#### Returns

- *EC\_E\_NOERROR*
- *EC\_E\_AOE\_VENDOR\_SPECIFIC*: will be returned in case the AoE device has responded with an user defined error code
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive

struct **EC\_T\_AOE\_CMD\_RESPONSE**

#### Public Members

**EC\_T\_DWORD dwErrorCode**  
AoE response error code

**EC\_T\_DWORD dwCmdResult**  
AoE command result code

#### See also:

[emGetSlaveId \(\)](#)

## 6.10.4 emNotify - eMbxTferType\_AOE\_READ

**emNotify - eMbxTferType\_AOE\_READ**

#### Parameter

- pbyInBuf: [in] pMbxTfer - Pointer to a structure of type EC\_T\_MBXTFER, this structure contains the used mailbox transfer object. This mailbox transfer object also contains AoE device error codes in case of an AoE access error.
- dwInBufSize: [in] Size of the transfer object.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### See also:

[emAoeReadReq \(\)](#)

## 6.10.5 emAoeWrite

```
static EC_T_DWORD ecatAoeWrite (
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emAoeWrite (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)
```

Execute a AoE mailbox write request to an EtherCAT slave device.

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

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID
- **wTargetPort** – [in] Target port
- **dwIndexGroup** – [in] AoE write command index group
- **dwIndexOffset** – [in] AoE write command index offset
- **dwDataLen** – [in] Buffer length in bytes
- **pbyData** – [in] Buffer containing transferred data
- **pdwErrorCode** – [out] Pointer to AoE response error code
- **pdwCmdResult** – [out] Pointer to AoE write command result code
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

### Returns

- ***EC\_E\_NOERROR***
- ***EC\_E\_AOE\_VENDOR\_SPECIFIC***: will be returned in case the AoE device has responded with an user defined error code
- ***EC\_E\_MASTER\_RED\_STATE\_INACTIVE*** if Master Redundancy is configured and master is inactive

**See also:**

- `emAoeWriteReq()`
- `emGetSlaveId()`

### 6.10.6 emAoeWriteReq

```
static EC_T_DWORD ecatAoeWriteReq(  
    EC_T_MBXTFER *pMbxTfer,  
    EC_T_DWORD dwSlaveId,  
    EC_T_AOE_NETID *poTargetNetId,  
    EC_T_WORD wTargetPort,  
    EC_T_DWORD dwIndexGroup,  
    EC_T_DWORD dwIndexOffset,  
    EC_T_DWORD dwTimeout  
)  
  
EC_T_DWORD emAoeWriteReq(  
    EC_T_DWORD dwInstanceID,  
    EC_T_MBXTFER *pMbxTfer,  
    EC_T_DWORD dwSlaveId,  
    EC_T_AOE_NETID *poTargetNetId,  
    EC_T_WORD wTargetPort,  
    EC_T_DWORD dwIndexGroup,  
    EC_T_DWORD dwIndexOffset,  
    EC_T_DWORD dwTimeout  
)
```

Execute a non-blocking AoE mailbox write request to an EtherCAT slave device.

If the functions returns `EC_E_AOE_DEVICE_XXXX` (well known device errors) or `EC_E_AOE_VENDOR_SPECIFIC` the original ADS error can be retrieved from the mailbox transfer object at `EC_T_MBXTFER.MbxData.AoE_Response`. A unique transfer ID must be written into `EC_T_MBXTFER.dwTferId`.

**Parameters**

- **`dwInstanceID`** – [in] Instance ID (Multiple EtherCAT Network Support)
- **`pMbxTfer`** – [in] Mailbox transfer object
- **`dwSlaveId`** – [in] Slave ID
- **`poTargetNetId`** – [in] Target NetID.
- **`wTargetPort`** – [in] Target port.
- **`dwIndexGroup`** – [in] AoE write command index group.
- **`dwIndexOffset`** – [in] AoE write command index offset
- **`dwTimeout`** – [in] Timeout [ms]

**Returns**

- `EC_E_NOERROR`
- `EC_E_AOE_VENDOR_SPECIFIC`: will be returned in case the AoE device has responded with an user defined error code
- `EC_E_MASTER_RED_STATE_INACTIVE` if Master Redundancy is configured and master is inactive

**See also:**`emGetSlaveId()`

### **6.10.7 emNotify - eMbxTferType\_AOE\_WRITE**

#### **emNotify - eMbxTferType\_AOE\_WRITE**

**Parameter**

- `pbyInBuf`: [in] `pMbxTfer` - Pointer to a structure of type `EC_T_MBXTFER`, this structure contains the used mailbox transfer object. This mailbox transfer object also contains AoE device error codes in case of an AoE access error.
- `dwInBufSize`: [in] Size of the transfer object.
- `pbyOutBuf`: [out] Should be set to `EC_NULL`
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to `EC_NULL`

**See also:**`emAoeWriteReq()`

### **6.10.8 emAoeReadWrite**

```
static EC_T_DWORD ecatAoeReadWrite (
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwReadDataLen,
    EC_T_DWORD dwWriteDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwDataOutLen,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emAoeReadWrite (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_DWORD dwIndexGroup,
    EC_T_DWORD dwIndexOffset,
    EC_T_DWORD dwReadDataLen,
    EC_T_DWORD dwWriteDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwDataOutLen,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)
    Execute a AoE mailbox read/write request to an EtherCAT slave device.
```

## Parameters

- **`dwInstanceID`** – [in] Instance ID (Multiple EtherCAT Network Support)
- **`dwSlaveId`** – [in] Slave ID
- **`poTargetNetId`** – [in] Target NetID
- **`wTargetPort`** – [in] Target port
- **`dwIndexGroup`** – [in] AoE read/write command index group.
- **`dwIndexOffset`** – [in] AoE read/write command index offset
- **`dwReadDataLen`** – [in] Number of bytes to read from the target device.
- **`dwWriteDataLen`** – [in] Number of bytes to read from the target device
- **`pbyData`** – [in, out] Buffer containing and receiving transferred data
- **`pdwDataOutLen`** – [out] Number of bytes read from the target device
- **`pdwErrorCode`** – [out] Pointer to AoE response error code
- **`pdwCmdResult`** – [out] Pointer to AoE write command result code
- **`dwTimeout`** – [in] Timeout [ms]. The function will block at most for this time. EC\_NOWAIT is not valid.

## Returns

- **`EC_E_NOERROR`**
- **`EC_E_AOE_VENDOR_SPECIFIC`**: will be returned in case the AoE device has responded with an user defined error code
- **`EC_E_MASTER_RED_STATE_INACTIVE`** if Master Redundancy is configured and master is inactive

## See also:

[emGetSlaveId \(\)](#)

## 6.10.9 emAoeWriteControl

```
static EC_T_DWORD ecatAoeWriteControl (
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_WORD wAoEState,
    EC_T_WORD wDeviceState,
    EC_T_DWORD dwDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)
```

```
EC_T_DWORD emAoeWriteControl (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_AOE_NETID *poTargetNetId,
    EC_T_WORD wTargetPort,
    EC_T_WORD wAoEState,
    EC_T_WORD wDeviceState,
    EC_T_DWORD dwDataLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD *pdwErrorCode,
    EC_T_DWORD *pdwCmdResult,
    EC_T_DWORD dwTimeout
)
```

) Execute a AoE mailbox write control request to an EtherCAT slave device.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID. The Target NetID of a AoE slave device can be retrieved by ecatAoeGetSlaveNetId()..
- **wTargetPort** – [in] Target port
- **wAoEState** – [in] AoE state
- **wDeviceState** – [in] Device specific state
- **dwDataLen** – [in] Buffer length in bytes
- **pbyData** – [in] Buffer containing transferred data
- **pdwErrorCode** – [out] Pointer to AoE response error code
- **pdwCmdResult** – [out] Pointer to AoE write command result code
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. EC\_NOWAIT is not valid.

#### Returns

- **EC\_E\_NOERROR**
- **EC\_E\_AOE\_VENDOR\_SPECIFIC**: will be returned in case the AoE device has responded with an user defined error code
- **EC\_E\_MASTER\_RED\_STATE\_INACTIVE** if Master Redundancy is configured and master is inactive

#### See also:

[emGetSlaveId\(\)](#)

### 6.10.10 emConvertEcErrorToAdsError

`EC_T_DWORD ecatConvertEcErrorToAdsError (EC_T_DWORD dwErrorCode)`

```
EC_T_DWORD emConvertEcErrorToAdsError (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwErrorCode
)
    Convert master error code to AoE error code.
```

**Returns**

AoE error code

## 6.11 CAN application protocol over EtherCAT (CoE)

### 6.11.1 emCoeSdoDownload

```
static EC_T_DWORD ecatCoeSdoDownload (
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYT byObSubIndex,
    EC_T_BYT *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout,
    EC_T_DWORD dwFlags
)
EC_T_DWORD emCoeSdoDownload (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYT byObSubIndex,
    EC_T_BYT *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout,
    EC_T_DWORD dwFlags
)
    Execute a CoE SDO download to an EtherCAT slave device.
```

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

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object Index
- **byObSubIndex** – [in] Object SubIndex. If Complete Access only 0 or 1 allowed
- **pbyData** – [in] Buffer containing transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **dwTimeout** – [in] Timeout [ms]
- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC\_MAILBOX\_FLAG\_SDO\_COMPLETE).

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- *EC\_E\_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support
- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_ADS\_IS\_RUNNING* if ADS server is running
- *CoE SDO error code*

**emCoeSdoDownload Example**

The following code demonstrates how to download a CoE object to a slave using the API `emCoeSdoDownload()`.

```
EC_T_DWORD CoeSdoDownloadExample()
{
    EC_T_DWORD dwRetVal = EC_E_ERROR;
    EC_T_DWORD dwRes = EC_E_ERROR;

    /* slave with station address 2002 used for CoE SDO download */
    EC_T_DWORD     dwSlaveId = ecatGetSlaveId(2002);
    EC_T_BYTEx     abyBuf[4];
    OsMemset(abyBuf, 0, sizeof(abyBuf));

    /* download value 123 to object index 0x40A2, subindex 2, timeout 5s, no
     * complete access */
    EC_SET_FRM_DWORD(abyBuf, 123);
    dwRes = ecatCoeSdoDownload(dwSlaveId, 0x40A2, 2, abyBuf, (EC_T_
    →DWORD)sizeof(abyBuf), 5000, 0);
    if (EC_E_NOERROR != dwRes)
    {
        dwRetVal = dwRes;
        goto Exit;
    }

    dwRetVal = EC_E_NOERROR;
Exit:
    return dwRetVal;
}
```

**See also:**

`emGetSlaveId()`

## 6.11.2 emCoeSdoDownloadReq

```
static 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
)
EC_T_DWORD emCoeSdoDownloadReq (
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTE byObSubIndex,
    EC_T_DWORD dwTimeout,
    EC_T_DWORD dwFlags
)
```

Initiates a CoE SDO download to an EtherCAT slave device and returns immediately.

The length of the data to be downloaded must be set in [EC\\_T\\_MBXTFER.dwDataLen](#). A unique transfer ID must be written into [EC\\_T\\_MBXTFER.dwTferId](#). EC\_NOTIFY\_MBOXRCV is given on completion.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object Index
- **byObSubIndex** – [in] Object SubIndex. If Complete Access only 0 or 1 allowed
- **dwTimeout** – [in] Timeout [ms]
- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC\_MAILBOX\_FLAG\_SDO\_COMPLETE).

### Returns

- [EC\\_E\\_NOERROR](#) if successful
- [EC\\_E\\_INVALIDSTATE](#) if master isn't initialized
- [EC\\_E\\_INVALIDPARM](#) if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- [EC\\_E\\_NOMEMORY](#) if the mailbox protocol queue of the slave is full
- [EC\\_E\\_SLAVE\\_NOT\\_PRESENT](#) if slave not present
- [EC\\_E\\_NOTFOUND](#) if no slave matching dwSlaveId can be found
- [EC\\_E\\_NO\\_MBX\\_SUPPORT](#) if slave has no mailbox support
- [EC\\_E\\_INVALID\\_SLAVE\\_STATE](#) if slave is in an invalid state for mailbox transfer
- [EC\\_E\\_MASTER\\_RED\\_STATE\\_INACTIVE](#) if Master Redundancy is configured and master is inactive
- [EC\\_E\\_ADS\\_IS\\_RUNNING](#) if ADS server is running
- [CoE SDO error code](#)

## emCoeSdoDownloadReq Example

The following code demonstrates how to download a CoE object to a slave using the non-blocking API `emCoeSdoDownloadReq()`.

In this example the mailbox transfer object state is polled. `emNotify - EC_NOTIFY_MBOXRCV` can be used as alternative to polling.

```
EC_T_DWORD CoeSdoDownloadReqExample()
{
    EC_T_DWORD dwRetVal = EC_E_ERROR;
    EC_T_DWORD dwRes = EC_E_ERROR;

    /* needed only for notifications, see
     - ecatRegisterClient(...),
     - pAppContext->pNotificationHandler->GetClientID() */
    EC_T_DWORD dwClientId = 0;

    /* slave with station address 2002 used for CoE SDO download */
    EC_T_DWORD dwSlaveId = ecatGetSlaveId(2002);

    EC_T_MBXTFER* pMbxTferObject = EC_NULL;
    EC_T_BYTEx abyBuf[4];
    EC_T_MBXTFER_DESC oObjectDataTferDesc;
    OsMemset(abyBuf, 0, sizeof(abyBuf));
    OsMemset(&oObjectDataTferDesc, 0, sizeof(EC_T_MBXTFER_DESC));

    /* create mailbox transfer object */
    oObjectDataTferDesc.pbyMbxTferDescData = abyBuf;
    oObjectDataTferDesc.dwMaxDataLen = sizeof(abyBuf);
    pMbxTferObject = ecatMbxTferCreate(&oObjectDataTferDesc);
    if (EC_NULL == pMbxTferObject)
    {
        dwRes = EC_E_NOMEMORY;
        dwRetVal = dwRes;
        goto Exit;
    }

    /* create mailbox transfer object */
    pMbxTferObject->dwClntId = dwClientId;
    pMbxTferObject->dwTferId = 1; /* assigned by application. should be unique for_
→each object */

    /* request download of value 123 to object index 0x40A2, subindex 2, timeout_
→5s, no complete access */
    EC_SET_FRM_DWORD(abyBuf, 123);
    dwRes = ecatCoeSdoDownloadReq(pMbxTferObject, dwSlaveId, 0x40A2, 2, 5000, 0);
    if (EC_E_NOERROR != dwRes)
    {
        dwRetVal = dwRes;
        goto Exit;
    }

    /* wait for transfer finished */
    while (eMbxTferStatus_Pend == pMbxTferObject->eTferStatus)
    {
        OsSleep(10);
    }

    /* check transfer result */
    dwRes = pMbxTferObject->dwErrorCode;
    if (EC_E_NOERROR != dwRes)
```

(continues on next page)

(continued from previous page)

```

{
    dwRetVal = dwRes;
    goto Exit;
}

dwRetVal = EC_E_NOERROR;
Exit:
    if (EC_NULL != pMbxTferObject)
    {
        pMbxTferObject->eTferStatus = eMbxTferStatus_Idle;
        ecatMbxTferDelete(pMbxTferObject);
    }

    return dwRetVal;
}

```

**See also:**

- *emNotify - eMbxTferType\_COE\_SDO\_DOWNLOAD*
- *emMbxTferCreate ()*
- *emGetSlaveId ()*

### **6.11.3 emNotify - eMbxTferType\_COE\_SDO\_DOWNLOAD**

SDO download transfer completion.

#### **emNotify - eMbxTferType\_COE\_SDO\_DOWNLOAD**

##### **Parameter**

- pbyInBuf: [in] Pointer to a structure of type EC\_T\_MBXTFER, this structure contains the corresponding mailbox transfer object.
- dwInBufSize: [in] Size of the transfer object pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

The corresponding transfer ID can be found in *EC\_T\_MBXTFER::dwTferId*. The transfer result is stored in *EC\_T\_MBXTFER::dwErrorCode*.

The request parameters stored in element *EC\_T\_MBX\_DATA::CoE* of type *EC\_T\_MBX\_DATA\_COE* are part of *EC\_T\_MBXTFER::MbxData* and may have to be buffered by the client. Access to the memory area *EC\_T\_MBXTFER::MbxData* outside of the notification caller context is illegal and the results are undefined.

struct **EC\_T\_MBX\_DATA\_COE**

**Public Members**

**EC\_T\_WORD wStationAddress**  
Station address of the slave

**EC\_T\_WORD wIndex**  
Object index

**EC\_T\_BYTEx bySubIndex**  
Object subindex

**EC\_T\_BOOL bCompleteAccess**  
Complete access

**6.11.4 emCoeSdoUpload**

```
static 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
)
EC_T_DWORD emCoeSdoUpload (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTE byObSubIndex,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD *pdwOutDataLen,
    EC_T_DWORD dwTimeout,
    EC_T_DWORD dwFlags
)
```

Execute a CoE SDO upload from an EtherCAT slave device to the master.

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

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **byObSubIndex** – [in] Object SubIndex. If Complete Access only 0 or 1 allowed
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **pdwOutDataLen** – [out] Length of received data in bytes
- **dwTimeout** – [in] Timeout [ms]

- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC\_MAILBOX\_FLAG\_SDO\_COMPLETE).

### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- *EC\_E\_NOMEMORY* if the mailbox protocol queue of the slave if full
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support
- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_ADS\_IS\_RUNNING* if ADS server is running
- *CoE SDO error code*

### emCoeSdoUpload Example

The following code demonstrates how to upload a CoE object from a slave using the API `emCoeSdoUpload()`.

```
EC_T_DWORD CoeSdoUploadExample()
{
    EC_T_DWORD dwRetVal = EC_E_ERROR;
    EC_T_DWORD dwRes = EC_E_ERROR;

    /* slave with station address 2002 used for CoE SDO upload */
    EC_T_DWORD     dwSlaveId = ecatGetSlaveId(2002);
    EC_T_BYTEx      abyBuf[4];
    OsMemset(abyBuf, 0, sizeof(abyBuf));

    /* upload object index 0x1018, subindex 1, timeout 5s, no complete access */
    dwRes = ecatCoeSdoUpload(dwSlaveId, 0x1018, 1, abyBuf, (EC_T_
    →DWORD)sizeof(abyBuf), EC_NULL, 5000, 0);
    if (EC_E_NOERROR != dwRes)
    {
        dwRetVal = dwRes;
        goto Exit;
    }

    dwRetVal = EC_E_NOERROR;
Exit:
    return dwRetVal;
}
```

### See also:

`emGetSlaveId()`

### 6.11.5 emCoeSdoUploadReq

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

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

The length of the data to be uploaded must be set in *EC\_T\_MBXTFER.dwDataLen*. A unique transfer ID must be written into *EC\_T\_MBXTFER.dwTferId*. EC\_NOTIFY\_MBOXRCV is given on completion.

#### Parameters

- **`dwInstanceID`** – [in] Instance ID (Multiple EtherCAT Network Support)
- **`pMbxTfer`** – [in] Mailbox transfer object created with emMbxTferCreate
- **`dwSlaveId`** – [in] Slave ID
- **`wObIndex`** – [in] Object Index
- **`byObSubIndex`** – [in] Object SubIndex. If Complete Access only 0 or 1 allowed
- **`dwTimeout`** – [in] Timeout [ms]
- **`dwFlags`** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC\_MAILBOX\_FLAG\_SDO\_COMPLETE).

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- *EC\_E\_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support
- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_ADS\_IS\_RUNNING* if ADS server is running
- *CoE SDO error code*

## emCoeSdoUploadReq Example

The following code demonstrates how to upload a CoE object from a slave using the non-blocking API `emCoeSdoUploadReq()`.

In this example the mailbox transfer object state is polled. `emNotify - EC_NOTIFY_MBOXRCV` can be used as alternative to polling.

```
EC_T_DWORD CoeSdoUploadReqExample()
{
    EC_T_DWORD dwRetVal = EC_E_ERROR;
    EC_T_DWORD dwRes = EC_E_ERROR;

    /* needed only for notifications, see
     - ecatRegisterClient(...)
     - pApplicationContext->pNotificationHandler->GetClientID() */
    EC_T_DWORD dwClientId = 0;

    /* slave with station address 2002 used for CoE SDO upload */
    EC_T_DWORD dwSlaveId = ecatGetSlaveId(2002);

    EC_T_MBXTFER* pMbxTferObject = EC_NULL;
    EC_T_BYTE abyBuf[4];
    EC_T_MBXTFER_DESC oObjectDataTferDesc;
    OsMemset(abyBuf, 0, sizeof(abyBuf));
    OsMemset(&oObjectDataTferDesc, 0, sizeof(EC_T_MBXTFER_DESC));

    /* create mailbox transfer object */
    oObjectDataTferDesc.pbyMbxTferDescData = abyBuf;
    oObjectDataTferDesc.dwMaxDataLen = sizeof(abyBuf);
    pMbxTferObject = ecatMbxTferCreate(&oObjectDataTferDesc);
    if (EC_NULL == pMbxTferObject)
    {
        dwRes = EC_E_NOMEMORY;
        dwRetVal = dwRes;
        goto Exit;
    }

    /* create mailbox transfer object */
    pMbxTferObject->dwClntId = dwClientId;
    pMbxTferObject->dwTferId = 1; /* assigned by application. should be unique for_
→each object */

    /* request upload of object index 0x1018, subindex 1, timeout 5s, no complete_
→access */
    dwRes = ecatCoeSdoUploadReq(pMbxTferObject, dwSlaveId, 0x1018, 1, 5000, 0);
    if (EC_E_NOERROR != dwRes)
    {
        dwRetVal = dwRes;
        goto Exit;
    }

    /* wait for transfer finished */
    while (eMbxTferStatus_Pend == pMbxTferObject->eTferStatus)
    {
        OsSleep(10);

        /* transfer can be canceled using ecatMbxTferAbort(...) if it takes too_
→long */
    }

    /* check transfer result */
}
```

(continues on next page)

(continued from previous page)

```

dwRes = pMbxTferObject->dwErrorCode;
if (EC_E_NOERROR != dwRes)
{
    dwRetVal = dwRes;
    goto Exit;
}

dwRetVal = EC_E_NOERROR;
Exit:
if (EC_NULL != pMbxTferObject)
{
    pMbxTferObject->eTferStatus = eMbxTferStatus_Idle;
    ecatMbxTferDelete(pMbxTferObject);
}

return dwRetVal;
}

```

**See also:**

- *emNotify - eMbxTferType\_COE\_SDO\_UPLOAD*
- *emMbxTferCreate ()*
- *emGetSlaveId ()*

### **6.11.6 emNotify - eMbxTferType\_COE\_SDO\_UPLOAD**

SDO upload transfer completion.

#### **emNotify - eMbxTferType\_COE\_SDO\_UPLOAD**

**Parameter**

- pbyInBuf: [in] Pointer to a structure of type EC\_T\_MBXTFER, contains the corresponding mailbox transfer object.
- dwInBufSize: [in] Size of the transfer object in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

The corresponding transfer ID can be found in *EC\_T\_MBXTFER::dwTferId*. The transfer result is stored in *EC\_T\_MBXTFER::dwErrorCode*.

The request parameters stored in element *EC\_T\_MBX\_DATA::CoE* of type *EC\_T\_MBX\_DATA\_COE* are part of *EC\_T\_MBXTFER::MbxData*. The SDO data stored in *EC\_T\_MBXTFER::pbyMbxTferData* may have to be buffered by the client. Access to the memory area referenced by *EC\_T\_MBXTFER::pbyMbxTferData* outside of the notification caller context is illegal and the results are undefined.

### 6.11.7 emCoeGetODListReq

```
static EC_T_DWORD ecatCoeGetODListReq (
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_COE_ODLIST_TYPE eListType,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emCoeGetODListReq (
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_COE_ODLIST_TYPE eListType,
    EC_T_DWORD dwTimeout
)
```

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

A unique transfer ID must be written into *EC\_T\_MBXTFER.dwTferId*. EC\_NOTIFY\_MBOXRCV is given on completion.

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

---

**Note:** The mailbox transfer object will receive the slave response containing the list type followed by the list itself. Therefore the buffer must be 2 bytes bigger than the expected list size.

---

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer
- **dwSlaveId** – [in] Slave ID
- **eListType** – [in] which object types shall be transferred
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. If the timeout value is set to EC\_NOWAIT the function will return immediately.

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- *EC\_E\_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support
- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_ADS\_IS\_RUNNING* if ADS server is running
- *CoE SDO error code*

enum **EC\_T\_COE\_ODLIST\_TYPE**  
*Values:*

enumerator **eODListType\_Lengths**  
 Lengths of each list type

enumerator **eODListType\_ALL**  
 List contains all objects

enumerator **eODListType\_RxPdoMap**  
 List with PDO mappable objects

enumerator **eODListType\_TxPdoMap**  
 List with objects that can be changed

enumerator **eODListType\_StoredFRep1**  
 Only stored for a device replacement objects

enumerator **eODListType\_StartupParm**  
 Only startup parameter objects

### emCoeGetODListReq Example

The following code demonstrates how to get the list of objects that are available in a slave using the non-blocking API *emCoeGetODListReq()*.

In this example the mailbox transfer object state is polled. *emNotify - EC\_NOTIFY\_MBOXRCV* can be used as alternative to polling.

```
EC_T_MBXTFER*      pMbxTfer = EC_NULL;
EC_T_MBXTFER_DESC oMbxTferDesc;

OsMemset (&oMbxTferDesc, 0, sizeof(EC_T_MBXTFER_DESC));
oMbxTferDesc.dwMaxDataLen = CROD_ODLTFER_SIZE;

/* allocate payload memory */
oMbxTferDesc.pbyMbxTferDescData = (EC_T_BYTE*)OsMalloc(oMbxTferDesc.
dwMaxDataLen);
if (EC_NULL == oMbxTferDesc.pbyMbxTferDescData)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}

/* create mailbox transfer object */
pMbxTfer = emMbxTferCreate(dwInstanceId, &oMbxTferDesc);
if (EC_NULL == pMbxTfer)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}

dwRes = emCoeGetODListReq(dwInstanceId, pMbxTfer, dwSlaveId, eODListType_ALL,
5000 /* timeout */);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
"emCoeGetODList: %s (0x%lx)\n", ecatGetText(dwRes), dwRes));
}
```

(continues on next page)

(continued from previous page)

```

    goto Exit;
}

/* wait for transfer finished, see also emMbxTferAbort(...) */
while (eMbxTferStatus_Pend == pMbxTfer->eTferStatus)
{
    OsSleep(10);
}

dwRetVal = pMbxTfer->dwErrorCode;
Exit:
if (EC_NULL != pMbxTfer)
{
    if (eMbxTferStatus_Pend == pMbxTfer->eTferStatus)
    {
        emMbxTferAbort(dwInstanceId, pMbxTfer);
    }
    emMbxTferDelete(dwInstanceId, pMbxTfer);
}

SafeOsFree(oMbxTferDesc.pbyMbxTferDescData);

```

**See also:**

- *emNotify - eMbxTferType\_COE\_GETODLIST*
- *emMbxTferCreate()*
- *emGetSlaveId()*
- See *CoeReadObjectDictionary()* in *EcSdoServices.cpp* as an example for *emCoeGetODListReq()*.

### 6.11.8 emNotify - eMbxTferType\_COE\_GETODLIST

Object dictionary list upload transfer completion.

#### **emNotify - eMbxTferType\_COE\_GETODLIST**

##### **Parameter**

- pbyInBuf: [in] Pointer to a structure of type EC\_T\_MBXTFER, contains the corresponding mailbox transfer object.
- dwInBufSize: [in] Size of the transfer object in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

The corresponding transfer ID can be found in *EC\_T\_MBXTFER::dwTferId*. The transfer result is stored in *EC\_T\_MBXTFER::dwErrorCode*.

The object list stored in element *EC\_T\_MBX\_DATA::CoE\_ODList* of type *EC\_T\_COE\_ODLIST* is part of *EC\_T\_MBXTFER::MbxData* and may have to be buffered by the client. Access to the memory area *EC\_T\_MBXTFER::MbxData* outside of the notification caller context is illegal and the results are undefined.

```
struct EC_T_COE_ODLIST
```

**Public Members**

**EC\_T\_COE\_ODLIST\_TYPE eOdListType**  
list type

**EC\_T\_WORD wLen**  
amount of object IDs

**EC\_T\_WORD wStationAddress**  
Station address of the slave

**EC\_T\_WORD \*pwOdList**  
array containing object IDs

### 6.11.9 emCoeGetObjectDescReq

```
static EC_T_DWORD ecatCoeGetObjectDescReq (
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emCoeGetObjectDescReq (
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_DWORD dwTimeout
)
```

Determines the description of a specific object.

A unique transfer ID must be written into **EC\_T\_MBXTFER.dwTferId**. EC\_NOTIFY\_MBOXRCV is given on completion. This function may not be called from within the JobTask's context.

**Parameters**

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

**Returns**

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master isn't initialized
- **EC\_E\_INVALIDPARAM** if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- **EC\_E\_NOMEMORY** if the mailbox protocol queue of the slave is full
- **EC\_E\_SLAVE\_NOT\_PRESENT** if slave not present

- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support
- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_ADS\_IS\_RUNNING* if ADS server is running
- *CoE SDO error code*

### emCoeGetObjectDescReq Example

The following code demonstrates how to determine the description of a specific object using the non-blocking API *emCoeGetObjectDescReq()*.

In this example the mailbox transfer object state is polled. *emNotify - EC\_NOTIFY\_MBOXRCV* can be used as alternative to polling.

```

EC_T_MBXTFER*      pMbxTfer = EC_NULL;
EC_T_MBXTFER_DESC oMbxTferDesc; /* mailbox transfer descriptor */

OsMemset(&oMbxTferDesc, 0, sizeof(EC_T_MBXTFER_DESC));
oMbxTferDesc.dwMaxDataLen = CROD_OBDESC_SIZE;

/* allocate payload memory */
oMbxTferDesc.pbyMbxTferDescData = (EC_T_BYTE*)OsMalloc(oMbxTferDesc.
→dwMaxDataLen);
if (EC_NULL == oMbxTferDesc.pbyMbxTferDescData)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}

/* create mailbox transfer object */
pMbxTfer = emMbxTferCreate(dwInstanceId, &oMbxTferDesc);
if (EC_NULL == pMbxTfer)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}
dwRes = emCoeGetObjectDescReq(dwInstanceId, pMbxTfer, dwSlaveId, 0x6411, 5000);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
→"emCoeGetObjectDescReq: %s (0x%lx)\n", ecatGetText(dwRes), dwRes));
    goto Exit;
}

/* wait for transfer finished, see also emMbxTferAbort(...) */
while (eMbxTferStatus_Pend == pMbxTfer->eTferStatus)
{
    OsSleep(10);
}

dwRetVal = pMbxTfer->dwErrorCode;
Exit:
if (EC_NULL != pMbxTfer)
{
    if (eMbxTferStatus_Pend == pMbxTfer->eTferStatus)
}

```

(continues on next page)

(continued from previous page)

```

    {
        emMbxTferAbort(dwInstanceId, pMbxTfer);
    }
    emMbxTferDelete(dwInstanceId, pMbxTfer);
}

SafeOsFree(oMbxTferDesc.pbyMbxTferDescData);

```

**See also:**

- *emNotify - eMbxTferType\_COE\_GETOBDESC*
- *emMbxTferCreate ()*
- *emGetSlaveId ()*
- See *CoeReadObjectDictionary ()* in *EcSdoServices.cpp* as an example for *emCoeGetObjectDescReq ()* .

### **6.11.10 emNotify - eMbxTferType\_COE\_GETOBDESC**

Completion of a SDO information service transfer to get a object description.

#### **emNotify - eMbxTferType\_COE\_GETOBDESC**

##### **Parameter**

- pbyInBuf: [in] Pointer to a structure of type *EC\_T\_MBXTFER*, contains the corresponding mailbox transfer object.
- dwInBufSize: [in] Size of the transfer object in bytes.
- pbyOutBuf: [out] Should be set to *EC\_NULL*
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to *EC\_NULL*

The corresponding transfer ID can be found in *EC\_T\_MBXTFER::dwTferId*. The transfer result is stored in *EC\_T\_MBXTFER::dwErrorCode*.

The object description stored in element *EC\_T\_MBX\_DATA::CoE\_ObDesc* of type *EC\_T\_COE\_OBDESC* is part of *EC\_T\_MBXTFER::MbxData* and may have to be buffered by the client. Access to the memory area *EC\_T\_MBXTFER::MbxData* outside of the notification caller context is illegal and the results are undefined.

struct **EC\_T\_COE\_OBDESC**

##### **Public Members**

###### **EC\_T\_WORD wObIndex**

Index in the object dictionary

###### **EC\_T\_WORD wDataType**

Data type of the object

###### **EC\_T\_BYTExObjCode**

Object code, see Table 62, ETG.1000 section 6

**EC\_T\_BYTExbyObjCategory**  
Object category

**EC\_T\_BYTExbyMaxNumSubIndex**  
Maximum sub index number

**EC\_T\_WORD wObNameLen**  
Length of the object name

**EC\_T\_WORD wStationAddress**  
Station address of the slave

**EC\_T\_CHAR \*pchObName**  
Object name (not NULL terminated!)

#### See also:

A more detailed description of the values for data type, object code etc. can be found in the EtherCAT specification ETG.1000, section 5.

### 6.11.11 emCoeGetEntryDescReq

```
static EC_T_DWORD ecatCoeGetEntryDescReq (
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTExbyObSubIndex,
    EC_T_BYTExbyValueInfo,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emCoeGetEntryDescReq (
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTExbyObSubIndex,
    EC_T_BYTExbyValueInfo,
    EC_T_DWORD dwTimeout
)
```

Determines the description of a specific object entry.

A unique transfer ID must be written into *EC\_T\_MBXTFER.dwTferId*. EC\_NOTIFY\_MBOXRCV is given on completion. This function may not be called from within the JobTask's context.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object Index
- **byObSubIndex** – [in] Object SubIndex
- **byValueInfo** – [in] The value info bit mask includes which elements shall be in the response. See *Value info flags* for available values.

- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. If the timeout value is set to EC\_NOWAIT the function will return immediately

### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- *EC\_E\_NOMEMORY* if the mailbox protocol queue of the slave if full
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support
- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_ADS\_IS\_RUNNING* if ADS server is running
- *CoE SDO error code*

### emCoeGetEntryDescReq Example

The following code demonstrates how to get the description of a specific object entry using the non-blocking API *emCoeGetEntryDescReq()*.

In this example the mailbox transfer object state is polled. *emNotify - EC\_NOTIFY\_MBOXRCV* can be used as alternative to polling.

```

EC_T_MBXTFER*      pMbxTfer = EC_NULL;
EC_T_MBXTFER_DESC oMbxTferDesc; /* mailbox transfer descriptor */

OsMemset (&oMbxTferDesc, 0, sizeof(EC_T_MBXTFER_DESC));
oMbxTferDesc.dwMaxDataLen = CROD_ENTRYDESC_SIZE;
oMbxTferDesc.pbyMbxTferDescData = (EC_T_BYTE*)OsMalloc(oMbxTferDesc.
→dwMaxDataLen);

/* allocate payload memory */
if (EC_NULL == oMbxTferDesc.pbyMbxTferDescData)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}

/* create transfer object */
pMbxTfer = emMbxTferCreate(dwInstanceId, &oMbxTferDesc);
if (EC_NULL == pMbxTfer)
{
    dwRetVal = EC_E_NOMEMORY;

    goto Exit;
}
dwRes = emCoeGetEntryDescReq(dwInstanceId, pMbxTfer, dwSlaveId, 0x6411, 1, 0,
→5000);
if (dwRes != EC_E_NOERROR)
{
```

(continues on next page)

(continued from previous page)

```

dwRetVal = dwRes;
goto Exit;
}

/* wait for transfer finished, see also emMbxTferAbort(...) */
while (eMbxTferStatus_Pend == pMbxTfer->eTferStatus)
{
    OsSleep(10);
}

dwRetVal = pMbxTfer->dwErrorCode;
Exit:
if (EC_NULL != pMbxTfer)
{
    if (eMbxTferStatus_Pend == pMbxTfer->eTferStatus)
    {
        emMbxTferAbort(dwInstanceId, pMbxTfer);
    }
    emMbxTferDelete(dwInstanceId, pMbxTfer);
}

SafeOsFree(oMbxTferDesc.pbyMbxTferDescData);

```

**Value info flags****EC\_COE\_ENTRY\_ObjAccess**

Object access

**EC\_COE\_ENTRY\_ObjCategory**

Object category

**EC\_COE\_ENTRY\_PdoMapping**

PDO mapping

**EC\_COE\_ENTRY\_UnitType**

Unit type

**EC\_COE\_ENTRY\_DefaultValue**

Default value

**EC\_COE\_ENTRY\_MinValue**

Minimum value

**EC\_COE\_ENTRY\_MaxValue**

Maximum value

**See also:**

- *emNotify - eMbxTferType\_COE\_GETENTRYDESC*
- *emMbxTferCreate()*
- *emGetSlaveId()*
- See *CoeReadObjectDictionary()* in *EcSdoServices.cpp* as an example for *emCoeGetEntryDesrcReq()* .

## 6.11.12 emNotify - eMbxTferType\_COE\_GETENTRYDESC

Completion of a SDO information service transfer to get a object entry description.

### emNotify - eMbxTferType\_COE\_GETENTRYDESC

#### Parameter

- pbyInBuf: [in] Pointer to a structure of type EC\_T\_MBXTFER, contains the corresponding mailbox transfer object.
- dwInBufSize: [in] Size of the transfer object in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

The corresponding transfer ID can be found in `EC_T_MBXTFER::dwTransferId`. The transfer result is stored in `EC_T_MBXTFER::dwErrorCode`.

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

```
struct EC_T_COE_ENTRYDESC
```

#### Public Members

##### **EC\_T\_WORD wObIndex**

Index in the object dictionary

##### **EC\_T\_BYTEx byObSubIndex**

Sub index in the object dictionary

##### **EC\_T\_BYTEx byValueInfo**

Bit mask which information is included in pbyData. See *Value info flags*

##### **EC\_T\_WORD wDataType**

Object data type according to ETG.1000

##### **EC\_T\_WORD wBitLen**

Object size (number of bits)

##### **EC\_T\_BYTEx byObAccess**

Access rights. See *Object access flags*

##### **EC\_T\_BOOL bRxPdoMapping**

Object is mappable in a RxPDO

##### **EC\_T\_BOOL bTxPdoMapping**

Object is mappable in a TxPDO

##### **EC\_T\_BOOL bObCanBeUsedForBackup**

Object can be used for backup

**EC\_T\_BOOL bObjCanBeUsedForSettings**  
Object can be used for settings

**EC\_T\_WORD wStationAddress**  
Station address of the slave

**EC\_T\_WORD wDataLen**  
Size of the remaining object data

**EC\_T\_BYTE \*pbyData**  
Remaining object data: dwUnitType, pbyDefaultValue, pbyMinValue, pbyMaxValue, pbyDescription  
(see ETG.1000.5 and ETG.1000.6)

#### Object access flags

**EC\_COE\_ENTRY\_Access\_R\_PREOP**  
Read access in Pre-Operational state

**EC\_COE\_ENTRY\_Access\_R\_SAFEOP**  
Read access in Safe-Operational state

**EC\_COE\_ENTRY\_Access\_R\_OP**  
Read access in Operational state

**EC\_COE\_ENTRY\_Access\_W\_PREOP**  
Write access in Pre-Operational state

**EC\_COE\_ENTRY\_Access\_W\_SAFEOP**  
Write access in Safe-Operational state

**EC\_COE\_ENTRY\_Access\_W\_OP**  
Write access in Operational state

#### See also:

- An example for the evaluation of *EC\_T\_COE\_ENTRYDESC::pbyData* comes with EcMasterDemo.
- A more detailed description of the values can be found in the EtherCAT specification ETG.1000, section 5 and 6.

### 6.11.13 emCoeProfileGetChannelInfo

```
static EC_T_DWORD ecatCoeProfileGetChannelInfo (
    EC_T_BOOL bStationAddress,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD dwChannel,
    EC_T_PROFILE_CHANNEL_INFO *pInfo
)
EC_T_DWORD emCoeProfileGetChannelInfo (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD dwChannel,
    EC_T_PROFILE_CHANNEL_INFO *pInfo
)
    Return information about a configured CoE profile channel from the ENI file.
```

## Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC\_TRUE: use station address, EC\_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **dwChannel1** – [in] Channel
- **pInfo** – [out] Channel info

## Returns

*EC\_E\_NOERROR* or error code

struct **EC\_T\_PROFILE\_CHANNEL\_INFO**

## Public Members

**EC\_T\_WORD wProfileNo**

[out] ProfileNo: “low word of CoE object 0x1000”

**EC\_T\_WORD wAddInfo**

[out] AddInfo : “high word of CoE object 0x1000”

**EC\_T\_CHAR szDisplayName[ECAT\_DEVICE\_NAMESIZE]**

[out] Display name

## emCoeProfileGetChannelInfo Example

The following code demonstrates how to return information about a configured CoE profile channel from the ENI file using the non-blocking API *emCoeProfileGetChannelInfo()*.

In this example the mailbox transfer object state is polled. *emNotify - EC\_NOTIFY\_MBOXRCV* can be used as alternative to polling.

```
/* get information about CoE profile channel configured in ENI */
EC_T_PROFILE_CHANNEL_INFO oInfo;
dwRes = emCoeProfileGetChannelInfo(dwInstanceId, EC_TRUE, wSlaveAddress, 0, &
→oInfo);
```

## 6.11.14 emNotify - EC\_NOTIFY\_COE\_INIT\_CMD

Indicates a COE mailbox transfer completion during slave state transition. This notification is disabled by default.

### emNotify - EC\_NOTIFY\_COE\_INIT\_CMD

#### Parameter

- **pbyInBuf**: [in] Pointer to a structure of type **EC\_T\_MBXTFER**, contains the corresponding mailbox transfer object.
- **dwInBufSize**: [in] Size of the transfer object provided at pbyInBuf in bytes.
- **pbyOutBuf**: [out] Should be set to **EC\_NULL**
- **dwOutBufSize**: [in] Should be set to 0

- pdwNumOutData: [out] Should be set to EC\_NULL

The `EC_T_MBX_DATA::CoE_InitCmd` element of type `EC_T_MBX_DATA_COE_INITCMD` is part of `EC_T_MBXTFER::MbxData` and may have to be buffered by the client. Access to the memory area `EC_T_MBXTFER::MbxData` outside of the notification caller context is illegal and the results are undefined.

```
struct EC_T_MBX_DATA_COE_INITCMD
```

### Public Members

`EC_T_SLAVE_PROP SlaveProp`

Slave properties

`EC_T_DWORD dwHandle`

Handle passed by EC\_IOCTL\_ADD\_COE\_INITCMD, otherwise zero

`EC_T_WORD wTransition`

Transition, e.g. ECAT\_INITCMD\_I\_P

`EC_T_CHAR szComment[MAX_STD_STRLEN]`

Comment (ENI)

`EC_T_DWORD dwErrorCode`

InitCmd result

`EC_T_BOOL bFixed`

Fixed flag (ENI)

`EC_T_BYTEx byCcs`

Client command specifier (read or write access)

`EC_T_BOOL bCompleteAccess`

Complete access

`EC_T_WORD wIndex`

Object Index

`EC_T_BYTEx bySubIndex`

Object SubIndex

`EC_T_DWORD dwDataLen`

InitCmd data length

`EC_T_BYTEx *pbyData`

InitCmd data

### See also:

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED*

### 6.11.15 CoE Emergency (emNotify - eMbxTferType\_COE\_EMERGENCY)

Indication of a CoE emergency request. A *emNotify - EC\_NOTIFY\_MBOXRCV* is given with *EC\_T\_MBXTFER::eMbxTferType = EC\_T\_MBXTFER\_TYPE::eMbxTferType\_COE\_EMERGENCY*.

#### **emNotify - eMbxTferType\_COE\_EMERGENCY**

##### **Parameter**

- pbyInBuf: [in] Pointer to a structure of type *EC\_T\_MBXTFER*, contains the corresponding mailbox transfer object.
- dwInBufSize: [in] Size of the transfer object in bytes.
- pbyOutBuf: [out] Should be set to *EC\_NULL*
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to *EC\_NULL*

In case of an emergency notification all registered clients will get this notification. The corresponding mailbox transfer object will be created inside the EtherCAT master. The content in *EC\_T\_MBXTFER::dwTferId* is undefined as it is not needed by the client and the master. The transfer result is stored in *EC\_T\_MBXTFER::dwErrorCode*.

The emergency data stored in element *EC\_T\_MBX\_DATA::CoE\_Emergency* of type *EC\_T\_COE\_EMERGENCY* is part of *EC\_T\_MBXTFER::MbxData* and may have to be buffered by the client. Access to the memory area *EC\_T\_MBXTFER::MbxData* outside of the notification caller context is illegal and the results are undefined.

```
struct EC_T_COE_EMERGENCY
```

##### **Public Members**

###### **EC\_T\_WORD wErrorCode**

Error code according to EtherCAT specification

###### **EC\_T\_BYTE byErrorRegister**

Error register

###### **EC\_T\_BYT abyData[EC\_COE\_EMERGENCY\_DATASIZE]**

Error data

###### **EC\_T\_WORD wStationAddress**

Slave node address of the faulty slave

#### **CoE Emergency Example**

```
/* send CoE Emergency */
EC_T_BYT abyEmergencyData[6] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 };
dwRes = esSendSlaveCoeEmergency(dwSimulatorId, wSlaveAddress, 0x1234 /* code */,
↔, abyEmergencyData, 6 /* data length */);
if (dwRes != EC_E_NOERROR)
{
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↔"esSendSlaveCoeEmergency failed: %s (0x%lx)\n", esGetText(dwSimulatorId, dwRes),
↔ dwRes));
    goto Exit;
}
```

**See also:**

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

### **6.11.16 CoE Abort (emNotify - EC\_NOTIFY\_MBSLAVE\_COE\_SDO\_ABORT)**

The application can abort asynchronous CoE Uploads and Downloads. The slave may abort CoE Uploads and Downloads which is indicated at the return code of `emCoeSdoUpload()`, ... . This notification is given if an SDO transfer aborts while sending init commands.

#### **emNotify - EC\_NOTIFY\_MBSLAVE\_COE\_SDO\_ABORT**

##### **Parameter**

- `pbyInBuf`: [in] Pointer to a structure of type `EC_T_MBXTFER`, contains the corresponding mailbox transfer object.
- `dwInBufSize`: [in] Size of the transfer object in bytes.
- `pbyOutBuf`: [out] Should be set to `EC_NULL`
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to `EC_NULL`

Detailed error information is stored in structure `EC_T_MBOX_SDO_ABORT_DESC` of the union element `SdoAbortDesc`.

```
struct EC_T_MBOX_SDO_ABORT_DESC
```

##### **Public Members**

`EC_T_SLAVE_PROP SlaveProp`  
Slave properties

`EC_T_DWORD dwErrorCode`  
Error code

`EC_T_WORD wObjIndex`  
SDO object index

`EC_T_BYTE bySubIndex`  
SDO object sub index

**See also:**

`emMbxTferAbort()`

### 6.11.17 emConvertEcErrorToCoeError

`EC_T_DWORD ecatConvertEcErrorToCoeError (EC_T_DWORD dwErrorCode)`

```
EC_T_DWORD emConvertEcErrorToCoeError (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwErrorCode
)
    Convert master error code to CoE error code.
```

#### Returns

CoE error code according to the following specifications:

- CoE Abort Codes defined in ETG.1000.6 V1.0.4, Table 41: SDO Abort Codes
- Additional codes defined in ETG.1020, V1.2.0, Table 21: CoE Abort Codes (extension)

#### emConvertEcErrorToCoeError Example

The following code demonstrates how to convert master error code to CoE error code using the non-blocking API `emConvertEcErrorToCoeError()`.

In this example the mailbox transfer object state is polled. `emNotify - EC_NOTIFY_MBOXRCV` can be used as alternative to polling.

```
dwCoeError = emConvertEcErrorToCoeError(dwInstanceId, EC_E_NOERROR);
```

## 6.12 File access over EtherCAT (FoE)

The File access over EtherCAT (FoE) mailbox command specifies a standard way to download a firmware or any other files from a client to a server or to upload a firmware or any other files from a server to a client.

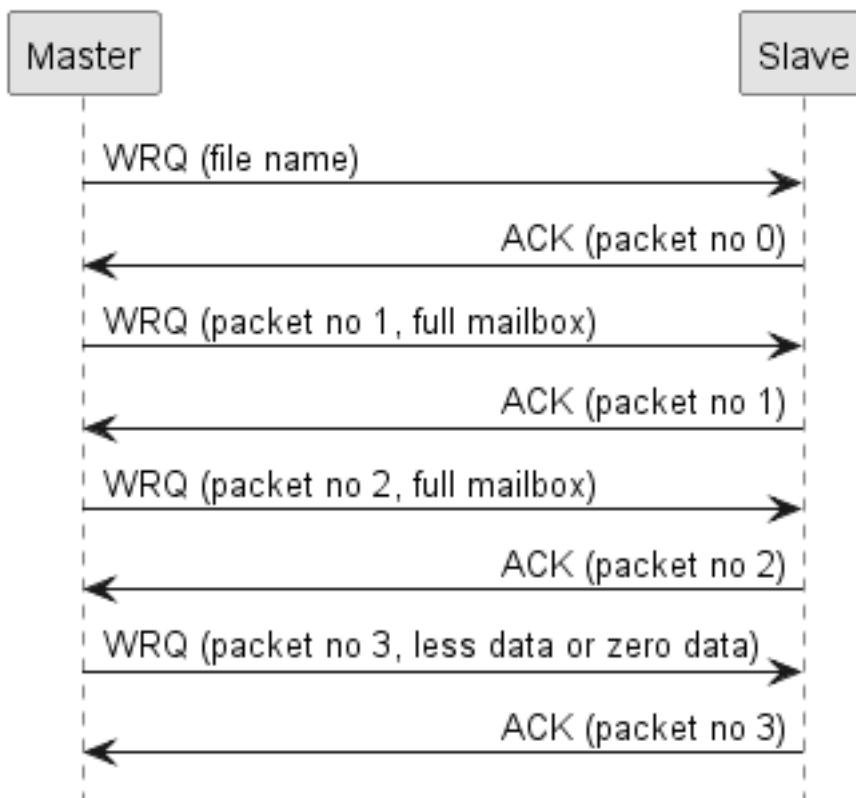
Reference:

- ETG.1000.5, ETG.1000.6 and ETG.5003.2

### 6.12.1 Specification

#### FoE file download

##### Regular download



### Segmented download

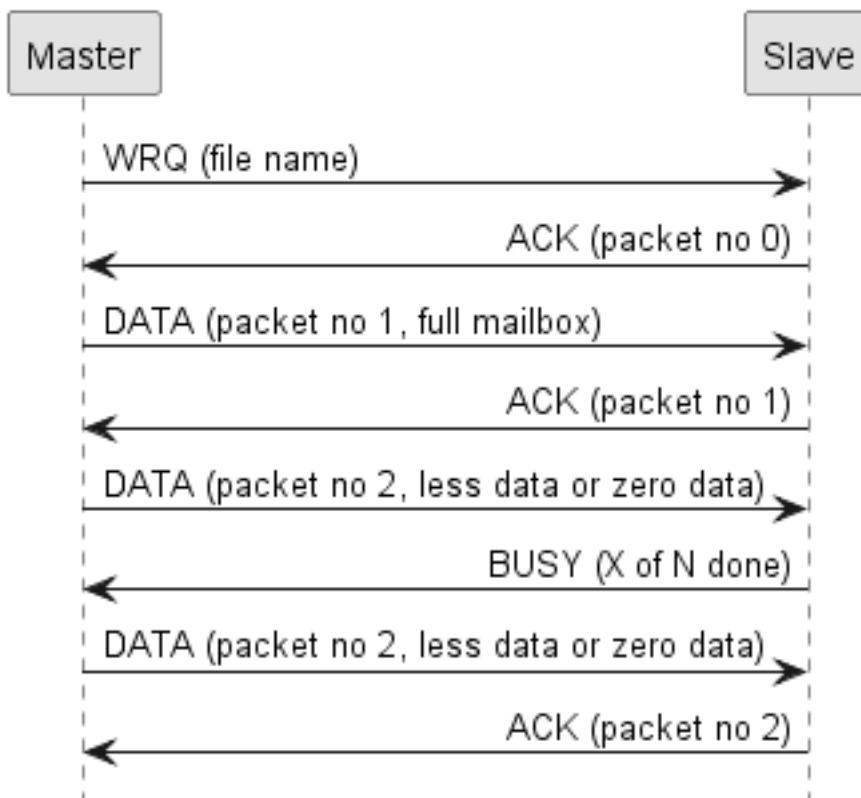
In case of segmented download the EC-Master raises emNotify - EC\_NOTIFY\_MBOXRCV in [EC-Master Class B documentation](#) to request more data from the application after each ACK from the slave. The notification handler may not block the EC-Master, e.g. due to reading from or writing to the file system, therefore applications typically do not handle EC\_NOTIFY\_MBOXRCV within the JobTask context. The segments are transferred using the slave's mailbox, so the maximum size of a segment is known from the configuration. The segment's size can be calculated as follows:

segment size = mailbox size - 12 (protocol overhead)

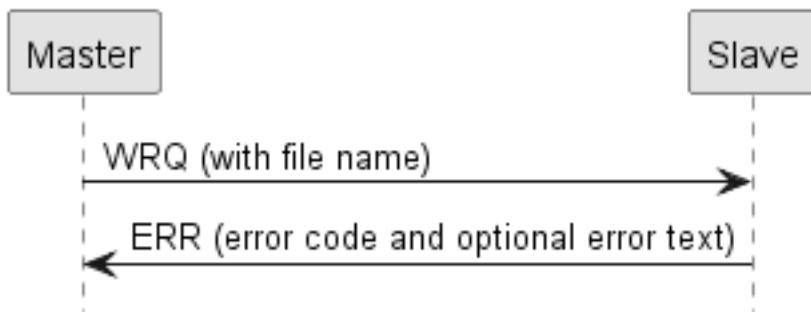
#### See also:

- [`EC\_T\_CFG\_SLAVE\_INFO::dwMbxOutSize\(emGetCfgSlaveInfo\(\)\)`](#)
- [`EC\_T\_CFG\_SLAVE\_INFO::dwMbxOutSize2\(emGetCfgSlaveInfo\(\)\)`](#)
- [`EC\_T\_CFG\_SLAVE\_INFO::dwMbxInSize\(emGetCfgSlaveInfo\(\)\)`](#)
- [`EC\_T\_CFG\_SLAVE\_INFO::dwMbxInSize2\(emGetCfgSlaveInfo\(\)\)`](#)

#### Download with busy



#### Download with error

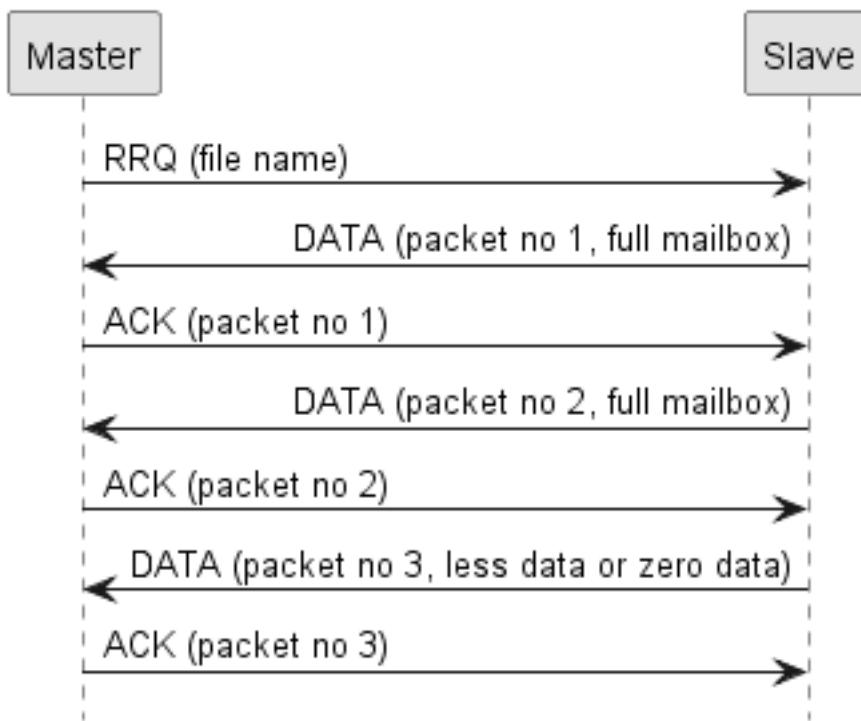


#### FoE file upload

The names of available files and their size are slave specific and cannot be retrieved using FoE. It is possible to upload the complete file in segments without the need to know the file size.

The segments are transferred using the slave's mailbox, so the maximum size of a segment is known from the configuration.

#### Regular upload



### Boot State

For the download of firmware the BOOT state in the EtherCAT state machine is defined. In bootstrap mode only FoE Download is possible. A special Mailbox size can be supported by the slave for the Boot state (ETG.2000). This is part of the Init-Commands in the network configuration.

#### See also:

- [EC\\_T\\_CFG\\_SLAVE\\_INFO::dwMbxOutSize2\(emGetCfgSlaveInfo\(\)\)](#)
- [EC\\_T\\_CFG\\_SLAVE\\_INFO::dwMbxInSize2\(emGetCfgSlaveInfo\(\)\)](#)

### 6.12.2 emFoeFileDownload

```

static EC_T_DWORD ecatFoeFileDownload (
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *achFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emFoeFileDownload (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *achFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
  
```

Execute a FoE File download to an EtherCAT slave device.

This function is used to download a complete file. The function returns after the download has been successfully completed or an error has occurred. This function may not be called from within the JobTask's context.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **achFileName** – [in] File name of slave file to write
- **dwFileNameLen** – [in] Length of slave file name in bytes
- **pbyData** – [in] Buffer containing transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **dwPassword** – [in] slave password
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or maximum file name length of 64 bytes (MAX\_FILE\_NAME\_SIZE) exceeded
- *EC\_E\_FOE\_ERRCODE\_NOTINBOOTSTRAP* if slave in BOOTSTRAP and filename not accepted by slave
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *FoE error code*

### See also:

`emGetSlaveId()`

The following example demonstrates how to do a firmware update using FoE (ETG.5003.2).

### emFoeFileDownload() Example

This example code shows how to download a file from a buffer at the master to a slave, e.g. in order to update the firmware.

```
EC_T_DWORD dwRetVal = EC_E_ERROR;
EC_T_DWORD dwRes = EC_E_ERROR;

EC_T_WORD wSlaveAddress = MY_FOE_SLAVE_ADDRESS /* e.g. 1001 */;
EC_T_DWORD dwSlaveId = ecatGetSlaveId(wSlaveAddress);

FILE* pfLocalFile = EC_NULL;

EC_T_BYTEx pbyBuffer = EC_NULL;
EC_T_DWORD dwBufferSize = 0;

/* read file to buffer */
pfLocalFile = OsFopen(MY_FOE_TRANSFER_LOCAL_FILENAME, "rb");
if (EC_NULL == pfLocalFile)
{
```

(continues on next page)

(continued from previous page)

```

dwRetVal = EC_E_OPENFAILED;
goto Exit;
}
dwBufferSize = OsGetFileSize(pfLocalFile);

pbyBuffer = (EC_T_BYTE*)OsMalloc(dwBufferSize);
if (EC_NULL == pbyBuffer)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}
dwRes = (EC_T_DWORD)OsFread(pbyBuffer, 1, dwBufferSize, pfLocalFile);
if (dwRes != dwBufferSize)
{
    dwRetVal = EC_E_ERROR;
    goto Exit;
}

/* download buffer to slave */
dwRes = ecatFoeFileDownload(dwSlaveId, MY_FOE_TRANSFER_SLAVE_FILENAME,
    (EC_T_DWORD)OsStrlen(MY_FOE_TRANSFER_SLAVE_FILENAME),
    pbyBuffer, dwBufferSize, 0, 600000 /* 10 minutes */);
if (EC_E_NOERROR != dwRes)
{
    dwRetVal = dwRes;
    goto Exit;
}

dwRetVal = EC_E_NOERROR;
Exit:
/* free resources */
if (EC_NULL != pfLocalFile)
{
    OsFclose(pfLocalFile);
}
SafeOsFree(pbyBuffer);

```

### 6.12.3 emFoeFileUpload

```

static EC_T_DWORD ecatFoeFileUpload(
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *achFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_BYT* pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD *pdwOutDataLen,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)

```

```
EC_T_DWORD emFoeFileUpload(
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *achFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD *pdwOutDataLen,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
```

) Execute a FoE File upload from an EtherCAT slave device.

This function is used to upload a complete file. The function returns after the upload has been successfully completed or an error has occurred. This function may not be called from within the JobTask's context.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **achFileName** – [in] File name of slave file to read.
- **dwFileNameLen** – [in] Length of slave file name in bytes.
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **pdwOutDataLen** – [out] Length of received data in bytes
- **dwPassword** – [in] slave password
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

#### Returns

- [\*EC\\_E\\_NOERROR\*](#) if successful
- [\*EC\\_E\\_INVALIDSTATE\*](#) if master isn't initialized
- [\*EC\\_E\\_INVALIDPARAM\*](#) if dwInstanceID is out of range or maximum file name length of 64 bytes (`MAX_FILE_NAME_SIZE`) exceeded
- [\*EC\\_E\\_TIMEOUT\*](#) if dwTimeout elapsed during the API call
- [\*EC\\_E\\_MASTER\\_RED\\_STATE\\_INACTIVE\*](#) if Master Redundancy is configured and master is inactive
- [\*FoE error code\*](#)

#### See also:

[\*emGetSlaveId\(\)\*](#)

### 6.12.4 emFoeDownloadReq

```
static EC_T_DWORD ecatFoeDownloadReq (
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *achFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emFoeDownloadReq (
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *achFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
```

Initiates an FoE File download to an EtherCAT slave device.

This function is used to download a complete file and returns immediately. After the download has been successfully completed or an error has occurred, EC\_NOTIFY\_MBOXRCV is raised. The progress of the file transfer is also notified with EC\_NOTIFY\_MBOXRCV.

*Deprecated:*

EC\_NOWAIT as a timeout is still accepted for reasons of compatibility and sets the timeout to 10 seconds

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **achFileName** – [in] File name of slave file to write
- **dwFileNameLen** – [in] Length of slave file name in bytes
- **dwPassword** – [in] Slave password
- **dwTimeout** – [in] Timeout [ms]

#### Returns

- **EC\_E\_NOERROR** if successful
- **EC\_E\_INVALIDSTATE** if master isn't initialized
- **EC\_E\_INVALIDPARAM** if dwInstanceID is out of range or maximum file name length of 64 bytes (MAX\_FILE\_NAME\_SIZE) exceeded
- **EC\_E\_MASTER\_RED\_STATE\_INACTIVE** if Master Redundancy is configured and master is inactive
- **FoE error codes**

#### See also:

[emGetSlaveId\(\)](#)

## 6.12.5 emFoeSegmentedDownloadReq

```
static EC_T_DWORD ecatFoeSegmentedDownloadReq (
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *szFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_DWORD dwFileSize,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emFoeSegmentedDownloadReq (
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *szFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_DWORD dwFileSize,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
```

Initiates or continues a segmented FoE File download to an EtherCAT slave device.

This function is used to download a file chunk-by-chunk and returns immediately. An EC\_NOTIFY\_MBOXRCV is raised to request the next chunk from the application or to provide information about the progress and the change in the transfer status.

The slave may have a different mailbox size for BOOTSTRAP than for PREOP, SAFEOP, OP. See [EC\\_T\\_CFG\\_SLAVE\\_INFO.dwMbxInSize2](#). The maximum chunk size is the slave's mailbox size - 12 bytes overhead for EtherCAT's FoE protocol. The mailbox transfer object's buffer must be at least as big as the chunks to be transferred.

*Deprecated:*

EC\_NOWAIT as a timeout is still accepted for reasons of compatibility and sets the timeout to 10 seconds

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Pointer to the corresponding mailbox transfer object.  
*EC\_T\_MBXTFER.pbyMbxTferData*: next chunk, *EC\_T\_MBXTFER.dwDataLen*: next chunk size.
- **dwSlaveId** – [in] Slave ID
- **szFileName** – [in] File name of slave file to write. Only evaluated when initiating the request.
- **dwFileNameLen** – [in] Length of slave file name in bytes
- **dwFileSize** – [in] Complete file size (mandatory). Used also for progress information. Only evaluated when initiating the request.
- **dwPassword** – [in] Slave password. Only evaluated when initiating the request.
- **dwTimeout** – [in] Timeout [ms]. Only evaluated when initiating the request.

### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or maximum file name length of 64 bytes (MAX\_FILE\_NAME\_SIZE) exceeded
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *FoE error code*

**See also:**

- *EC\_T\_Cfg\_Slave\_Info::dwMbxOutSize(emGetCfgSlaveInfo())*
- *EC\_T\_Cfg\_Slave\_Info::dwMbxOutSize2(emGetCfgSlaveInfo())*
- *emGetSlaveId()*

## 6.12.6 emFoeUploadReq

```
static EC_T_DWORD ecatFoeUploadReq(
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *achFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emFoeUploadReq(
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *achFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
```

) Initiates an FoE File upload from an EtherCAT slave device.

This function is used to upload a complete file and returns immediately. After the upload has been successfully completed or an error has occurred, EC\_NOTIFY\_MBOXRCV is raised. The progress of the file transfer is also notified with EC\_NOTIFY\_MBOXRCV.

*Deprecated:*

EC\_NOWAIT as a timeout is still accepted for reasons of compatibility and sets the timeout to 10 seconds

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **achFileName** – [in] File name of slave file to read.
- **dwFileNameLen** – [in] Length of slave file name in bytes.
- **dwPassword** – [in] slave password

- **dwTimeout** – [in] Timeout [ms]

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or maximum file name length of 64 bytes (MAX\_FILE\_NAME\_SIZE) exceeded
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *FoE error code*

#### See also:

*emGetSlaveId()*

### 6.12.7 emFoeSegmentedUploadReq

```
static EC_T_DWORD ecatFoeSegmentedUploadReq (
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *szFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_DWORD dwFileSize,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
```

```
EC_T_DWORD emFoeSegmentedUploadReq (
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    const EC_T_CHAR *szFileName,
    EC_T_DWORD dwFileNameLen,
    EC_T_DWORD dwFileSize,
    EC_T_DWORD dwPassword,
    EC_T_DWORD dwTimeout
)
```

Initiates or continues a segmented FoE File upload from an EtherCAT slave device.

This function is used to upload a file chunk-by-chunk and returns immediately. An EC\_NOTIFY\_MBOXRCV is raised to provide the next chunk to the application or to get information about the progress and the change in the transfer status.

The slave may have a different mailbox size for BOOTSTRAP than for PREOP, SAFEOP, OP. See *EC\_T\_CFG\_SLAVE\_INFO.dwMbxInSize2*. The maximum chunk size is the slave's mailbox size - 12 bytes overhead for EtherCAT's FoE protocol. The mailbox transfer object's buffer must be at least as big as the chunks to be transferred.

*Deprecated:*

*EC\_NOWAIT* as a timeout is still accepted for reasons of compatibility and sets the timeout to 10 seconds

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Pointer to the corresponding mailbox transfer object. *EC\_T\_MBXTFER.pbyMbxTferData*: next chunk, *EC\_T\_MBXTFER.dwDataLen*: next chunk size.
- **dwSlaveId** – [in] Slave ID
- **szFileName** – [in] File name of slave file to write. Only evaluated when initiating the request.
- **dwFileNameLen** – [in] Length of slave file name in bytes.
- **dwFileSize** – [in] Used only for progress information.
- **dwPassword** – [in] Slave password. Only evaluated when initiating the request.
- **dwTimeout** – [in] Timeout [ms]. Only evaluated when initiating the request.

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or maximum file name length of 64 bytes (MAX\_FILE\_NAME\_SIZE) exceeded
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *FoE error code*

#### See also:

- *EC\_T\_CFG\_SLAVE\_INFO::dwMbxInSize(emGetCfgSlaveInfo())*
- *emGetSlaveId()*

The following example demonstrates how to upload a file in segments:

#### **emFoeSegmentedUploadReq() Example**

The following code demonstrates how to receive FoE from the slave with address MY\_FOE\_SLAVE\_ADDRESS and store it in a file. The data uploaded from MY\_FOE\_TRANSFER\_SLAVE\_FILENAME of the slave is stored in a file with filename MY\_FOE\_TRANSFER\_LOCAL\_FILENAME in this example.

The example code can be placed after the corresponding *emSetMasterState()*-call in EcDemoApp.cpp:

```

EC_T_DWORD dwRetVal = EC_E_ERROR;
EC_T_DWORD dwRes = EC_E_ERROR;

EC_T_WORD wSlaveAddress = MY_FOE_SLAVE_ADDRESS /* e.g. 1001 */;
EC_T_DWORD dwSlaveId = ecatGetSlaveId(wSlaveAddress);

EC_T_CFG_SLAVE_INFO oCfgSlaveInfo;

FILE* pfLocalFile = EC_NULL;
EC_T_DWORD dwFileSize = 0;

EC_T_MBXTFER_DESC oMbxTferDesc;
EC_T_MBXTFER* pMbxTfer = EC_NULL;
EC_T_BYTE* pbyBuffer = EC_NULL;
EC_T_DWORD dwBufferSize = 0;

OsMemset(&oCfgSlaveInfo, 0, sizeof(EC_T_CFG_SLAVE_INFO));

```

(continues on next page)

(continued from previous page)

```

OsMemset (&oMbxTferDesc, 0, sizeof(EC_T_MBXTFER_DESC)) ;

/* get max. FoE segment size */
dwRes = ecatGetCfgSlaveInfo(EC_TRUE, wSlaveAddress, &oCfgSlaveInfo);
if (EC_E_NOERROR != dwRes)
{
    dwRetVal = dwRes;
    goto Exit;
}
/* mailbox contains mailbox header, FoE header and payload for buffer */
dwBufferSize = oCfgSlaveInfo.dwMbxInSize - ETHERCAT_MAX_FOE_MBOX_HDR_LEN;

/* allocate segment buffer */
pbyBuffer = (EC_T_BYTE*)OsMalloc(dwBufferSize);
if (EC_NULL == pbyBuffer)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}
oMbxTferDesc.pbyMbxTferDescData = pbyBuffer;
oMbxTferDesc.dwMaxDataLen = dwBufferSize;
pMbxTfer = ecatMbxTferCreate(&oMbxTferDesc);
if (EC_NULL == pMbxTfer)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}
pMbxTfer->dwTferId = 0x12345678; /* uniq ID from application */

/* open local file */
pfLocalFile = OsFopen(MY_FOE_TRANSFER_LOCAL_FILENAME /* e.g. (EC_T_CHAR*)
→"MyFile.dat" */, "wb");
if (EC_NULL == pfLocalFile)
{
    dwRetVal = EC_E_OPENFAILED;
    goto Exit;
}

/* start upload */
dwRes = ecatFoeSegmentedUploadReq(pMbxTfer, dwSlaveId,
    MY_FOE_TRANSFER_SLAVE_FILENAME /* e.g. (EC_T_CHAR*)"MyFile.dat" */, (EC_T_
→DWORD)OsStrlen(MY_FOE_TRANSFER_SLAVE_FILENAME),
    0xFFFFFFFF /* unknown file size */, 0, MBX_TIMEOUT /* e.g. 5000 */);
if (EC_E_NOERROR != dwRes)
{
    dwRetVal = dwRes;
    goto Exit;
}

/* upload file writing segments to disk */
while ((eMbxTferStatus_Pend == pMbxTfer->eTferStatus)
    || (eMbxTferStatus_TferWaitingForContinue == pMbxTfer->eTferStatus))
{
    /* wait for master received data from slave */
    if (eMbxTferStatus_TferWaitingForContinue == pMbxTfer->eTferStatus)
    {
        /* write segment */
        OsFwrite(pMbxTfer->pbyMbxTferData, pMbxTfer->dwDataLen, 1,_
→pfLocalFile);
        dwFileSize = dwFileSize + pMbxTfer->dwDataLen;
    }
}

```

(continues on next page)

(continued from previous page)

```

/* acknowledge segment so master can receive the next segment */
dwRes = ecatFoeSegmentedUploadReq(pMbxTfer, 0, EC_NULL, 0, 0, 0, 0);
if (EC_E_NOERROR != dwRes)
{
    dwRetVal = dwRes;
    goto Exit;
}
}
OsSleep(10);
}
if (eMbxTferStatus_TferReqError == pMbxTfer->eTferStatus)
{
    dwRetVal = pMbxTfer->dwErrorCode;
    goto Exit;
}

/* transfer done */
if (eMbxTferStatus_TferDone == pMbxTfer->eTferStatus)
{
    dwRetVal = pMbxTfer->dwErrorCode; /* e.g. EC_E_NOERROR */
    pMbxTfer->eTferStatus = eMbxTferStatus_Idle;
}

Exit:
/* free resources */
if (EC_NULL != pfLocalFile)
{
    OsFclose(pfLocalFile);
}
if (EC_NULL != pMbxTfer)
{
    if (eMbxTferStatus_Pend == pMbxTfer->eTferStatus)
    {
        CEcTimer oTimeout(MBX_TIMEOUT);
        ecatMbxTferAbort(pMbxTfer);
        while ((eMbxTferStatus_Pend == pMbxTfer->eTferStatus) && !oTimeout.
→IsElapsed())
        {
            OsSleep(100);
        }
    }
    ecatMbxTferDelete(pMbxTfer);
}
SafeOsFree(pbyBuffer);

```

### 6.12.8 emConvertEcErrorToFoeError

EC\_T\_DWORD **ecatConvertEcErrorToFoeError** (EC\_T\_DWORD dwErrorCode)

```

EC_T_DWORD emConvertEcErrorToFoeError (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwErrorCode
)

```

Convert master error code to FoE error code.

#### Returns

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

### 6.12.9 emNotify - 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. Detailed error information is stored in structure `EC_T_WKCERR_DESC` of the union element `WkcErrDesc`.

struct `EC_T_WKCERR_DESC`

```
EC_T_SLAVE_PROP SlaveProp
EC_T_BYTEx byCmd
EC_T_DWORD dwAddr
EC_T_WORD wWkcSet
EC_T_WORD wWkcAct
EC_T_DWORD dwTaskId
EC_T_WORD wMsId
```

### 6.12.10 emNotify - EC\_NOTIFY\_FOE\_MBSLAVE\_ERROR

This error will be indicated in case a FoE mailbox slave send an error message. Detailed error information is stored in structure `EC_T_MBOX_FOE_ABORT_DESC` of the union element `FoeErrorDesc`.

struct `EC_T_MBOX_FOE_ABORT_DESC`

#### Public Members

```
EC_T_SLAVE_PROP SlaveProp
    Slave properties

EC_T_DWORD dwErrorCode
    Error code

EC_T_CHAR achErrorMessage[MAX_STD_STRLEN]
    FoE error string
```

### 6.12.11 Extending EC\_T\_MBX\_DATA

FoE transfer data, e.g. progress information in notification.

struct `EC_T_MBX_DATA_FOE`

#### Public Members

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

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

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

**EC\_T\_DWORD dwBusyEntire**  
 [out] If dwBusyEntire > 0: Slave is busy

**EC\_T\_CHAR szBusyComment[EC\_FOE\_BUSY\_COMMENT\_SIZE]**  
 [out] Busy Comment from slave

**EC\_T\_DWORD dwFileSize**  
 [out] File size

**EC\_T\_WORD wStationAddress**  
 [out] Station address of the slave

## 6.13 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 `emSoeWrite()` the data and elements of an IDN which are writeable can be written.

- **Read IDN:**

With `emSoeRead()` the data and elements of an IDN can be read.

- **Procedure command Execution:**

With `emSoeWrite()` 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 `emSoeAbortProcCmd()` 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 `emRegisterClient()` 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 `emRegisterClient()` must be called.

Abbreviations:

- **IDN:**

identification number: 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 Servo drive profile over EtherCAT (SoE)

- **SSC:**

SoE Service Channel (non-cyclic data exchange)

### 6.13.1 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**

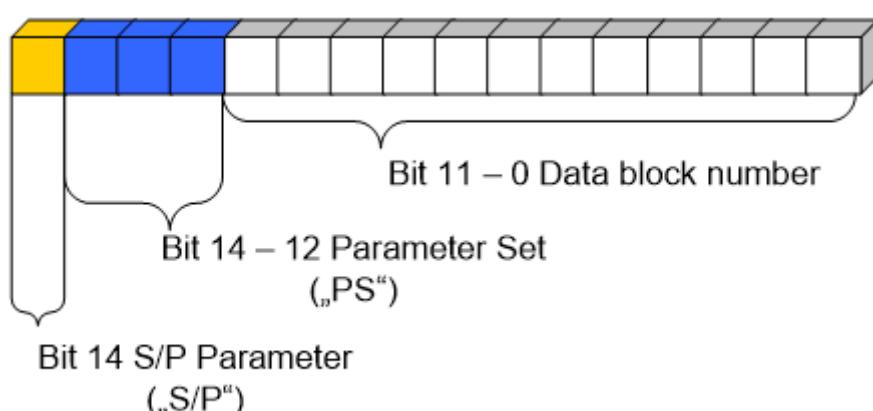
Operation data

#### **SOE\_BM\_ELEMENTFLAG\_DEFAULT**

The IDN default value

### 6.13.2 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 = 0x8001

### 6.13.3 emSoeWrite

```
static EC_T_DWORD ecatSoeWrite(
    EC_T_DWORD dwSlaveId,
    EC_T_BYT byDriveNo,
    EC_T_BYT *pbyElementFlags,
    EC_T_WORD wIDN,
    EC_T_BYT *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emSoeWrite (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_BYT byDriveNo,
    EC_T_BYT *pbyElementFlags,
    EC_T_WORD wIdn,
    EC_T_BYT *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout
)
```

Execute a SoE SSC Write service which download data to an EtherCAT slave device.

The function returns after timeout expired or download is completed successfully (Write response is received). It 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. 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.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **byDriveNo** – [in] Drive number to address inside slave device
- **pbyElementFlags** – [in/out] SoE ElementFlags. Flag indicating elements to address
- **wIdn** – [in] IDN of the object to address
- **pbyData** – [in] Buffer containing transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **dwTimeout** – [in] Timeout [ms]

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- *EC\_E\_NOMEMORY* if the mailbox protocol queue of the slave if full
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support

- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_AMS\_IS\_RUNNING* if AMS server is running
- *SoE error code*

**See also:**

`emGetSlaveId()`

### 6.13.4 emSoeWriteReq

Requests an SoE SSC Write and returns immediately.

```
static 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  
)  
  
EC_T_DWORD emSoeWriteReq(  
    EC_T_DWORD dwInstanceID,  
    EC_T_MBXTFER *pMbxTfer,  
    EC_T_DWORD dwSlaveId,  
    EC_T_BYTE byDriveNo,  
    EC_T_BYTE *pbyElementFlags,  
    EC_T_WORD wIdn,  
    EC_T_DWORD dwTimeout  
)
```

Requests an SoE SSC Write and returns immediately.

This function may be called from within the JobTask's context.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **byDriveNo** – [in] Drive number to address inside slave device
- **pbyElementFlags** – [in/out] SoE ElementFlags. Flag indicating elements to address
- **wIdn** – [in] IDN of the object to address
- **dwTimeout** – [in] Timeout [ms]

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- *EC\_E\_NOMEMORY* if the mailbox protocol queue of the slave is full

- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support
- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_AMS\_IS\_RUNNING* if ADS server is running
- *SoE error code*

**See also:**

- *emSoeWrite()*
- *emGetSlaveId()*

### 6.13.5 emSoeRead

```
static 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
)
EC_T_DWORD emSoeRead (
    EC_T_DWORD dwInstanceID,
    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
)
    Execute a SoE SCC Read service which upload data from an EtherCAT SoE slave device.
```

The received data can consist of several fragments. The reserved data buffer (pbyData) must have space for all received data segments and the selected Timeout should be increasing with the count of fragments.

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **byDriveNo** – [in] Drive number to address inside slave device
- **pbyElementFlags** – [in/out] SoE ElementFlags. Flag indicating elements to address
- **wIdn** – [in] IDN of the object to address
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length in bytes

- **pdwOutDataLen** – [out] Length of received data in bytes
- **dwTimeout** – [in] Timeout [ms]

#### Returns

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- *EC\_E\_NOMEMORY* if the mailbox protocol queue of the slave if full
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support
- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_AMS\_IS\_RUNNING* if AMS server is running
- *SoE error code*

#### See also:

`emGetSlaveId()`

### 6.13.6 emSoeReadReq

```
static 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
)
EC_T_DWORD emSoeReadReq(
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_BYTE byDriveNo,
    EC_T_BYTE *pbyElementFlags,
    EC_T_WORD wIDn,
    EC_T_DWORD dwTimeout
)
```

Requests an SoE SSC Read and returns immediately.

This function may be called from within the JobTask's context.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID

- **byDriveNo** – [in] Drive number to address inside slave device
- **pbyElementFlags** – [in/out] SoE ElementFlags. Flag indicating elements to address
- **wIdn** – [in] IDN of the object to address
- **dwTimeout** – [in] Timeout [ms]. Must not be set to EC\_NOWAIT

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC\_NULL or contains EC\_NULL pointer, or dwTimeout is EC\_NOWAIT
- *EC\_E\_NOMEMORY* if the mailbox protocol queue of the slave if full
- *EC\_E\_SLAVE\_NOT\_PRESENT* if slave not present
- *EC\_E\_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC\_E\_NO\_MBX\_SUPPORT* if slave has no mailbox support
- *EC\_E\_INVALID\_SLAVE\_STATE* if slave is in an invalid state for mailbox transfer
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC\_E\_ADS\_IS\_RUNNING* if ADS server is running
- *SoE error code*

**See also:**

- *emSoeRead()*
- *emGetSlaveId()*

**6.13.7 emSoeAbortProcCmd**

```
static EC_T_DWORD ecatSoeAbortProcCmd (
    EC_T_DWORD dwSlaveId,
    EC_T_BYT byDriveNo,
    EC_T_BYT *pbyElementFlags,
    EC_T_WORD wIDN,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emSoeAbortProcCmd (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_BYT byDriveNo,
    EC_T_BYT *pbyElementFlags,
    EC_T_WORD wIdn,
    EC_T_DWORD dwTimeout
)
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.

---

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

---

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **byDriveNo** – [in] Drive number to address inside slave device
- **pbyElementFlags** – [in/out] SoE ElementFlags. Flag indicating elements to address
- **wIdn** – [in] IDN of the object to address
- **dwTimeout** – [in] Timeout [ms]

**Returns***EC\_E\_NOERROR* or error code**See also:***emGetSlaveId()***6.13.8 emConvertEcErrorToSoeError**EC\_T\_DWORD **ecatConvertEcErrorToSoeError** (EC\_T\_DWORD dwErrorCode)

```
EC_T_DWORD emConvertEcErrorToSoeError (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwErrorCode
)
```

Convert master error code to SoE error code.

**Returns**

SoE error code

**6.13.9 emNotify - 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 *EC\_T\_ERROR\_NOTIFICATION\_DESC*.

**6.13.10 emNotify - 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 *EC\_T\_ERROR\_NOTIFICATION\_DESC*.

**6.14 Vendor specific protocol over EtherCAT (VoE)**

VoE is for vendor specific protocols. With VoE the vendor has access to a raw EtherCAT mailbox without a specific header or specific protocol mechanism.

### 6.14.1 emVoeWrite

```
static EC_T_DWORD ecatVoeWrite(
    EC_T_DWORD dwSlaveId,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emVoeWrite (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout
)
```

Execute a VoE mailbox write to an EtherCAT slave device.

This function blocks until the VoE write has been successfully completed or an error has occurred.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **pbyData** – [in] Buffer containing transferred data
- **dwDataLen** – [in] Buffer length in bytes The maximum data length including 6 bytes for the mailbox header is given at [EC\\_T\\_CFG\\_SLAVE\\_INFO.dwMbxOutSize](#)
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

#### Returns

[EC\\_E\\_NOERROR](#) or error code

#### emVoeWrite() Example

```
dwRes = emVoeWrite(dwInstanceId, dwSlaveId, abyVoeDataOutBuf,
    sizeof(abyVoeDataOutBuf), 5000);
if(dwRes != EC_E_NOERROR) {
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
        "emVoeWrite(...): %s (0x%lx)\n", ecatGetText(dwRes), dwRes));
}
```

#### See also:

- [emGetCfgSlaveInfo\(\)](#)
- [emGetSlaveId\(\)](#)

## 6.14.2 emVoeWriteReq

```
static EC_T_DWORD ecatVoeWriteReq (
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emVoeWriteReq (
    EC_T_DWORD dwInstanceID,
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_DWORD dwTimeout
)
```

Initiates a VoE mailbox write to an EtherCAT slave device.

The amount of data bytes to write has to be stored in *EC\_T\_MBXTFER.dwDataLen*. The maximum data length including 6 bytes for the mailbox header is given at *EC\_T\_CFG\_SLAVE\_INFO.dwMbxOutSize*. A unique transfer ID must be written into *EC\_T\_MBXTFER.dwTferId*.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **dwTimeout** – [in] Timeout [ms]

### Returns

*EC\_E\_NOERROR* or error code

### emVoeWriteReq() Example

```
dwRes = emVoeWriteReq(dwInstanceId, pMbxTfer, dwSlaveId, 5000);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
→"emVoeWriteReq: %s (0x%lx)\n", ecatGetText(dwRes), dwRes));
    goto Exit;
}
```

### See also:

*emGetSlaveId()*

## 6.14.3 emVoeRead

```
static EC_T_DWORD ecatVoeRead (
    EC_T_DWORD dwSlaveId,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD *pdwOutDataLen,
    EC_T_DWORD dwTimeout
)
```

```
EC_T_DWORD emVoeRead (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD *pdwOutDataLen,
    EC_T_DWORD dwTimeout
)
```

) Retrieves VoE mailbox, that was sent by an EtherCAT slave device.

If a VoE mailbox was already received, further received VoE mailboxes will be discarded as long as this function was not called. The received data includes the Mailbox header of type [ETHERCAT\\_MBOX\\_HEADER](#) followed by the VoE payload.

This function blocks until the VoE data has been successfully received or an error has occurred.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **pdwOutDataLen** – [out] Length of received data in bytes
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.  
EC\_NOWAIT returns the function immediately

### Returns

- [EC\\_E\\_NOERROR](#) if the VoE slave has provided some VoE data
- [EC\\_E\\_INVALIDSTATE](#) if master isn't initialized
- [EC\\_E\\_INVALIDPARM](#) if dwInstanceID is out of range
- [EC\\_E\\_TIMEOUT](#) if dwTimeout elapsed during the API call
- [EC\\_E\\_VOE\\_NO\\_MBX\\_RECEIVED](#) no VoE data received
- [EC\\_E\\_MASTER\\_RED\\_STATE\\_INACTIVE](#) if Master Redundancy is configured and master is inactive

struct [ETHERCAT\\_MBOX\\_HEADER](#)

### Public Members

#### EC\_T\_WORD **wLength**

Following bytes (payload length)

#### EC\_T\_WORD **wAddress**

Station address of destination (READ) or source (WRITE). 0 == Master (ETHERCAT\_MBOX\_MASTER\_ADDRESS)

#### EC\_T\_BYT**e** **byChnPri**

Channel, Priority

**EC\_T\_BYTExTypCntRsvd**  
wMbxType, Counter, Rsvd

### emVoeRead() Example

```
dwRes = emVoeRead(dwInstanceId, dwSlaveId, abyVoeDataInBuf,
    sizeof(abyVoeDataInBuf), &dwDataOutLen, 5000);
if(dwRes != EC_E_NOERROR) {
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
        "emVoeRead(...): %s (0x%lx)\n", ecatGetText(dwRes), dwRes));
}
```

#### See also:

[emGetSlaveId\(\)](#)

### 6.14.4 emNotify - eMbxTferType\_VOE\_READ

The corresponding Slave ID can be found in pMbxTfer->dwTferId. The MBX data stored in pMbxTfer->pbyMbxTferData may have to be buffered by the client. After emNotify returns the pointer and thus the data is invalid. Access to the memory area pointed to by pMbxTfer-> pbyMbxTferData after returning from emNotify is illegal and the results are undefined.

#### emNotify - eMbxTferType\_VOE\_READ

##### Parameter

- pbyInBuf: [in] pMbxTfer - Pointer to a structure of type EC\_T\_MBXTFER, this structure contains the used mailbox transfer object . To retrieve this VoE mailbox data emVoeRead has to be called.
- dwInBufSize: [in] Size of the transfer object.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

### 6.14.5 emNotify - eMbxTferType\_VOE\_WRITE

VoE mailbox was successfully written to the VoE slave. The corresponding transfer ID can be found in pMbxTfer->dwTferId. The transfer result is stored in pMbxTfer->dwErrorCode.

#### emNotify - eMbxTferType\_VOE\_WRITE

##### Parameter

- 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.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

## 6.15 Raw command transfer

### 6.15.1 emTferSingleRawCmd

```
static EC_T_DWORD ecatTferSingleRawCmd (
    EC_T_BYTE byCmd,
    EC_T_DWORD dwMemoryAddress,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emTferSingleRawCmd (
    EC_T_DWORD dwInstanceID,
    EC_T_BYTE byCmd,
    EC_T_DWORD dwMemoryAddress,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen,
    EC_T_DWORD dwTimeout
)
```

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 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. 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. 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.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **byCmd** – [in] EtherCAT command type. EC\_CMD\_TYPE\_...
- **dwMemoryAddress** – [in] Slave memory address, depending on the command to be sent this is either a physical or logical address.
- **pbyData** – [in, out] Buffer containing or receiving transferred data
- **wLen** – [in] Number of bytes to transfer
- **dwTimeout** – [in] Timeout [ms]

#### Returns

- ***EC\_E\_NOERROR*** if successful
- ***EC\_E\_INVALIDSTATE*** if master isn't initialized
- ***EC\_E\_INVALIDPARM*** if dwInstanceID is out of range or the command is not supported or the timeout value is set to EC\_NOWAIT
- ***EC\_E\_TIMEOUT*** if dwTimeout elapsed during the API call
- ***EC\_E\_BUSY*** another transfer request is already pending or the master or the corresponding slave is currently changing its operational state

- *EC\_E\_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC\_E\_INVALIDSIZE* if the size of the complete command does not fit into a single Ethernet frame. The maximum amount of data to transfer must not exceed 1486 bytes
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive

The following EtherCAT commands are supported:

- eRawCmd\_AP RD Auto Increment Physical Read (avoid to use, see below)
- eRawCmd\_AP WR Auto Increment Physical Write (avoid to use, see below)
- eRawCmd\_AP RW Auto Increment Physical Read/Write (avoid to use, see below)
- eRawCmd\_F PRD Fixed addressed Physical Read
- eRawCmd\_F PWR Fixed addressed Physical Write
- eRawCmd\_F PRW Fixed addressed Physical Read/Write
- eRawCmd\_BR D Broadcast (wire or'ed) Read
- eRawCmd\_B WR Broadcast Write
- eRawCmd\_BR W Broadcast Read/Write
- eRawCmd\_L RD Logical Read
- eRawCmd\_L WR Logical Write
- eRawCmd\_L RW Logical Read/Write
- eRawCmd\_A RMW Auto Increment Physical Read, multiple Write

### 6.15.2 emCIntSendRawMbx

```
static EC_T_DWORD ecatCIntSendRawMbx (
    EC_T_DWORD dwClntId,
    EC_T_BYT E *pbyMbxCmd,
    EC_T_DWORD dwMbxCmdLen,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emCIntSendRawMbx (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwClntId,
    EC_T_BYT E *pbyMbxCmd,
    EC_T_DWORD dwMbxCmdLen,
    EC_T_DWORD dwTimeout
)
Send a raw mailbox command.
```

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClntId** – [in] Client ID
- **pbyMbxCmd** – [in] Buffer containing the raw mailbox command starting with mailbox header
- **dwMbxCmdLen** – [in] Length of pbyMbxCmd buffer
- **dwTimeout** – [in] Timeout [ms]

**Returns**

*EC\_E\_NOERROR* or error code

### 6.15.3 emClntQueueRawCmd

```
static EC_T_DWORD ecatClntQueueRawCmd (
    EC_T_DWORD dwClntId,
    EC_T_WORD wInvokeId,
    EC_T_BYTE byCmd,
    EC_T_DWORD dwMemoryAddress,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen
)
EC_T_DWORD emClntQueueRawCmd (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwClntId,
    EC_T_WORD wInvokeId,
    EC_T_BYTE byCmd,
    EC_T_DWORD dwMemoryAddress,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen
)
```

) 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 function queues a single EtherCAT command. Queued raw commands will be sent after sending cyclic process data values. If a timeout occurs the corresponding frame will be sent again, the timeout value and retry counter can be set using the master configuration parameters *EC\_T\_INIT\_MASTER\_PARMS.dwEcatCmdTimeout* and *EC\_T\_INIT\_MASTER\_PARMS.dwEcatCmdMaxRetries*.

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.

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClntId** – [in] Client ID to be notified (0 if all registered clients shall be notified).
- **wInvokeId** – [in] Invoke ID to reassign the results to the sent CMD
- **byCmd** – [in] EtherCAT command
- **dwMemoryAddress** – [in] Slave memory address, depending on the command to be sent this is either a physical or logical address
- **pbyData** – [in, out] Buffer containing or receiving transferred data. 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.

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range or the command is not supported

- *EC\_E\_BUSY* if the master or the corresponding slave is currently changing its operational state
- *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

The following EtherCAT commands are supported:

enum **EC\_T\_RAWCMD**

Values:

enumerator **eRawCmd\_APRD**  
Auto-Increment physical read

enumerator **eRawCmd\_APWR**  
Auto-Increment physical write

enumerator **eRawCmd\_APPRW**  
Auto-Increment physical read/write

enumerator **eRawCmd\_BRD**  
Broadcast (wire-or'ed) read

enumerator **eRawCmd\_BWR**  
Broadcast write

enumerator **eRawCmd\_BRW**  
Broadcast read/write

enumerator **eRawCmd\_LRD**  
Logical read

enumerator **eRawCmd\_LWR**  
Logical write

enumerator **eRawCmd\_LRW**  
Logical read/write

enumerator **eRawCmd\_ARMW**  
Auto-increment physical read, multiple write

enumerator **eRawCmd\_FPRD**  
Fixed address physical read

enumerator **eRawCmd\_FPWR**  
Fixed address physical write

enumerator **eRawCmd\_FPRW**  
Fixed address physical read/write

## 6.15.4 emQueueRawCmd

```
static EC_T_DWORD ecatQueueRawCmd (
    EC_T_WORD wInvokeId,
    EC_T_BYTE byCmd,
    EC_T_DWORD dwMemoryAddress,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen
)
EC_T_DWORD emQueueRawCmd (
    EC_T_DWORD dwInstanceID,
    EC_T_WORD wInvokeId,
    EC_T_BYTE byCmd,
    EC_T_DWORD dwMemoryAddress,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen
)
```

Transfers a raw EtherCAT command to one or multiple slaves.

All registered clients will be notified. This function may not be called from within the JobTask's context.

### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **wInvokeId** – [in] Invoke ID to reassign the results to the sent CMD
- **byCmd** – [in] EtherCAT command
- **dwMemoryAddress** – [in] Slave memory address, depending on the command to be sent this is either a physical or logical address.
- **pbyData** – [in, out] Buffer containing or receiving transferred data 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.

### Returns

*EC\_E\_NOERROR* or error code

### See also:

[emCIntQueueRawCmd \(\)](#)

## 6.15.5 emNotify - EC\_NOTIFY\_RAWCMD\_DONE

This notification is given when the response to an [emTransferSingleRawCmd \(\)](#) or [emCIntQueueRawCmd \(\)](#) is received.

### emNotify - EC\_NOTIFY\_RAWCMD\_DONE

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_RAWCMDRESPONSE\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

---

```
struct EC_T_RAWCMDRESPONSE_NTFY_DESC
```

#### Public Members

**EC\_T\_DWORD dwInvokeId**

[in] Invoke Id from callee. Only lower 16 bits are relevant

**EC\_T\_DWORD dwResult**

[in] EC\_E\_NOERROR on success, error code otherwise

**EC\_T\_DWORD dwWkc**

[in] Received working counter

**EC\_T\_DWORD dwCmdIdx**

[in] Command Index Field

**EC\_T\_DWORD dwAddr**

[in] Address Field

**EC\_T\_DWORD dwLength**

[in] Length of data portion (11 relevant bits)

**EC\_T\_BYT\* 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

## 6.16 EtherCAT Bus Scan

The acontis EtherCAT Master stack supports scanning the EtherCAT Bus to determine which devices are available. If a configuration was provided the connected slaves are validated against the given ENI.

#### See also:

[emScanBus \(\)](#)

### 6.16.1 emIoControl - EC\_IOCTL\_SB\_ENABLE

Enables Busscan support.

#### emIoControl - EC\_IOCTL\_SB\_ENABLE

##### Parameter

- 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: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

The Scanbus support is enabled by default.

## 6.16.2 emIoControl - EC\_IOCTL\_SB\_RESTART

This call will restart the bus scanning cycle. On completion the Notification *emNotify - EC\_NOTIFY\_SB\_STATUS* is given.

### emIoControl - EC\_IOCTL\_SB\_RESTART

**Parameter**

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

EC\_E\_NOERROR or error code

The timeout value given by *emIoControl - EC\_IOCTL\_SB\_ENABLE* will be used. This function may be called prior to running *emConfigureNetwork ()*. 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 *emExecJob ()* is called cyclically to trigger master state machines, timers, send acyc and receive frames accordingly.

## 6.16.3 emIoControl - EC\_IOCTL\_SB\_STATUS\_GET

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

### emIoControl - EC\_IOCTL\_SB\_STATUS\_GET

**Parameter**

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Pointer to EC\_T\_SB\_STATUS\_NTFY\_DESC.
- dwOutBufSize: [in] Size of the output buffer in bytes.
- pdwNumOutData: [out] Pointer to EC\_T\_DWORD. Amount of bytes written to the output buffer.

**Return**

EC\_E\_NOERROR or error code

**See also:**

*emNotify - EC\_NOTIFY\_SB\_STATUS*

## 6.16.4 emIoControl - EC\_IOCTL\_SB\_SET\_TOPOLOGY\_CHANGED\_DELAY

This call will set the topology changed delay value. The master will wait this duration in msec to react on appearing links in topology. The default value is 1000 ms.

### **emIoControl - EC\_IOCTL\_SB\_SET\_TOPOLOGY\_CHANGED\_DELAY**

#### **Parameter**

- 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: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### **Return**

EC\_E\_NOERROR or error code

## 6.16.5 emIoControl - EC\_IOCTL\_SB\_SET\_TOPOLOGY\_CHANGED\_DELAYS

This call will set the topology changed delay values individually. The master will wait individual durations in msec (0 msec: disabled) for slave ports, main link and red link to react on appearing links in topology. The default value is 1000 ms.

### **emIoControl - EC\_IOCTL\_SB\_SET\_TOPOLOGY\_CHANGED\_DELAYS**

#### **Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_TOPOLOGY\_CHANGED\_DELAYS containing the delay information in msec
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### **Return**

EC\_E\_NOERROR or error code

struct **EC\_T\_TOPOLOGY\_CHANGED\_DELAYS**

#### **Public Members**

##### **EC\_T\_DWORD dwSlavePort**

[in] Delay before opening slave port after link connection detected

##### **EC\_T\_DWORD dwMainLine**

[in] Delay before sending frames at main line after link connection detected

##### **EC\_T\_DWORD dwRedLine**

[in] Delay before sending frames at red line after link connection detected

EC\_T\_DWORD **adwReserved[5]**  
reserved

## 6.16.6 emIoControl - 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.

### emIoControl - EC\_IOCTL\_SB\_SET\_ERROR\_ON\_CROSSED\_LINES

#### Parameter

- 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: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

EC\_E\_NOERROR or error code

## 6.16.7 emIoControl - EC\_IOCTL\_SB\_SET\_ERROR\_ON\_LINEBREAK

This call will enable or disable bus mismatch if a line is broken in a redundant network. If enabled, line breaks in cable or junction redundant networks will lead to bus mismatch.

### emIoControl - EC\_IOCTL\_SB\_SET\_ERROR\_ON\_LINEBREAK

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_BOOL. EC\_TRUE: Return error code, EC\_FALSE: Return no error.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

EC\_E\_NOERROR or error code

Default: No error on line break.

Error codes:

- EC\_E\_REDLINEBREAK: Line break in a cable redundant network.
- EC\_E\_JUNCTION\_RED\_LINE\_BREAK: Line break in a junction redundant network.

## 6.16.8 emIoControl - EC\_IOCTL\_SB\_SET\_TOPOLOGY\_CHANGE\_AUTO\_MODE

**Warning:** 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.

### emIoControl - EC\_IOCTL\_SB\_SET\_TOPOLOGY\_CHANGE\_AUTO\_MODE

#### Parameter

- 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: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

EC\_E\_NOERROR or error code

In automatical mode, new slaves will be discovered automatically. In manual mode, after new slaves have been connected, a *emNotify - 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 *emIoControl - EC\_IOCTL\_SB\_ACCEPT\_TOPOLOGY\_CHANGE*

## 6.16.9 emIoControl - EC\_IOCTL\_SB\_ACCEPT\_TOPOLOGY\_CHANGE

**Warning:** This documentation is preliminary and is subject to change

This call will trigger a scan bus. On completion the Notification *emNotify - EC\_NOTIFY\_SB\_STATUS* is given.

### emIoControl - EC\_IOCTL\_SB\_ACCEPT\_TOPOLOGY\_CHANGE

#### Parameter

- pbyInBuf: [in] Should be set to EC\_NULL
- dwInBufSize: [in] Should be set to 0
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

EC\_E\_NOERROR or error code

This function may be called after a *emNotify - EC\_NOTIFY\_HC\_TOPOCHGDONE* notification was given if the automatical topology change mode was previously disabled using *emIoControl - 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 *emIoControl - EC\_IOCTL\_SB\_ENABLE* will be used. When issuing this IoControl, the application has to take care *emExecJob()* is called cyclically to trigger master state machines, timers, send acyc and receive frames accordingly.

## 6.16.10 emNotify - EC\_NOTIFY\_SB\_STATUS

Scan bus status notification.

This notification is enabled by default.

### **emNotify - EC\_NOTIFY\_SB\_STATUS**

#### **Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_SB\_STATUS\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

struct **EC\_T\_SB\_STATUS\_NTFY\_DESC**

#### **Public Members**

##### **EC\_T\_DWORD dwresultCode**

[in] EC\_E\_NOERROR: success EC\_E\_NOTREADY: no bus scan executed  
EC\_E\_BUSCONFIG\_MISMATCH: bus configuration mismatch Result of scanbus

##### **EC\_T\_DWORD dwSlaveCount**

[in] number of slaves connected to the bus

#### **See also:**

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the deactivation

## 6.16.11 emNotify - EC\_NOTIFY\_SB\_MISMATCH

This notification will be initiated if scan bus detects mismatch of connected slaves and configuration, due to unexpected slaves or missing mandatory slaves.

### **emNotify - EC\_NOTIFY\_SB\_MISMATCH**

#### **Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_SB\_MISMATCH\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

This notification is enabled by default. In case of permanent frame loss no slaves can be found although the slaves are connected.

struct **EC\_T\_SB\_MISMATCH\_DESC**

## Public Members

**EC\_T\_WORD wPrevFixedAddress**  
[in] Previous slave station address

**EC\_T\_WORD wPrevPort**  
[in] Previous slave station address

**EC\_T\_WORD wPrevAIncAddress**  
[in] Previous slave auto-increment address

**EC\_T\_WORD wBusAIncAddress**  
[in] Unexpected slave (bus) auto-inc address

**EC\_T\_DWORD dwBusVendorId**  
[in] Unexpected slave (bus) vendor ID

**EC\_T\_DWORD dwBusProdCode**  
[in] Unexpected slave (bus) product code

**EC\_T\_DWORD dwBusRevisionNo**  
[in] Unexpected slave (bus) revision number

**EC\_T\_DWORD dwBusSerialNo**  
[in] Unexpected slave (bus) serial number

**EC\_T\_WORD wBusFixedAddress**  
[in] Unexpected slave (bus) station address

**EC\_T\_BOOL bIdentificationError**  
[in] Identification command sent to slave but failed

**EC\_T\_WORD wIdentificationAdo**  
[in] Identification register

**EC\_T\_WORD wIdentificationVal**  
[in] last identification value read from slave according to the last used identification method

**EC\_T\_WORD wIdentificationValExpected**  
[in] Identification expected value

**EC\_T\_WORD wCfgFixedAddress**  
[in] Missing slave (config) station Address

**EC\_T\_WORD wCfgAIncAddress**  
[in] Missing slave (config) Auto-Increment Address

**EC\_T\_DWORD dwCfgVendorId**  
[in] Missing slave (config) Vendor ID

**EC\_T\_DWORD dwCfgProdCode**  
[in] Missing slave (config) Product code

**EC\_T\_DWORD dwCfgRevisionNo**  
[in] Missing slave (config) Revision Number

**EC\_T\_DWORD dwCfgSerialNo**  
 [in] Missing slave (config) Serial Number

**See also:**

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the deactivation

## 6.16.12 emNotify - EC\_NOTIFY\_SB\_DUPLICATE\_HC\_NODE

Scan bus mismatch was detected while scan because of a duplicated slave(s). An application get this notification if the there are two slaves on the network with the same product code, vendor ID and identification value (alias address or switch id).

### emNotify - EC\_NOTIFY\_SB\_DUPLICATE\_HC\_NODE

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_SB\_MISMATCH\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

The members of *EC\_T\_SB\_MISMATCH\_DESC* have the following meaning:

- *EC\_T\_SB\_MISMATCH\_DESC::wCfgFixedAddress* Duplicated slave (config) station Address
- *EC\_T\_SB\_MISMATCH\_DESC::wCfgAIncAddress* Duplicated slave (config) Auto-Increment Address.
- *EC\_T\_SB\_MISMATCH\_DESC::dwCfgVendorId* Duplicated slave (config) Vendor ID
- *EC\_T\_SB\_MISMATCH\_DESC::dwCfgProdCode* Duplicated slave (config) Product code
- *EC\_T\_SB\_MISMATCH\_DESC::dwCfgRevisionNo* Duplicated slave (config) Revision Number
- *EC\_T\_SB\_MISMATCH\_DESC::dwCfgSerialNo* Duplicated slave (config) Serial Number

## 6.16.13 emNotify - EC\_NOTIFY\_SLAVE\_PRESENCE

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

This notification is enabled by default.

### emNotify - EC\_NOTIFY\_SLAVE\_PRESENCE

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_SLAVE\_PRESENCE\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

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

struct **EC\_T\_SLAVE\_PRESENCE\_NTFY\_DESC**

**Public Members**

**EC\_T\_WORD wStationAddress**  
Slave station address

**EC\_T\_BYTE bPresent**  
EC\_TRUE: present , EC\_FALSE: absent

**See also:**

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED* for how to control the deactivation

**6.16.14 emNotify - EC\_NOTIFY\_SLAVES\_PRESENCE**

This notification collects notifications of the type *emNotify - EC\_NOTIFY\_SLAVE\_PRESENCE*. Notification is given either upon completion or when master status is changed, whichever comes first. Disconnecting slaves from the network, turning them off, or having a bad connection can lead to this notification.

This notification is disabled by default.

**emNotify - EC\_NOTIFY\_SLAVES\_PRESENCE****Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_SLAVES\_PRESENCE\_NTFY\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

struct **EC\_T\_SLAVES\_PRESENCE\_NTFY\_DESC**

**Public Members**

**EC\_T\_WORD wCount**  
Number of slave presence notifications

**EC\_T\_SLAVE\_PRESENCE\_NTFY\_DESC**  
**SlavePresence[MAX\_SLAVES\_PRESENCE\_NTFY\_ENTRIES]**  
slave presence descriptions

**See also:**

*emIoControl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED*

## 6.16.15 emNotify - EC\_NOTIFY\_LINE\_CROSSED

Cable swapping detected. All slaves' port 0 must lead to Master.

### emNotify - EC\_NOTIFY\_LINE\_CROSSED

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_LINE\_CROSSED\_DESC
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

struct **EC\_T\_LINE\_CROSSED\_DESC**

#### Public Members

**EC\_T\_SLAVE\_PROP SlaveProp**  
slave properties

**EC\_T\_WORD wInputPort**  
port where frame was received

## 6.16.16 emNotify - EC\_NOTIFY\_SLAVE\_NOTSUPPORTED

Is currently generated during Bus Scan if [emConfigureNetwork\(\)](#) (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).

### emNotify - EC\_NOTIFY\_SLAVE\_NOTSUPPORTED

#### Parameter

- 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: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

struct **EC\_T\_SLAVE\_NOTSUPPORTED\_DESC**

**Public Members**

***EC\_T\_SLAVE\_PROP SlaveProp***  
Slave properties

**6.16.17 emNotify - EC\_NOTIFY\_FRAMELOSS\_AFTER\_SLAVE**

Is currently generated and automatically handled during *emRescueScan()* if opening port destroys communication (frameloss). This notification should only print a log message or be ignored.

**emNotify - EC\_NOTIFY\_FRAMELOSS\_AFTER\_SLAVE****Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_ERROR\_NOTIFICATION\_DESC containing EC\_T\_FRAMELOSS\_AFTER\_SLAVE\_NTFY\_DESC.
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

```
struct EC_T_FRAMELOSS_AFTER_SLAVE_NTFY_DESC
```

**Public Members**

***EC\_T\_SLAVE\_PROP SlaveProp***  
slave properties

***EC\_T\_WORD wPort***  
port

**6.16.18 emNotify - 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 belong to the Feature Pack Hot Connect.

**6.16.19 emIoControl - EC\_IOCTL\_SB\_NOTIFY\_UNEXPECTED\_BUS\_SLAVES**

Specifies if unexpected bus slaves must be notified as bus mismatch.

**emIoControl - EC\_IOCTL\_SB\_NOTIFY\_UNEXPECTED\_BUS\_SLAVES****Parameter**

- 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: [out] Should be set to EC\_NULL

- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**Return**

`EC_E_NOERROR` or error code

### 6.16.20 emIsTopologyChangeDetected

```
static EC_T_DWORD ecatIsTopologyChangeDetected (
    EC_T_BOOL *pbTopologyChangeDetected
)
EC_T_DWORD emIsTopologyChangeDetected (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL *pbTopologyChangeDetected
)
```

Returns whether topology change detected.

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pbTopologyChangeDetected** – [out] Pointer to EC\_T\_BOOL value: EC\_TRUE if Topology Change Detected, EC\_FALSE if not.

**Returns**

- *EC\_E\_NOERROR* if successful
- *EC\_E\_INVALIDSTATE* if master isn't initialized
- *EC\_E\_INVALIDPARAM* if dwInstanceID is out of range
- *EC\_E\_MASTER\_RED\_STATE\_INACTIVE* if Master Redundancy is configured and master is inactive

### 6.16.21 emNotify - EC\_NOTIFY\_HC\_TOPOCHGDONE

This notification is raised when topology change has completely processed.

#### emNotify - EC\_NOTIFY\_HC\_TOPOCHGDONE

**Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_DWORD (EC\_E\_NOERROR on success, Error code otherwise)
- dwInBufSize: [in] sizeof(EC\_T\_DWORD).
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

The notification is raised when the slaves have been detected and DC initialized.

## 6.16.22 emIoControl - EC\_IOCTL\_SB\_SET\_NO\_DC\_SLAVES\_AFTER\_JUNCTION

Declares that no DC slaves are located after junction

### emIoControl - EC\_IOCTL\_SB\_SET\_NO\_DC\_SLAVES\_AFTER\_JUNCTION

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_BOOL variable. If set to EC\_TRUE the hidden slave detection and the junction redundancy specific propagation delay measurement are not executed
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes.
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

#### Return

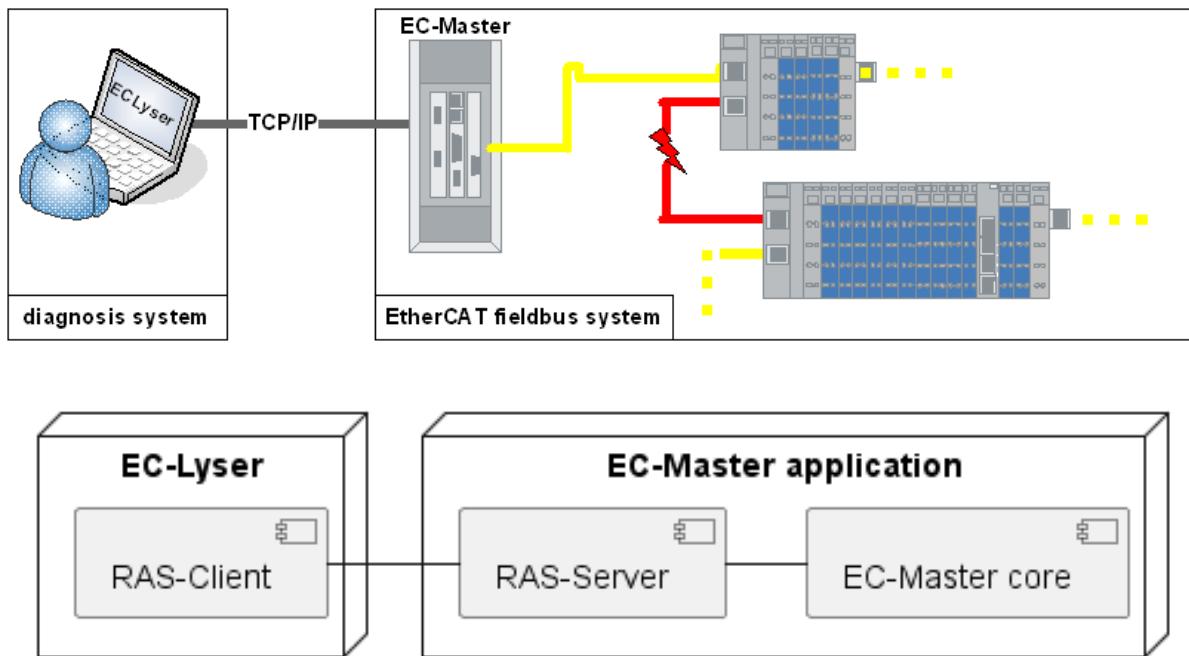
EC\_E\_NOERROR or error code

Calling this IOCTL if DC slaves are located in or after a junction redundancy segment will generate an undefined behavior.

## 7 RAS-Server for EC-Lyser and EC-Engineer

### 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 the application is given with the example application EcMasterDemo, which in case is pre-configured to listen for EC-Lyser on TCP Port 6000 when command line parameter `EcMasterDemo -sp` is given.

To clarify the steps, which are needed within a custom application, a developer may use the following pseudo-code segment as a point of start. The Remote API Server library `EcMasterRasServer.lib` (or respectively `EcMasterRasServer.a`) must be linked.

## 7.2 Application programming interface, reference

### 7.2.1 emRasSrvStart

```
EC_T_DWORD EC_NAMESPACE::emRasSrvStart (
    ECMASTER_RAS_T_SRVPARMS *pParms,
    EC_T_PVOID *ppHandle
)
```

Initializes and start remote API Server Instance.

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

#### Parameters

- **pParms** – [in] Server start-up parameters

- **ppHandle** – [out] Handle to opened instance, used for ctrl access

**Returns**

EC\_E\_NOERROR or error code

struct **ECMASTERRAS\_T\_SRVPARMS**

RAS Server init parameters.

**Public Members**EC\_T\_DWORD **dwSignature**

[in] Set to ECMASTERRASSERVER\_SIGNATURE

EC\_T\_DWORD **dwSize**

[in] Set to sizeof(ECMASTERRAS\_T\_SRVPARMS)

EC\_T\_LOG\_PARMS **LogParms**

[in] Logging parameters

EC\_T\_IPADDR **oAddr**

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

EC\_T\_WORD **wPort**

[in] Remote Access Server (RAS) listen port

EC\_T\_WORD **wMaxClientCnt**

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

EC\_T\_DWORD **dwCycleTime**

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

EC\_T\_DWORD **dwCommunicationTimeout**

[in] timeout before automatically closing connection

EC\_T\_CPUSET **oAcceptorThreadCpuAffinityMask**

[in] Acceptor Thread CPU affinity mask

EC\_T\_DWORD **dwAcceptorThreadPrio**

[in] Acceptor Thread Priority

EC\_T\_DWORD **dwAcceptorThreadStackSize**

[in] Acceptor Thread Stack Size

EC\_T\_CPUSET **oClientWorkerThreadCpuAffinityMask**

[in] Client Worker Thread CPU affinity mask

EC\_T\_DWORD **dwClientWorkerThreadPrio**

[in] Client Worker Thread Priority

EC\_T\_DWORD **dwClientWorkerThreadStackSize**

[in] Client Worker Thread Stack Size

EC\_T\_DWORD **dwMaxQueuedNotificationCnt**

[in] Amount of concurrently queue able Notifications

**EC\_T\_DWORD dwMaxParallelMbxTferCnt**  
 [in] Amount of concurrent active mailbox transfers

**EC\_PF\_NOTIFY pfnRasNotify**  
 [in] Function pointer called to notify error and status information generated by Remote API Layer

**EC\_T\_VOID \*pvRasNotifyCtxt**  
 [in] Notification context returned while calling pfNotification

**EC\_T\_DWORD dwCycErrInterval**  
 [in] Interval which allows cyclic Notifications

**EC\_T\_DWORD dwMaxQueuedNotificationSize**  
 [in] Size of concurrent active mailbox transfers

**EC\_PF\_CHECK\_TOKEN pfCheckToken**  
 [in] Function pointer called to check token

**EC\_T\_VOID \*pvCheckTokenContext**  
 [in] Check token context

```
union EC_T_IPADDR
{
  #include <EthernetServices.h>
```

### Public Members

**EC\_T\_INNER\_IPADDR sAddr**  
 IPv4 address (endianness independent)

**EC\_T\_DWORD dwAddr**  
 Reserved, use EC\_T\_IPADDR::sAddr.by instead. OS-Layer socket API calls (SOCK-ADDR\_IN::sin\_addr)

```
struct EC_T_INNER_IPADDR
```

### Public Members

**EC\_T\_BYTE by[4]**  
 IPv4 address (endianness independent)

## 7.2.2 emRasSrvStop

```
EC_T_DWORD EC_NAMESPACE::emRasSrvStop (
    EC_T_PVOID pvHandle,
    EC_T_DWORD dwTimeout
)
{
    Stop and de-initialize remote API Server Instance.
}
```

### Parameters

- **pvHandle** – [in] Handle to previously started Server

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

**Returns**

EC\_E\_NOERROR or error code

### 7.2.3 emRasNotify - xxx

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

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

### 7.2.4 emRasNotify - ECMASTERRAS\_NOTIFY\_CONNECTION

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

**emRasNotify - ECMASTERRAS\_T\_CONNNOTIFYDESC****Parameter**

- pbyInBuf: [in] Pointer to data of type ECMASTERRAS\_T\_CONNNOTIFYDESC
- dwInBufSize: [in] Size of the input buffer in bytes
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

```
struct ECMASTERRAS_T_CONNNOTIFYDESC
```

**Public Members****EC\_T\_DWORD dwCause**

[in] Cause of state connection state change

**EC\_T\_DWORD dwCookie**

[in] Unique identification cookie of connection instance.

### 7.2.5 emRasNotify - ECMASTERRAS\_NOTIFY\_REGISTER

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

**emRasNotify - ECMASTERRAS\_NOTIFY\_REGISTER****Parameter**

- pbyInBuf: [in] Pointer to data of type ECMASTERRAS\_T\_REGNOTIFYDESC
- dwInBufSize: [in] Size of the input buffer in bytes
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

---

```
struct ECMASTERRAS_T_REGNOTIFYDESC
```

#### Public Members

**EC\_T\_DWORD dwCookie**

[in] Unique identification cookie of connection instance

**EC\_T\_DWORD dwResult**

[in] Result of registration request

**EC\_T\_DWORD dwInstanceId**

[in] Master Instance client registered to

**EC\_T\_DWORD dwClientId**

[in] Client ID of registered client

### 7.2.6 emRasNotify - ECMASTERRAS\_NOTIFY\_UNREGISTER

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

#### emRasNotify - ECMASTERRAS\_NOTIFY\_UNREGISTER

##### Parameter

- pbyInBuf: [in] Pointer to data of type ECMASTERRAS\_T\_REGNOTIFYDESC
- dwInBufSize: [in] Size of the input buffer in bytes
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

##### See also:

[ECMASTERRAS\\_T\\_REGNOTIFYDESC](#)

### 7.2.7 emRasNotify - ECMASTERRAS\_NOTIFY\_MARSHALERROR

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

#### emRasNotify - ECMASTERRAS\_NOTIFY\_MARSHALERRORDESC

##### Parameter

- pbyInBuf: [in] Pointer to data of type ECMASTERRAS\_T\_MARSHALERRORDESC
- dwInBufSize: [in] Size of the input buffer in bytes
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

---

```
struct ECMASTERRAS_T_MARSHALERRORDESC
```

**Public Members****EC\_T\_DWORD dwCookie**

[in] Unique identification cookie of connection instance

**EC\_T\_DWORD dwCause**

[in] Cause of the command marshalling error

**EC\_T\_DWORD dwLenStatCmd**

[in] Length faulty command

**EC\_T\_DWORD dwCommandCode**

[in] Command code of faulty command

**7.2.8 emRasNotify - ECMASTERRAS\_NOTIFY\_ACKERROR**

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

**emRasNotify - ECMASTERRAS\_NOTIFY\_ACKERROR****Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_DWORD containing error code
- dwInBufSize: [in] Size of the input buffer in bytes
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

**7.2.9 emRasNotify - ECMASTERRAS\_NOTIFY\_NONOTIFYMEMORY**

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

**emRasNotify - ECMASTERRAS\_NOTIFY\_NONOTIFYMEMORY****Parameter**

- pbyInBuf: [in] Pointer to EC\_T\_DWORD containing unique identification cookie of connection instance
- dwInBufSize: [in] Size of the input buffer in bytes
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

## 7.2.10 emRasNotify - ECMASTERRAS\_NOTIFY\_STDNOTIFYMEMORYSMALL

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

### emRasNotify - ECMASTERRAS\_NOTIFY\_STDNOTIFYMEMORYSMALL

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_DWORD containing unique identification cookie of connection instance
- dwInBufSize: [in] Size of the input buffer in bytes
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

## 7.2.11 emRasNotify - ECMASTERRAS\_NOTIFY\_MBXNOTIFYMEMORYSMALL

Notification given, when buffersize for Mailbox notifications available in pre-allocated notification store are too small to carry a specific notification. This points to a configuration error. This is a serious error. If this error is given, Mailbox Transfer objects may have been become out of sync and therefore no more valid usable. Mailbox notifications should be dimensioned correctly see emRasSrvStart ()

### emRasNotify - ECMASTERRAS\_NOTIFY\_MBXNOTIFYMEMORYSMALL

#### Parameter

- pbyInBuf: [in] Pointer to EC\_T\_DWORD containing unique identification cookie of connection instance
- dwInBufSize: [in] Size of the input buffer in bytes
- pbyOutBuf: [out] Should be set to EC\_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC\_NULL

## 8 Error Codes

### 8.1 Groups

No.	Group	Abbr.	Description
1	Application Error	APP	Error within application, running the master E.g. API function call with invalid parameters
2	EtherCAT network information file problem	ENI	Master configuration XML file mismatches slave configuration on bus E.g. Bus Topology Scan cannot detect all slaves configured within network information file
3	Master parameter configuration	CFG	Master configuration parameters erroneous E.g. mailbox command queue not large enough
4	Bus/Slave Error	SLV	Slave error E.g. Working Counter Error
5	Real-time Ethernet Driver	LLA	Real-time Ethernet Driver error (network interface driver) E.g. Intel Pro/1000 NIC could not be found
6	Remote API	RAS	Remote API error E.g. connection to Remote API server is not possible from client
7	Internal software error	ISW	Master internal error E.g. Master state machine in undefined state
8	DC Master Sync	DCM	DC slave and host time synchronization
9	Pass-Through-Server	PTS	Initialization/De-Initialization errors
10	System Setup	SYS	Errors from Operating System or obviously due to System Setup

### 8.2 Generic Error Codes

**EC\_E\_NOERROR**

0x00000000: No Error

**EC\_E\_ERROR**

0x98110000: Unspecific Error

**EMRAS\_E\_ERROR**

0x98110180: Unspecific RAS Error

**EC\_E\_NOTSUPPORTED**

0x98110001: APP: Feature not supported (e.g. function or property not available)

**EC\_E\_INVALIDINDEX**

0x98110002: APP: Invalid index (e.g. CoE: invalid SDO index)

**EC\_E\_INVALIDOFFSET**

0x98110003: ISW: Invalid offset (e.g. invalid offset while accessing Process Data Image)

**EC\_E\_CANCEL**

0x98110004: APP: Cancel (e.g. master should abort current mailbox transfer)

**EC\_E\_INVALIDSIZE**

0x98110005: APP: Invalid size

**EC\_E\_INVALIDDATA**

0x98110006: ISW: Invalid data (multiple error sources)

**EC\_E\_NOTREADY**

0x98110007: ISW: Not ready (multiple error sources)

**EC\_E\_BUSY**

0x98110008: APP: Busy (e.g. stack is busy currently and not available to process the API request. The function may be called again later)

**EC\_E\_ACYC\_FRM\_FREEQ\_EMPTY**

0x98110009: ISW: Cannot queue acyclic EtherCAT command (Acyclic command queue is full. Possible solution: Increase of configuration value dwMaxQueuedEthFrames)

**EC\_E\_NOMEMORY**

0x9811000A: CFG: No memory left (e.g. memory full / fragmented))

**EC\_E\_INVALIDPARAM**

0x9811000B: APP: Invalid parameter (e.g. API function called with erroneous parameter set)

**EC\_E\_NOTFOUND**

0x9811000C: APP: Not found (e.g. Network Information File ENI not found or API called with invalid slave ID)

**EC\_E\_DUPLICATE**

0x9811000D: ISW: Duplicated fixed address detected (handled internally)

**EC\_E\_INVALIDSTATE**

0x9811000E: ISW: Invalid state (master not initialized or not configured)

**EC\_E\_TIMER\_LIST\_FULL**

0x9811000F: ISW: Cannot add slave to timer list (slave timer list full)

**EC\_E\_TIMEOUT**

0x98110010: Timeout

**EC\_E\_OPENFAILED**

0x98110011: ISW: Open failed

**EC\_E\_SENDFAILED**

0x98110012: LLA: Frame send failed

**EC\_E\_INSERTMAILBOX**

0x98110013: CFG: Insert Mailbox error (internal limit MAX\_QUEUED\_COE\_CMDS: 20)

**EC\_E\_INVALIDCMD**

0x98110014: ISW: Invalid Command (Unknown mailbox command code)

**EC\_E\_UNKNOWN\_MBX\_PROTOCOL**

0x98110015: ISW: Unknown Mailbox Protocol Command (Unknown Mailbox protocol or mailbox command with unknown protocol association)

**EC\_E\_ACCESSDENIED**

0x98110016: ISW: Access Denied (e.g. master internal software error)

**EC\_E\_IDENTIFICATIONFAILED**

0x98110017: ENI: Identification failed (e.g. identification command failed)

**EC\_E\_LOCK\_CREATE\_FAILED**

0x98110018: SYS: Create lock failed (e.g. OsCreateLockTyped failed)

**EC\_E\_PRODKEY\_INVALID**

0x9811001A: CFG: Product Key Invalid (e.g. application using protected version of the stack, which stops operation after the evaluation time limit reached if a license is not provided)

**EC\_E\_WRONG\_FORMAT**

0x9811001B: ENI: Wrong configuration format (e.g. Network information file empty or malformed), SLV: Malformed EEPROM content

**EC\_E\_FEATURE\_DISABLED**

0x9811001C: APP: Feature disabled (e.g. Application tried to perform a missing or disabled API function)

**EC\_E\_SHADOW\_MEMORY**

0x9811001D: Shadow memory requested in wrong mode

**EC\_E\_BUSCONFIG\_MISMATCH**

0x9811001E: ENI: Bus configuration mismatch (e.g. Network information file and currently connected bus topology does not match)

**EC\_E\_CONFIGDATAREAD**

0x9811001F: ENI: Error reading configuration file (e.g. Network information file could not be read)

**EC\_E\_ENI\_NO\_SAFEOP\_OP\_SUPPORT**

0x98110020: Configuration doesn't support SAFEOP and OP requested state

**EC\_E\_XML\_CYCCMDS\_MISSING**

0x98110021: ENI: Cyclic commands are missing (e.g. Network information file does not contain cyclic commands)

**EC\_E\_XML\_ALSTATUS\_READ\_MISSING**

0x98110022: ENI: AL\_STATUS register read missing in XML file for at least one state (e.g. Read of AL Status register is missing in cyclic part of given network information file)

**EC\_E\_MCSM\_FATAL\_ERROR**

0x98110023: ISW: Fatal internal McSm (master control state machine is in an undefined state)

**EC\_E\_SLAVE\_ERROR**

0x98110024: SLV: Slave error (e.g. A slave error was detected. See also EC\_NOTIFY\_STATUS\_SLAVE\_ERROR and EC\_NOTIFY\_SLAVE\_ERROR\_STATUS\_INFO)

**EC\_E\_FRAME\_LOST**

0x98110025: SLV: Frame lost, IDX mismatch (EtherCAT *frame(s)* lost on bus, means the response was not received. In case this error shows frequently a problem with the wiring could be the cause)

**EC\_E\_CMD\_MISSING**

0x98110026: SLV: At least one EtherCAT command is missing in the received frame (e.g. received EtherCAT frame incomplete)

**EC\_E\_CYCCMD\_WKC\_ERROR**

0x98110027: Cyclic command WKC error

**EC\_E\_INVALID\_DCL\_MODE**

0x98110028: APP: IOCTL EC\_IOCTL\_DC\_LATCH\_REQ\_LTIMVALS invalid in DCL auto read mode (this function cannot be used if DC Latching is running in mode “Auto Read”)

**EC\_E\_AI\_ADDRESS**

0x98110029: SLV: Auto increment address increment mismatch (e.g. Network information file and bus topology doesn't match any more. Error shows only, if a already recognized slave isn't present any more)

**EC\_E\_INVALID\_SLAVE\_STATE**

0x9811002A: APP: Slave in invalid state, e.g. not in OP (API not callable in this state) (mailbox commands are not allowed in current slave state)

**EC\_E\_SLAVE\_NOT\_ADDRESSABLE**

0x9811002B: SLV: Station address lost (or slave missing) - FPRD to AL\_STATUS failed (e.g. Slave had a power cycle)

**EC\_E\_CYC\_CMDS\_OVERFLOW**

0x9811002C: ENI: Too many cyclic commands in XML configuration file (e.g. [EC\\_T\\_INIT\\_MASTER\\_PARMS.dwMaxAcycFramesQueued](#) too small)

**EC\_E\_LINK\_DISCONNECTED**

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

**EC\_E\_MASTERCORE\_INACCESSIBLE**

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

**EC\_E\_COE\_MBXSND\_WKC\_ERROR**

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

**EC\_E\_COE\_MBXRCV\_WKC\_ERROR**

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

**EC\_E\_NO\_MBX\_SUPPORT**

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

**EC\_E\_NO\_COE\_SUPPORT**

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

**EC\_E\_NO\_EOE\_SUPPORT**

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

**EC\_E\_NO\_FOE\_SUPPORT**

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

**EC\_E\_NO\_SOE\_SUPPORT**

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

**EC\_E\_NO\_VOE\_SUPPORT**

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

**EC\_E\_EVAL\_VIOLATION**

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

**EC\_E\_EVAL\_EXPIRED**

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

**EC\_E\_LICENSE\_MISSING**

0x98110039: License key invalid or missing

**EC\_E\_CFGFILENOTFOUND**

0x98110070: CFG: Master configuration not found (e.g. path to master configuration file (XML) was wrong or the file is not available)

**EC\_E\_EEPROMREADERROR**

0x98110071: SLV: Command error while EEPROM upload (read slave EEPROM)

**EC\_E\_EEPROMWRITEERROR**

0x98110072: SLV: Command error while EEPROM download (write slave EEPROM)

**EC\_E\_XML\_CYCCMDS\_SIZEISMATCH**

0x98110073: ENI: Cyclic command wrong size (too long) (size in master configuration file (XML) does not match size of process data)

**EC\_E\_XML\_INVALID\_INP\_OFF**

0x98110074: ENI: Invalid input offset in cyclic command, please check InputOffs

**EC\_E\_XML\_INVALID\_OUT\_OFF**

0x98110075: ENI: Invalid output offset in cyclic command, please check OutputOffs

**EC\_E\_PORTCLOSE**

0x98110076: Port close failed

**EC\_E\_PORTOPEN**

0x98110077: Port open failed

**EC\_E\_SLAVE\_NOT\_PRESENT**

0x9811010E: APP / SLV: command not executed (slave not present on bus) (e.g. slave disappeared or was never present)

**EC\_E\_EEPROMRELOADERROR**

0x98110110: Command error while EEPROM reload

**EC\_E\_SLAVECTRLRESETERROR**

0x98110111: Command error while Reset Slave Controller

**EC\_E\_SYSDRIVERMISSING**

0x98110112: SYS: Cannot open system driver (e.g. system driver was not loaded)

**EC\_E\_BUSCONFIG\_TOPOCHANGE**

0x9811011E: Bus configuration not detected, Topology changed (e.g. Topology changed while scanning bus)

**EC\_E\_EOE\_MBX\_WKC\_ERROR**

0x9811011F: EoE: Mailbox receive: working counter

**EC\_E\_FOE\_MBX\_WKC\_ERROR**

0x98110120: FoE: Mailbox receive: working counter

**EC\_E\_SOE\_MBX\_WKC\_ERROR**

0x98110121: SoE: mailbox receive: working counter

**EC\_E\_AOE\_MBX\_WKC\_ERROR**

0x98110122: AoE: Mailbox receive: working counter

**EC\_E\_VOE\_MBX\_WKC\_ERROR**

0x98110123: SLV: VoE mailbox send: working counter (VoE mailbox couldn't be written)

**EC\_E\_EEPROMASSIGNERROR**

0x98110124: SLV: EEPROM assignment failed

**EC\_E\_MBX\_ERROR\_TYPE**

0x98110125: SLV: Unknown mailbox error code received in mailbox

**EC\_E\_REDLINEBREAK**

0x98110126: SLV: Redundancy line break (e.g. cable break between slaves or between master and first slave)

**EC\_E\_XML\_INVALID\_CMD\_WITH\_RED**

0x98110127: ENI: Invalid EtherCAT command in cyclic frame with redundancy (e.g. BRW commands are not allowed with redundancy)

**EC\_E\_XML\_PREV\_PORT\_MISSING**

0x98110128: ENI: <PreviousPort>-tag is missing (e.g. if the auto increment address is not the first slave on the bus we check if a previous port tag OR a hot connect tag is available)

**EC\_E\_XML\_DC\_CYCCMDS\_MISSING**

0x98110129: DC enabled and DC cyclic commands missing (e.g. access to 0x0900)

**EC\_E\_DLSTATUS\_IRO\_TOPOCHANGED**

0x98110130: SLV: Data link (DL) status interrupt because of changed topology (automatically handled by master)

**EC\_E\_PTS\_IS\_NOT\_RUNNING**

0x98110131: PTS: Pass Through Server is not running (Pass-Through-Server was tried to be enabled/disabled or stopped without being started)

**EC\_E\_PTS\_IS\_RUNNING**

0x98110132: PTS: Pass Through Server is running (obsolete, replaced by EC\_E\_ADS\_IS\_RUNNING)

**EC\_E\_ADS\_IS\_RUNNING**

0x98110132: PTS: ADS adapter (Pass Through Server) is running (API call conflicts with ADS state (running))

**EC\_E\_PTS\_THREAD\_CREATE\_FAILED**

0x98110133: PTS: Could not start the Pass Through Server

**EC\_E\_PTS\_SOCK\_BIND\_FAILED**

0x98110134: PTS: The Pass Through Server could not bind the IP address with a socket (e.g. Possibly because the IPaddress (and Port) is already in use or the IP-address does not exist)

**EC\_E PTS NOT ENABLED**

0x98110135: PTS: The Pass Through Server is running but not enabled

**EC\_E PTS\_LL\_MODE\_NOT\_SUPPORTED**

0x98110136: PTS: The Link Layer mode is not supported by the Pass Through Server (e.g. The Master is running in interrupt mode but the Pass-Through-Server only supports polling mode)

**EC\_E VOE\_NO\_MBX RECEIVED**

0x98110137: SLV: No VoE mailbox received yet from specific slave

**EC\_E DC\_REF\_CLOCK\_SYNC\_OUT\_UNIT\_DISABLED**

0x98110138: DC (time loop control) unit of reference clock disabled

**EC\_E DC\_REF\_CLOCK\_NOT\_FOUND**

0x98110139: SLV: Reference clock not found! May happen if reference clock is removed from network.

**EC\_E MBX\_CMD\_WKC\_ERROR**

0x9811013B: SLV: Mailbox command working counter error (e.g. Mailbox init command Retry Count exceeded)

**EC\_E NO\_AOE\_SUPPORT**

0x9811013C: APP / SLV: AoE: Protocol not supported (e.g. Application calls AoE-API although not implemented at slave)

**EC\_E AOE\_INV\_RESPONSE\_SIZE**

0x9811013D: AoE: Invalid AoE response received

**EC\_E AOE\_ERROR**

0x9811013E: AoE: Common AoE device error

**EC\_E AOE\_SRVNOTSUPP**

0x9811013F: AoE: Service not supported by server

**EC\_E AOE\_INVALIDGRP**

0x98110140: AoE: Invalid index group

**EC\_E AOE\_INVALIDOFFSET**

0x98110141: AoE: Invalid index offset

**EC\_E AOE\_INVALIDACCESS**

0x98110142: AoE: Reading/writing not permitted

**EC\_E AOE\_INVALIDSIZE**

0x98110143: AoE: Parameter size not correct

**EC\_E AOE\_INVALIDDATA**

0x98110144: AoE: Invalid parameter *value(s)*

**EC\_E AOE\_NOTREADY**

0x98110145: AoE: Device not in a ready state

**EC\_E AOE\_BUSY**

0x98110146: AoE: Device busy

**EC\_E\_AOE\_INVALIDCONTEXT**

0x98110147: AoE: Invalid context

**EC\_E\_AOE\_NOMEMORY**

0x98110148: AoE: Out of memory

**EC\_E\_AOE\_INVALIDPARM**0x98110149: AoE: Invalid parameter *value(s)***EC\_E\_AOE\_NOTFOUND**

0x9811014A: AoE: Not found

**EC\_E\_AOE\_SYNTAX**

0x9811014B: AoE: Syntax error in command or file

**EC\_E\_AOE\_INCOMPATIBLE**

0x9811014C: AoE: Objects do not match

**EC\_E\_AOE\_EXISTS**

0x9811014D: AoE: Object already exists

**EC\_E\_AOE\_SYMBOLNOTFOUND**

0x9811014E: AoE: Symbol not found

**EC\_E\_AOE\_SYMBOLVERSIONINVALID**

0x9811014F: AoE: Symbol version invalid

**EC\_E\_AOE\_INVALIDSTATE**

0x98110150: AoE: Server in invalid state

**EC\_E\_AOE\_TRANSMODENOTSUPP**

0x98110151: AoE: AdsTransMode not supported

**EC\_E\_AOE\_NOTIFYHNDINVALID**

0x98110152: AoE: Notification handle invalid

**EC\_E\_AOE\_CLIENTUNKNOWN**

0x98110153: AoE: Notification client not registered

**EC\_E\_AOE\_NOMOREHDLS**

0x98110154: AoE: No more notification handles

**EC\_E\_AOE\_INVALIDWATCHSIZE**

0x98110155: AoE: Size for watch too big

**EC\_E\_AOE\_NOTINIT**

0x98110156: AoE: Device not initialized

**EC\_E\_AOE\_TIMEOUT**

0x98110157: AoE: Device has a timeout

**EC\_E\_AOE\_NOINTERFACE**

0x98110158: AoE: Query interface failed

**EC\_E\_AOE\_INVALIDINTERFACE**

0x98110159: AoE: Wrong interface required

**EC\_E\_AOE\_INVALIDCLSID**

0x9811015A: AoE: Class ID invalid

**EC\_E\_AOE\_INVALIDOBJID**

0x9811015B: AoE: Object ID invalid

**EC\_E\_AOE\_PENDING**

0x9811015C: AoE: Request pending

**EC\_E\_AOE\_ABORTED**

0x9811015D: AoE: Request aborted

**EC\_E\_AOE\_WARNING**

0x9811015E: AoE: Signal warning

**EC\_E\_AOE\_INVALIDARRAYIDX**

0x9811015F: AoE: Invalid array index

**EC\_E\_AOE\_SYMBOLNOTACTIVE**

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

**EC\_E\_AOE\_ACCESSDENIED**

0x98110161: AoE: Access denied

**EC\_E\_AOE\_INTERNAL**

0x98110162: AoE: Internal error

**EC\_E\_AOE\_TARGET\_PORT\_NOT\_FOUND**

0x98110163: AoE: Target port not found

**EC\_E\_AOE\_TARGET\_MACHINE\_NOT\_FOUND**

0x98110164: AoE: Target machine not found

**EC\_E\_AOE\_UNKNOWN\_CMD\_ID**

0x98110165: AoE: Unknown command ID

**EC\_E\_AOE\_PORT\_NOT\_CONNECTED**

0x98110166: AoE: Port not connected

**EC\_E\_AOE\_INVALID\_AMS\_LENGTH**

0x98110167: AoE: Invalid AMS length

**EC\_E\_AOE\_INVALID\_AMS\_ID**

0x98110168: AoE: invalid AMS Net ID

**EC\_E\_AOE\_PORT\_DISABLED**

0x98110169: AoE: Port disabled

**EC\_E\_AOE\_PORT\_CONNECTED**

0x9811016A: AoE: Port already connected

**EC\_E\_AOE\_INVALID\_AMS\_PORT**

0x9811016B: AoE: Invalid AMS port

**EC\_E\_AOE\_NO\_MEMORY**

0x9811016C: AoE: No memory

**EC\_E\_AOE\_VENDOR\_SPECIFIC**

0x9811016D: AoE: Vendor specific AoE device error

**EC\_E\_XML\_AOE\_NETID\_INVALID**

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

**EC\_E\_MAX\_BUS\_SLAVES\_EXCEEDED**

0x9811016F: CFG: Error: Maximum number of bus slave has been exceeded (The maximum number of preallocated bus slave objects are too small. The maximum number can be adjusted by the master initialization parameter EC\_T\_INITMASTERPARMS.dwMaxBusSlaves)

**EC\_E\_MBXERR\_SYNTAX**

0x98110170: SLV: Mailbox error: Syntax of 6 octet Mailbox header is wrong (Slave error mailbox return value: 0x01)

**EC\_E\_MBXERR\_UNSUPPORTEDPROTOCOL**

0x98110171: SLV: Mailbox error: The Mailbox protocol is not supported (Slave error mailbox return value: 0x02)

**EC\_E\_MBXERR\_INVALIDCHANNEL**

0x98110172: SLV: Mailbox error: Field contains wrong value (Slave error mailbox return value: 0x03)

**EC\_E\_MBXERR\_SERVICENOTSUPPORTED**

0x98110173: SLV: Mailbox error: The mailbox protocol header of the mailbox protocol is wrong (Slave error mailbox return value: 0x04)

**EC\_E\_MBXERR\_INVALIDHEADER**

0x98110174: SLV: Mailbox error: The mailbox protocol header of the mailbox protocol is wrong (Slave error mailbox return value: 0x05)

**EC\_E\_MBXERR\_SIZETOOSHORT**

0x98110175: SLV: Mailbox error: Length of received mailbox data is too short (Slave error mailbox return value: 0x06)

**EC\_E\_MBXERR\_NOMOREMEMORY**

0x98110176: SLV: Mailbox error: Mailbox protocol can not be processed because of limited resources (Slave error mailbox return value: 0x07)

**EC\_E\_MBXERR\_INVALIDSIZE**

0x98110177: SLV: Mailbox error: The length of data is inconsistent (Slave error mailbox return value: 0x08)

**EC\_E\_DC\_SLAVES\_BEFORE\_REF\_CLOCK**

0x98110178: ENI: Slaves with DC configured present on bus before reference clock (e.g. The first DC Slave was not configured as potential reference clock)

**EC\_E\_DATA\_TYPE\_CONVERSION\_FAILED**

0x98110179: Data type conversion failed

**EC\_E\_LINE\_CROSSED**

0x9811017B: Line crossed (cabling wrong)

**EC\_E\_LINE\_CROSSED\_SLAVE\_INFO**

0x9811017C: Line crossed at slave (obsolete)

**EC\_E\_ADO\_NOT\_SUPPORTED**

0x9811017E: SLV: ADO for slave identification not supported (e.g. Request ID mechanism (ADO 0x134) not supported by slave)

**EC\_E\_FRAMELOSS\_AFTER\_SLAVE**

0x9811017F: Frameloss after Slave (opening port destroys communication)

**EC\_E\_OEM\_SIGNATURE\_MISMATCH**

0x98130008: ENI, OEM: Manufacturer signature mismatch

**EC\_E\_ENI\_ENCRYPTION\_WRONG\_VERSION**

0x98130009: ENI, OEM: ENI encryption algorithm version not supported

**EC\_E\_ENI\_ENCRYPTED**

0x9813000A: OEM: Loading encrypted ENI needs OEM key

**EC\_E\_OEM\_KEY\_MISMATCH**

0x9813000B: RAS, APP: OEM key mismatch

**EC\_E\_OEM\_KEY\_MISSING**

0x9813000C: APP: OEM key access needs OEM key set (e.g. Application must call esSetOemKey (HiL) or set EC\_T\_LINK\_PARMS\_SIMULATOR::qwOemKey (SiL))

**EC\_E\_S2SMBX\_NOT\_CONFIGURED**

0x98130020: S2S: Not Configured

**EC\_E\_S2SMBX\_NO\_MEMORY**

0x98130021: S2S: No Memory

**EC\_E\_S2SMBX\_NO\_DESCRIPTOR**

0x98130022: S2S: No Descriptor

**EC\_E\_S2SMBX\_DEST\_SLAVE\_NOT\_FOUND**

0x98130023: S2S: Destination Slave not found

**EC\_E\_MASTER\_RED\_STATE\_INACTIVE**

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

**EC\_E\_MASTER\_RED\_STATE\_ACTIVE**

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

**EC\_E\_JUNCTION\_RED\_LINE\_BREAK**

0x98130026: Junction redundancy line break

**EC\_E\_VALIDATION\_ERROR**

0x98130027: Validation error (validation data mismatch)

**EC\_E\_TIMEOUT\_WAITING\_FOR\_DC**

0x98130028: Timeout waiting for DC

**EC\_E\_TIMEOUT\_WAITING\_FOR\_DCM**

0x98130029: Timeout waiting for DCM

**EC\_E\_SIGNATURE\_MISMATCH**

0x98130030: Signature mismatch

**EC\_E\_PDIWATCHDOG**

0x98130031: PDI watchdog expired

**EC\_E\_BAD\_CONNECTION**

0x98130032: Bad connection

**EC\_E\_XML\_INCONSISTENT**

0x98130033: ENI: Inconsistent content

## 8.3 DCM Error Codes

**DCM\_E\_ERROR**

0x981201C0: Unspecific DCM Error

**DCM\_E\_NOTINITIALIZED**

0x981201C1: Not initialized

**DCM\_E\_MAX\_CTL\_ERROR\_EXCEED**

0x981201C2: DCM controller - synchronization out of limit

**DCM\_E\_NOMEMORY**

0x981201C3: Not enough memory

**DCM\_E\_INVALID\_HWLAYER**

0x981201C4: Hardware layer - (BSP) invalid

**DCM\_E\_TIMER MODIFY ERROR**

0x981201C5: Hardware layer - error modifying timer

**DCM\_E\_TIMER NOT RUNNING**

0x981201C6: Hardware layer - timer not running

**DCM\_E\_WRONG\_CPU**

0x981201C7: Hardware layer - function called on wrong CPU

**DCM\_E\_INVALID\_SYNC\_PERIOD**

0x981201C8: Invalid DC sync period length (invalid clock master?)

**DCM\_E\_INVALID\_SETVAL**

0x981201C9: DCM controller SetVal to small

**DCM\_E\_DRIFT\_TO\_HIGH**

0x981201CA: DCM controller - Drift between local timer and ref clock to high

**DCM\_E\_BUS\_CYCLE\_WRONG**

0x981201CB: DCM controller - Bus cycle time (dwBusCycleTimeUsec) doesn't match real cycle

**DCX\_E\_NO\_EXT\_CLOCK**

0x981201CC: DCX controller - No external synchronization clock found

**DCM\_E\_INVALID\_DATA**

0x981201CD: DCM controller - Invalid data

## 8.4 ADS over EtherCAT (AoE) Error Codes

**EC\_E\_AOE\_NO\_RTIME**

0x9813000D: AoE: No Rtime

**EC\_E\_AOE\_LOCKED\_MEMORY**

0x9813000E: AoE: Allocation locked memory

**EC\_E\_AOE\_MAILBOX**

0x9813000F: AoE: Insert mailbox error

**EC\_E\_AOE\_WRONG\_HMSG**

0x98130010: AoE: Wrong receive HMSG

**EC\_E\_AOE\_BAD\_TASK\_ID**

0x98130011: AoE: Bad task ID

**EC\_E\_AOE\_NO\_IO**

0x98130012: AoE: No IO

**EC\_E\_AOE\_UNKNOWN\_AMS\_COMMAND**

0x98130013: AoE: Unknown ADS command

**EC\_E\_AOE\_WIN32**

0x98130014: AoE: Win 32 error

**EC\_E\_AOE\_LOW\_INSTALL\_LEVEL**

0x98130015: AoE: Low installation level

**EC\_E\_AOE\_NO\_DEBUG**

0x98130016: AoE: No debug available

**EC\_E\_AOE\_AMS\_SYNC\_WIN32**

0x98130017: AoE: Sync Win 32 error

**EC\_E\_AOE\_AMS\_SYNC\_TIMEOUT**

0x98130018: AoE: Sync Timeout

**EC\_E\_AOE\_AMS\_SYNC\_AMS**

0x98130019: AoE: Sync AMS error

**EC\_E\_AOE\_AMS\_SYNC\_NO\_INDEX\_MAP**

0x9813001A: AoE: Sync no index map

**EC\_E\_AOE\_TCP\_SEND**

0x9813001B: AoE: TCP send error

**EC\_E\_AOE\_HOST\_UNREACHABLE**

0x9813001C: AoE: Host unreachable

**EC\_E\_AOE\_INVALIDAMSFRAGMENT**

0x9813001D: AoE: Invalid AMS fragment

**EC\_E\_AOE\_NO\_LOCKED\_MEMORY**

0x9813001E: AoE: No allocation locked memory

**EC\_E\_AOE\_MAILBOX\_FULL**

0x9813001F: AoE: Mailbox full

## 8.5 CAN application protocol over EtherCAT (CoE) SDO Error Codes

### **EC\_E\_SDO\_ABORTCODE\_TOGGLE**

0x98110040: SLV: SDO: Toggle bit not alternated (CoE abort code 0x05030000 of slave)

### **EC\_E\_SDO\_ABORTCODE\_TIMEOUT**

0x98110041: SLV: SDO: Protocol timed out (CoE abort code 0x05040000 of slave)

### **EC\_E\_SDO\_ABORTCODE\_CCS\_SCS**

0x98110042: SLV: SDO: Client/server command specifier not valid or unknown (CoE abort code 0x05040001 of slave)

### **EC\_E\_SDO\_ABORTCODE\_BLK\_SIZE**

0x98110043: SLV: SDO: Invalid block size (block mode only) (CoE abort code 0x05040002 of slave)

### **EC\_E\_SDO\_ABORTCODE\_SEQNO**

0x98110044: SLV: SDO: Invalid sequence number (block mode only) (CoE abort code 0x05040003 of slave)

### **EC\_E\_SDO\_ABORTCODE\_CRC**

0x98110045: SLV: SDO: CRC error (block mode only) (CoE abort code 0x05040004 of slave)

### **EC\_E\_SDO\_ABORTCODE\_MEMORY**

0x98110046: SLV: SDO: Out of memory (CoE abort code 0x05040005 of slave)

### **EC\_E\_SDO\_ABORTCODE\_ACCESS**

0x98110047: SLV: SDO: Unsupported access to an object (CoE abort code 0x06010000 of slave)

### **EC\_E\_SDO\_ABORTCODE\_WRITEONLY**

0x98110048: SLV: SDO: Attempt to read a write only object (CoE abort code 0x06010001 of slave)

### **EC\_E\_SDO\_ABORTCODE\_READONLY**

0x98110049: SLV: SDO: Attempt to write a read only object (CoE abort code 0x06010002 of slave)

### **EC\_E\_SDO\_ABORTCODE\_INDEX**

0x9811004A: SLV: SDO: Object does not exist in the object dictionary (CoE abort code 0x06020000 of slave)

### **EC\_E\_SDO\_ABORTCODE\_PDO\_MAP**

0x9811004B: SLV: SDO: Object cannot be mapped to the PDO (CoE abort code 0x06040041 of slave)

### **EC\_E\_SDO\_ABORTCODE\_PDO\_LEN**

0x9811004C: SLV: SDO: The number and length of the objects to be mapped would exceed PDO length (CoE abort code 0x06040042 of slave)

### **EC\_E\_SDO\_ABORTCODE\_P\_INCOMP**

0x9811004D: SLV: SDO: General parameter incompatibility reason (CoE abort code 0x06040043 of slave)

### **EC\_E\_SDO\_ABORTCODE\_I\_INCOMP**

0x9811004E: SLV: SDO: General internal incompatibility in the device (CoE abort code 0x06040047 of slave)

### **EC\_E\_SDO\_ABORTCODE\_HARDWARE**

0x9811004F: SLV: SDO: Access failed due to an hardware error (CoE abort code 0x06060000 of slave)

**EC\_E\_SDO\_ABORTCODE\_DATA\_LENGTH\_NOT\_MATCH**

0x98110050: SLV: SDO: Data type does not match, length of service parameter does not match (CoE abort code 0x06070010 of slave)

**EC\_E\_SDO\_ABORTCODE\_DATA\_LENGTH\_TOO\_HIGH**

0x98110051: SLV: SDO: Data type does not match, length of service parameter too high (CoE abort code 0x06070012 of slave)

**EC\_E\_SDO\_ABORTCODE\_DATA\_LENGTH\_TOO\_LOW**

0x98110052: SLV: SDO: Data type does not match, length of service parameter too low (CoE abort code 0x06070013 of slave)

**EC\_E\_SDO\_ABORTCODE\_OFFSET**

0x98110053: SLV: SDO: Sub-index does not exist (CoE abort code 0x06090011 of slave)

**EC\_E\_SDO\_ABORTCODE\_VALUE\_RANGE**

0x98110054: SLV: SDO: Value range of parameter exceeded (only for write access) (CoE abort code 0x06090030 of slave)

**EC\_E\_SDO\_ABORTCODE\_VALUE\_TOO\_HIGH**

0x98110055: SLV: SDO: Value of parameter written too high (CoE abort code 0x06090031 of slave)

**EC\_E\_SDO\_ABORTCODE\_VALUE\_TOO\_LOW**

0x98110056: SLV: SDO: Value of parameter written too low (CoE abort code 0x06090032 of slave)

**EC\_E\_SDO\_ABORTCODE\_MINMAX**

0x98110057: SLV: SDO: Maximum value is less than minimum value (CoE abort code 0x06090036 of slave)

**EC\_E\_SDO\_ABORTCODE\_GENERAL**

0x98110058: SLV: SDO: General error (CoE abort code 0x08000000 of slave)

**EC\_E\_SDO\_ABORTCODE\_TRANSFER**

0x98110059: SLV: SDO: Data cannot be transferred or stored to the application (CoE abort code 0x08000020 of slave)

**EC\_E\_SDO\_ABORTCODE\_TRANSFER\_LOCAL\_CONTROL**

0x9811005A: SLV: SDO: Data cannot be transferred or stored to the application because of local control (CoE abort code 0x08000021 of slave)

**EC\_E\_SDO\_ABORTCODE\_TRANSFER\_DEVICE\_STATE**

0x9811005B: SLV: SDO: Data cannot be transferred or stored to the application because of the present device state (CoE abort code 0x08000022 of slave)

**EC\_E\_SDO\_ABORTCODE\_DICTIONARY**

0x9811005C: SLV: SDO: Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error) (CoE abort code 0x08000023 of slave)

**EC\_E\_SDO\_ABORTCODE\_UNKNOWN**

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

**EC\_E\_SDO\_ABORTCODE\_MODULE\_ID\_LIST\_NOT\_MATCH**

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

**EC\_E\_SDO\_ABORTCODE\_SI\_NOT\_WRITTEN**

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

**EC\_E\_SDO\_ABORTCODE\_CA\_TYPE\_MISM**

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

**EC\_E\_SDO\_ABORTCODE\_OBJ\_TOO\_BIG**

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

**EC\_E\_SDO\_ABORTCODE\_PDO\_MAPPED**

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

## 8.6 File Transfer over EtherCAT (FoE) Error Codes

### **EC\_E\_FOE\_ERRCODE\_NOTDEFINED**

0x98110060: SLV: ERROR FoE: not defined (FoE Error Code 0 (0x8000) of slave)

### **EC\_E\_FOE\_ERRCODE\_NOTFOUND**

0x98110061: SLV: ERROR FoE: not found (FoE Error Code 1 (0x8001) of slave)

### **EC\_E\_FOE\_ERRCODE\_ACCESS**

0x98110062: SLV: ERROR FoE: access denied (FoE Error Code 2 (0x8002) of slave)

### **EC\_E\_FOE\_ERRCODE\_DISKFULL**

0x98110063: SLV: ERROR FoE: disk full (FoE Error Code 3 (0x8003) of slave)

### **EC\_E\_FOE\_ERRCODE\_ILLEGAL**

0x98110064: SLV: ERROR FoE: illegal (FoE Error Code 4 (0x8004) of slave)

### **EC\_E\_FOE\_ERRCODE\_PACKENO**

0x98110065: SLV: ERROR FoE: packet number wrong (FoE Error Code 5 (0x8005) of slave)

### **EC\_E\_FOE\_ERRCODE\_EXISTS**

0x98110066: SLV: ERROR FoE: already exists (FoE Error Code 6 (0x8006) of slave)

### **EC\_E\_FOE\_ERRCODE\_NOUSER**

0x98110067: SLV: ERROR FoE: no user (FoE Error Code 7 (0x8007) of slave)

### **EC\_E\_FOE\_ERRCODE\_BOOTSTRAPONLY**

0x98110068: SLV: ERROR FoE: bootstrap only (FoE Error Code 8 (0x8008) of slave)

### **EC\_E\_FOE\_ERRCODE\_NOTINBOOTSTRAP**

0x98110069: SLV: ERROR FoE: Downloaded file name is not valid in Bootstrap state (FoE Error Code 9 (0x8009) of slave)

### **EC\_E\_FOE\_ERRCODE\_INVALIDPASSWORD**

0x9811006A: SLV: ERROR FoE: no rights (FoE Error Code 10 (0x800A) of slave)

### **EC\_E\_FOE\_ERRCODE\_PROGERROR**

0x9811006B: SLV: ERROR FoE: program error (FoE Error Code 11 (0x800B) of slave)

### **EC\_E\_FOE\_ERRCODE\_INVALID\_CHECKSUM**

0x9811006C: FoE: Wrong checksum

### **EC\_E\_FOE\_ERRCODE\_INVALID\_FIRMWARE**

0x9811006D: SLV: ERROR FoE: Firmware does not fit for Hardware (FoE Error Code 13 (0x800D) of slave)

### **EC\_E\_FOE\_ERRCODE\_NO\_FILE**

0x9811006F: SLV: ERROR FoE: No file to read (FoE Error Code 15 (0x800F) of slave)

### **EC\_E\_NO\_FOE\_SUPPORT\_BS**

0x9811010F: APP: ERROR FoE: Protocol not supported in boot strap (e.g. Application requested FoE in Bootstrap although slave does not support this)

### **EC\_E\_FOE\_ERRCODE\_MAX\_FILE\_SIZE**

0x9811017A: APP: ERROR FoE: File is bigger than max file size (e.g. Slave returned more data than the

buffer provided by application can store.)

**EC\_E\_FOE\_ERRCODE\_FILE\_HEAD\_MISSING**

0x98130001: SLV: ERROR FoE: File header does not exist (FoE Error Code 16 (0x8010) of slave)

**EC\_E\_FOE\_ERRCODE\_FLASH\_PROBLEM**

0x98130002: SLV: ERROR FoE: Flash problem (FoE Error Code 17 (0x8011) of slave)

**EC\_E\_FOE\_ERRCODE\_FILE\_INCOMPATIBLE**

0x98130003: SLV: ERROR FoE: File incompatible (FoE Error Code 18 (0x8012) of slave)

## 8.7 Servo Drive Profil over EtherCAT (SoE) Error Codes

**EC\_E\_SOE\_ERRORCODE\_INVALID\_ACCESS**

0x98110078: ERROR SoE: Invalid access to element 0

**EC\_E\_SOE\_ERRORCODE\_NOT\_EXIST**

0x98110079: ERROR SoE: Does not exist

**EC\_E\_SOE\_ERRORCODE\_INVL\_ACC\_ELEM1**

0x9811007A: ERROR SoE: Invalid access to element 1

**EC\_E\_SOE\_ERRORCODE\_NAME\_NOT\_EXIST**

0x9811007B: ERROR SoE: Name does not exist

**EC\_E\_SOE\_ERRORCODE\_NAME\_UNDERSIZE**

0x9811007C: ERROR SoE: Name undersize in transmission

**EC\_E\_SOE\_ERRORCODE\_NAME\_OVERSIZE**

0x9811007D: ERROR SoE: Name oversize in transmission

**EC\_E\_SOE\_ERRORCODE\_NAME\_UNCHANGE**

0x9811007E: ERROR SoE: Name unchangeable

**EC\_E\_SOE\_ERRORCODE\_NAME\_WR\_PROT**

0x9811007F: ERROR SoE: Name currently write-protected

**EC\_E\_SOE\_ERRORCODE\_UNDERS\_TRANS**

0x98110080: ERROR SoE: Attribute undersize in transmission

**EC\_E\_SOE\_ERRORCODE\_OVERS\_TRANS**

0x98110081: ERROR SoE: Attribute oversize in transmission

**EC\_E\_SOE\_ERRORCODE\_ATTR\_UNCHANGE**

0x98110082: ERROR SoE: Attribute unchangeable

**EC\_E\_SOE\_ERRORCODE\_ATTR\_WR\_PROT**

0x98110083: ERROR SoE: Attribute currently write-protected

**EC\_E\_SOE\_ERRORCODE\_UNIT\_NOT\_EXIST**

0x98110084: ERROR SoE: Unit does not exist

**EC\_E\_SOE\_ERRORCODE\_UNIT\_UNDERSIZE**

0x98110085: ERROR SoE: Unit undersize in transmission

**EC\_E\_SOE\_ERRORCODE\_UNIT\_OVERSIZE**

0x98110086: ERROR SoE: Unit oversize in transmission

**EC\_E\_SOE\_ERRORCODE\_UNIT\_UNCHANGE**

0x98110087: ERROR SoE: Unit unchangeable

**EC\_E\_SOE\_ERRORCODE\_UNIT\_WR\_PROT**

0x98110088: ERROR SoE: Unit currently write-protected

**EC\_E\_SOE\_ERRORCODE\_MIN\_NOT\_EXIST**

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

**EC\_E\_SOE\_ERRORCODE\_MIN\_UNDERSIZE**

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

**EC\_E\_SOE\_ERRORCODE\_MIN\_OVERSIZE**

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

**EC\_E\_SOE\_ERRORCODE\_MIN\_UNCHANGE**

0x9811008C: ERROR SoE: Minimum input value unchangeable

**EC\_E\_SOE\_ERRORCODE\_MIN\_WR\_PROT**

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

**EC\_E\_SOE\_ERRORCODE\_MAX\_NOT\_EXIST**

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

**EC\_E\_SOE\_ERRORCODE\_MAX\_UNDERSIZE**

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

**EC\_E\_SOE\_ERRORCODE\_MAX\_OVERSIZE**

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

**EC\_E\_SOE\_ERRORCODE\_MAX\_UNCHANGE**

0x98110091: ERROR SoE: Maximum input value unchangeable

**EC\_E\_SOE\_ERRORCODE\_MAX\_WR\_PROT**

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

**EC\_E\_SOE\_ERRORCODE\_DATA\_NOT\_EXIST**

0x98110093: ERROR SoE: Data item does not exist

**EC\_E\_SOE\_ERRORCODE\_DATA\_UNDERSIZE**

0x98110094: ERROR SoE: Data item undersize in transmission

**EC\_E\_SOE\_ERRORCODE\_DATA\_OVERSIZE**

0x98110095: ERROR SoE: Data item oversize in transmission

**EC\_E\_SOE\_ERRORCODE\_DATA\_UNCHANGE**

0x98110096: ERROR SoE: Data item unchangeable

**EC\_E\_SOE\_ERRORCODE\_DATA\_WR\_PROT**

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

**EC\_E\_SOE\_ERRORCODE\_DATA\_MIN\_LIMIT**

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

**EC\_E\_SOE\_ERRORCODE\_DATA\_MAX\_LIMIT**

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

**EC\_E\_SOE\_ERRORCODE\_DATA\_INCOR**

0x9811009A: ERROR SoE: Data item incorrect

**EC\_E\_SOE\_ERRORCODE\_PASWD\_PROT**

0x9811009B: ERROR SoE: Data item protected by password

**EC\_E\_SOE\_ERRORCODE\_TEMP\_UNCHANGE**

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

**EC\_E\_SOE\_ERRORCODE\_INVL\_INDIRECT**

0x9811009D: ERROR SoE: Invalid indirect

**EC\_E\_SOE\_ERRORCODE\_TEMP\_UNCHANGE1**

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

**EC\_E\_SOE\_ERRORCODE\_ALREADY\_ACTIVE**

0x9811009F: ERROR SoE: Command already active

**EC\_E\_SOE\_ERRORCODE\_NOT\_INTERRUPT**

0x98110100: ERROR SoE: Command not interruptible

**EC\_E\_SOE\_ERRORCODE\_CMD\_NOT\_AVAIL**

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

**EC\_E\_SOE\_ERRORCODE\_CMD\_NOT\_AVAIL1**

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

**EC\_E\_SOE\_ERRORCODE\_DRIVE\_NO**

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

**EC\_E\_SOE\_ERRORCODE\_IDN**

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

**EC\_E\_SOE\_ERRORCODE\_FRAGMENT\_LOST**

0x98110105: ERROR SoE: At least one fragment lost

**EC\_E\_SOE\_ERRORCODE\_BUFFER\_FULL**

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

**EC\_E\_SOE\_ERRORCODE\_NO\_DATA**

0x98110107: ERROR SoE: No data state

**EC\_E\_SOE\_ERRORCODE\_NO\_DEFAULT\_VALUE**

0x98110108: ERROR SoE: No default value

**EC\_E\_SOE\_ERRORCODE\_DEFAULT\_LONG**

0x98110109: ERROR SoE: Default value transmission too long

**EC\_E\_SOE\_ERRORCODE\_DEFAULT\_WP**

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

**EC\_E\_SOE\_ERRORCODE\_INVL\_DRIVE\_NO**

0x9811010B: ERROR SoE: Invalid drive number

**EC\_E\_SOE\_ERRORCODE\_GENERAL\_ERROR**

0x9811010C: ERROR SoE: General error

**EC\_E\_SOE\_ERRCODE\_NO\_ELEM\_ADR**

0x9811010D: ERROR SoE: No element addressed

## 8.8 Remote API Error Codes

**EC\_E\_SOCKET\_DISCONNECTED**

0x9811017D: RAS: Socket disconnected (e.g. IP connection terminated or lost)

**EMRAS\_E\_INVALIDCOOKIE**

0x98110181: RAS: Invalid Cookie (e.g. obsolete)

**EMRAS\_E\_MULSRVDISMULCON**

0x98110183: RAS: Connect 2nd server denied because Multi Server support is disabled (obsolete)

**EMRAS\_E\_LOGONCANCELLED**

0x98110184: RAS: Logon canceled (Server-side connection reject while opening a client connection.)

**EMRAS\_E\_INVALIDVERSION**

0x98110186: RAS: Invalid Version (Connection reject because of using mismatching protocol versions on client and server side)

**EMRAS\_E\_INVALIDACCESSCONFIG**

0x98110187: RAS: Access configuration is invalid (e.g. SPoC access configuration invalid)

**EMRAS\_E\_ACCESSLESS**

0x98110188: RAS: No access to this call at this access level (e.g. a higher SPoC access level is needed to use the called Remote API function)

**EMRAS\_E\_INVALIDDATA RECEIVED**

0x98110189: RAS: Invalid data received (communication corrupted)

**EMRAS\_EVT\_SERVERSTOPPED**

0x98110191: RAS: Server stopped (e.g. connection dropped because of Remote API Server stop)

**EMRAS\_EVT\_WDEXPIRED**

0x98110192: RAS: Watchdog expired (e.g. connection dropped because of missing keep-alive messages)

**EMRAS\_EVT\_RECONEXPIRED**

0x98110193: RAS: Reconnect expired (obsolete)

**EMRAS\_EVT\_CLIENTLOGON**

0x98110194: RAS Server: Client logged on

**EMRAS\_EVT\_RECONNECT**

0x98110195: RAS: obsolete

**EMRAS\_EVT\_SOCKCHANGE**

0x98110196: RAS: Socket exchanged after reconnect (obsolete)

**EMRAS\_EVT\_CLNTDISC**

0x98110197: RAS: Client disconnect

**EMRAS\_E\_ACCESS\_NOT\_FOUND**

0x98110198: RAS: Access not configured for this call (e.g. SPoC access configuration missing)

**EMRAS\_E\_TOKEN\_MISSING**

0x98110199: RAS: Token missing

**EMRAS\_E\_TOKEN\_INVALID**  
0x9811019A: RAS: Token invalid

**EMRAS\_E\_TOKEN\_DENIED**  
0x9811019B: RAS: Token denied