



acontis technologies GmbH

SOFTWARE

EC-Master

EtherCAT® Master Stack Class B

V3.3

AT3302

Edition: April 26, 2026

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

© Copyright **acontis technologies GmbH**

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

Contents

1	Introduction	13
1.1	What is EtherCAT®?	13
1.2	The EC-Master - Features	13
1.3	Protected version	17
1.3.1	Licensing procedure for Development Licenses	17
1.3.2	Licensing procedure for Runtime Licenses	17
1.4	Dongled version	18
1.4.1	Licensing using a virtual WibuCmRaU File on Linux	18
1.4.2	Maintain the license after a linux image update	19
1.5	License	19
1.5.1	EC-Master license	19
1.5.2	Free Open Source Software contained in EC-Master	19
1.5.3	Free Open Source Software supported by EC-Master	20
1.6	Versioning	20
2	Getting Started	21
2.1	EC-Master Architecture	21
2.2	EtherCAT® Network Configuration (ENI)	21
2.3	Operating system configuration	22
2.4	Running EcMasterDemo	22
2.4.1	Command line parameters	22
2.5	Compiling the EcMasterDemo	34
2.5.1	EtherCAT® MainDevice Software Development Kit (SDK)	35
2.5.2	Include search path	35
2.5.3	Libraries	35
3	Software Integration	36
3.1	Timing	36
3.1.1	JobTask: Interaction between application and EtherCAT® network	36
3.1.2	Timing models	37
3.1.3	Standard EtherCAT® Timing: Short output dead time	38
3.1.4	Alternative EtherCAT® Timing: Short Input dead time	40
3.2	Example application	42
3.2.1	File reference	43
3.2.2	EC-Master lifecycle	45
3.2.3	Synchronization	48
3.2.4	Event notification	49
3.2.5	Logging	51
3.3	EC-Master startup	51
3.3.1	Asynchronous (deferred) startup	52
3.3.2	Synchronous startup	53
3.4	EtherCAT® Network Configuration ENI	55
3.4.1	Single cyclic entry configuration	55
3.4.2	Multiple cyclic entries configuration	57
3.4.3	Copy Information for SubDevice-to-SubDevice communication	58
3.4.4	Swap bytes of variables according to ENI	59
3.5	Process Data Access	61
3.5.1	Process data access using hard coded offsets	63
3.5.2	Process data access using variable names from ENI	64
3.5.3	Process data access using variable object index from ENI	65
3.5.4	Process data access using SubDevice station address	67
3.5.5	Process data access using generated PD layout C-header file	68
3.5.6	Process Data Access Functions	71
3.5.7	Process and application variable linking	71

3.6	Process Data Memory	73
3.6.1	EC-Master as process data memory provider	73
3.6.2	Application as process data memory provider with fixed buffers	75
3.6.3	Application as process data memory provider with dynamic buffers	77
3.7	Error detection and diagnosis	79
3.7.1	Cyclic cmd WKC validation and Frame Loss	79
3.7.2	Working Counter (WKC) State in Diagnosis Image	79
3.7.3	MainDevice Sync Units (MSU)	80
3.8	EtherCAT® traffic logging in application	81
3.9	Trace Data	81
3.9.1	Trace Data configuration via EC-Engineer	82
3.9.2	Trace Data configuration via API	84
3.10	EC-Master MainDevice Stack Source Code	85
3.10.1	Components	85
3.10.2	Excluding features	86
3.11	Reduced Feature Set	90
3.11.1	Rfs1: Convenience functionality excluded	91
3.11.2	Rfs2: Rare functionalities excluded	91
3.11.3	Rfs3: Common functionalities excluded	92
3.11.4	Rfs4: Error detection and diagnosis excluded	93
3.11.5	Rfs5: Only CoE SubDevices supported	93
4	Platform and Operating Systems (OS)	94
4.1	CMSIS-RTOS for STM32	94
4.1.1	Setting up and running EcMasterDemo in Keil µVision IDE	94
4.1.2	OS Compiler settings	94
4.1.3	Setting up and running EcMasterDemo in STM32CubeIDE for STM32H747I-DISCO	95
4.2	eCos	96
4.2.1	Setting up and running EcMasterDemo	96
4.2.2	OS Compiler settings	98
4.3	FreeRTOS	98
4.3.1	Setting up and running EcMasterDemo on Xilinx Zynq UltraScale+ (ZCU104) and Xilinx Zynq-7000 (ZC702 Evaluation Kit)	98
4.3.2	Setting up and running EcMasterDemo on TI AM64x EVM for R5 Core	100
4.3.3	Setting up and running EcMasterDemo on TI AM243x LP and TI AM243x EVM	100
4.3.4	Setting up and running EcMasterDemo on TI J784s4 EVM for R5 Core	101
4.4	tenAsys INtime	102
4.4.1	Setting up and running EcMasterDemo	102
4.4.2	OS Compiler settings	102
4.5	Linux	103
4.5.1	OS optimizations	103
4.5.2	atemsys kernel module	104
4.5.3	Unbind Ethernet Driver instance	105
4.5.4	Docker	106
4.5.5	Setting up and running EcMasterDemo	107
4.5.6	OS Compiler settings	108
4.5.7	Build using cmake on Linux	109
4.5.8	Cross-platform development under Windows	109
4.6	PC / BIOS	110
4.7	QNX Neutrino	110
4.7.1	Thread priority	110
4.7.2	Unbind Ethernet Driver instance	110
4.7.3	IOMMU/SMMU support	111
4.7.4	Setting up and running EcMasterDemo	111
4.7.5	OS Compiler settings	111
4.8	IntervalZero RTX	112
4.8.1	Unbind Ethernet Driver instance	112
4.8.2	Setting up and running EcMasterDemo	112

4.8.3	OS Compiler settings	112
4.9	SylixOS	113
4.9.1	Setting up and running EcMasterDemo on SylixOS	113
4.10	TI-RTOS	113
4.10.1	Setting up and running EcMasterDemo	113
4.10.2	OS Compiler settings	114
4.11	µC3 for STM32	115
4.11.1	Setting up and running EcMasterDemo in IAR for ARM IDE	115
4.11.2	OS Compiler settings	116
4.12	µC3 for i.MX8	116
4.12.1	Setting up and running EcMasterDemo on NXP 8MPLUSLPD4-EVK board	116
4.13	Windriver VxWorks	118
4.13.1	VxWorks native	118
4.13.2	SNARF Ethernet Driver	119
4.13.3	Setting up and running EcMasterDemo	119
4.13.4	OS Compiler settings	121
4.14	Microsoft Windows	122
4.14.1	EcMasterDemo	122
4.14.2	EcMasterDemoDotNet (.NET) - Microsoft Windows	123
4.14.3	EcMasterDemoGuiDotNet (.NET) - Microsoft Windows	124
4.14.4	OS Compiler settings	124
4.14.5	RtaccDevice for Real-time Ethernet Driver	124
4.15	Microsoft Windows CE	137
4.15.1	Identification of the Real-time Ethernet Driver	138
4.15.2	KUKA CeWin	139
4.15.3	Windows CE 5.0	140
4.15.4	Windows CE 6.0	140
4.15.5	Windows CE 2013	140
4.15.6	Setting up and running EcMasterDemo	141
4.15.7	OS Compiler settings	142
4.16	Xenomai	143
4.16.1	Setting up and running EcMasterDemo	144
4.16.2	OS compiler settings	144
4.17	Zephyr	145
4.17.1	Setting up and running EcMasterDemo	145
4.17.2	OS Compiler settings	146
5	Real-time Ethernet Driver	147
5.1	Real-time Ethernet Driver initialization	148
5.1.1	Real-time Ethernet Driver instance selection via PCI location	150
5.2	Intel Pro/1000 - emllIntelGbe	151
5.2.1	TTS Feature	152
5.2.2	Supported PCI devices	152
5.3	Intel Pro/100 - emllI8255x	154
5.3.1	Supported PCI devices	154
5.4	Broadcom Genet - emllBcmGenet	154
5.5	Broadcom NetXtreme - emllBcmNetXtreme	155
5.5.1	Supported PCI devices	155
5.6	Berkeley Packet Filter - emllBPF	156
5.7	Beckhoff CCAT - emllCCAT	156
5.7.1	Supported PCI devices	157
5.8	CMSIS - emllCmsisEth	157
5.9	Texas Instruments CPSW - emllCPSW	158
5.9.1	CPSW usage under Linux	159
5.10	Texas Instruments CPSWG for AM6x and Jacinto 7 - emllCPSWG	159
5.10.1	CPSWG usage under Linux	160
5.11	Linux DPDK - emllDpdk	161
5.11.1	Linux System Requirements	162

5.11.2	Huge pages setup	162
5.11.3	DPDK for PCI Network Adapter	162
5.11.4	DPDK for DPAA	163
5.11.5	DPDK for ENETC4	165
5.11.6	Limitations	166
5.12	DW3504 - emllDW3504	167
5.12.1	Supported PCI devices	169
5.13	Freescale TSEC / eTSEC - emllETSEC	169
5.13.1	ETSEC supported MAC's	171
5.13.2	Shared MII bus	171
5.13.3	Locking	171
5.13.4	Link check	172
5.13.5	Fixed Link	172
5.14	Freescale FslFec - emllFslFec	172
5.15	Cadence GEM/MACB - emllGEM	174
5.16	INtime HPE - emllHPE	175
5.17	Texas Instruments ICSS - emllICSS	176
5.17.1	TTS Feature	177
5.17.2	TI AM335x ICEV2	177
5.17.3	TI AM57xx IDK	178
5.17.4	AM5728 IDK and AM5718 IDK boards and Technical Limitations	178
5.18	Texas Instruments ICSSG - emllICSSG on AM654x	178
5.18.1	TI AM654x IDK	179
5.19	Microchip LAN743x - emlllan743x	179
5.19.1	Supported PCI devices	179
5.20	Beckhoff CUxxxx Multiplier - emllMultiplier	179
5.20.1	Configuration with EC-Engineer	180
5.21	Windows NDIS - emllNdis	182
5.22	Windows WinPcap - emllPcap	183
5.22.1	WinPcap, Npcap support	183
5.23	RDC R6040 - emllR6040	184
5.23.1	Supported PCI devices	185
5.24	emllRemote	185
5.25	Realtek RTL8169 - emllRTL8169	186
5.25.1	RTL8169 usage under Linux	186
5.25.2	Supported PCI devices	186
5.26	Renesas RZ/T1 - emllRZT1	187
5.27	Renesas SHEth - emllSHEth	187
5.27.1	SHEth link status update	189
5.27.2	SHEth usage under Linux	189
5.28	VxWorks SNARF - emllSNARF	189
5.29	Linux SockRaw - emllSockRaw	190
5.30	Linux SockXdp - emllSockXdp	191
5.30.1	Linux System Requirements	192
5.30.2	Getting started	192
5.31	AMD Tri-Mode Ethernet Media Access Controller (TEMAC)	192
5.31.1	Encustra Mercury_XU5_PE1	193
5.32	Texas Instruments CPSWG for AM6x and Jacinto 7 based on Enet LLD - emllTiEnetCpswg	193
5.32.1	TI J784S4X EVM	193
5.33	Texas Instruments ICSSG for AM6x and Jacinto 7 based on Enet LLD - emllTiEnetIcssg	193
5.33.1	TI AM64x EVM	194
5.33.2	TI AM243x LP and TI AM243x EVM	194
5.34	Virtual Local Area Network - emllVlan	194
5.34.1	Switch Configuration	195
6	Application programming interface, reference	196
6.1	Generic API return status values	196
6.2	Multiple EtherCAT® Bus Support	196

6.2.1	Licensing	196
6.2.2	Overview	197
6.2.3	Example application	197
6.3	General functions	198
6.3.1	emInitMaster	198
6.3.2	emDeinitMaster	203
6.3.3	emGetMasterParms	203
6.3.4	emSetMasterParms	204
6.3.5	emScanBus	205
6.3.6	emRescueScan	206
6.3.7	emConfigureNetwork	206
6.3.8	emConfigGet	210
6.3.9	emConfigExtend	211
6.3.10	emRegisterClient	212
6.3.11	emUnregisterClient	213
6.3.12	emGetSrcMacAddress	214
6.3.13	emSetMasterState	214
6.3.14	emSetMasterStateReq	216
6.3.15	emGetMasterState	217
6.3.16	emGetMasterStateEx	217
6.3.17	emExecJob	218
6.3.18	emGetVersion	223
6.3.19	emSetLicenseKey	224
6.3.20	emSetOemKey	225
6.3.21	emIoCtl	225
6.3.22	emIoCtl - EC_IOCTL_GET_PDMEMORYSIZE	226
6.3.23	emIoCtl - EC_IOCTL_REGISTER_PDMEMORYPROVIDER	227
6.3.24	emIoCtl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB	229
6.3.25	emIoCtl - EC_IOCTL_ISLINK_CONNECTED	230
6.3.26	emIoCtl - EC_IOCTL_GET_LINKLAYER_MODE	231
6.3.27	emIoCtl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO	231
6.3.28	emIoCtl - EC_IOCTL_IS_SLAVETOSLAVE_COMM_CONFIGURED	232
6.3.29	emIoCtl - EC_LINKIOCTL...	233
6.3.30	emIoCtl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS	233
6.3.31	emIoCtl - EC_LINKIOCTL_GET_SPEED	233
6.3.32	emIoCtl - EC_LINKIOCTL_GET_PCI_INFO	234
6.3.33	emIoCtl - EC_IOCTL_SET_CYCFRAME_LAYOUT	236
6.3.34	emIoCtl - EC_IOCTL_SET_MASTER_DEFAULT_TIMEOUTS	237
6.3.35	emIoCtl - EC_IOCTL_SET_COPYINFO_IN_SENDCYCFRAMES	238
6.3.36	emIoCtl - EC_IOCTL_SET_BUS_CYCLE_TIME	238
6.3.37	emIoCtl - EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES	239
6.3.38	emIoCtl - EC_IOCTL_SLV_ALIAS_ENABLE	239
6.3.39	emIoCtl - EC_IOCTL_SET_IGNORE_INPUTS_ON_WKC_ERROR	240
6.3.40	emIoCtl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ERROR	240
6.3.41	emIoCtl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ZERO	241
6.3.42	emIoCtl - EC_IOCTL_SET_ZERO_INPUTS_ON_FRAME_LOSS	241
6.3.43	emIoCtl - EC_IOCTL_SET_GENENI_ASSIGN_EEPROM_BACK_TO_ECAT	242
6.3.44	emIoCtl - EC_IOCTL_SET_EOE_DEFFERED_SWITCHING_ENABLED	242
6.3.45	emIoCtl - EC_IOCTL_SET_MAILBOX_POLLING_CYCLES	243
6.3.46	emIoCtl - EC_IOCTL_SET_MASTER_MAX_STATE	243
6.3.47	emIoCtl - EC_IOCTL_SET_AUTO_ADJUST_MBX_STATE_COUNT_ENABLED	244
6.3.48	emIoCtl - EC_IOCTL_ACTIVATE_VOE_RECV_FIFO	244
6.3.49	emIoCtl - EC_IOCTL_SET_GEN_ENI_PARM	245
6.3.50	emIoCtl - EC_IOCTL_REALLOC_MBX_QUEUE	246
6.4	Process Data Access	246
6.4.1	emGetProcessData	246
6.4.2	emGetProcessDataBits	248
6.4.3	emSetProcessData	248

6.4.4	emSetProcessDataBits	249
6.4.5	emForceProcessDataBits	250
6.4.6	emReleaseProcessDataBits	251
6.4.7	emReleaseAllProcessDataBits	252
6.4.8	emGetProcessImageInputPtr	253
6.4.9	emGetProcessImageOutputPtr	253
6.4.10	emGetDiagnosisImagePtr	254
6.4.11	emGetDiagnosisImageSize	254
6.4.12	emGetProcessVarInfoNumOf, emGetProcessVarInfoEx	254
6.4.13	emGetSlaveInpVarInfoNumOf	257
6.4.14	emGetSlaveInpVarInfo	258
6.4.15	emGetSlaveInpVarInfoEx	259
6.4.16	emGetSlaveOutpVarInfoNumOf	261
6.4.17	emGetSlaveOutpVarInfo	262
6.4.18	emGetSlaveOutpVarInfoEx	263
6.4.19	emGetSlaveInpVarByObjectEx	264
6.4.20	emGetSlaveOutpVarByObjectEx	265
6.4.21	emFindInpVarByName	266
6.4.22	emFindInpVarByNameEx	266
6.4.23	emFindOutpVarByName	267
6.4.24	emFindOutpVarByNameEx	268
6.4.25	emLinkInputVarByName	268
6.4.26	emLinkInputVarByObject	269
6.4.27	emLinkOutputVarByName	270
6.4.28	emLinkOutputVarByObject	271
6.4.29	emTraceDataConfig	271
6.4.30	emTraceDataGetInfo	272
6.4.31	EC_GET_FRM_WORD	273
6.4.32	EC_GET_FRM_DWORD	273
6.4.33	EC_GET_FRM_QWORD	274
6.4.34	EC_SET_FRM_WORD	274
6.4.35	EC_SET_FRM_DWORD	274
6.4.36	EC_SET_FRM_QWORD	275
6.4.37	EC_COPYBITS	275
6.4.38	EC_COMPAREBITS	276
6.4.39	EC_GETBITS	277
6.4.40	EC_SETBITS	277
6.4.41	EC_COPYBIT	278
6.4.42	EC_TESTBIT	278
6.4.43	EC_SETBIT	278
6.4.44	EC_CLRBIT	278
6.5	Generic notification interface	279
6.5.1	Notification callback: emNotify	279
6.5.2	emNotify - EC_NOTIFY_STATECHANGED	280
6.5.3	emNotify - EC_NOTIFY_XXXX	280
6.5.4	Feature Pack MainDevice Redundancy Notifications	281
6.5.5	emNotifyApp	281
6.5.6	emIoCtl - EC_IOCTL_SET_NOTIFICATION_ENABLED	281
6.5.7	emIoCtl - EC_IOCTL_GET_NOTIFICATION_ENABLED	283
6.6	SubDevice control and status functions	284
6.6.1	emGetNumConfiguredSlaves	284
6.6.2	emGetNumConnectedSlaves	284
6.6.3	emGetSlaveId	284
6.6.4	emGetSlaveIdAtPosition	285
6.6.5	emSetSlaveState	285
6.6.6	emSetSlaveStateReq	287
6.6.7	emGetSlaveState	288
6.6.8	emIsSlavePresent	289

6.6.9	emGetSlaveProp	290
6.6.10	emSlaveSerializeMbxTfers	290
6.6.11	emSlaveParallelMbxTfers	291
6.6.12	emIoctl - EC_IOCTL_SET_MBX_RETRYACCESS_PERIOD	292
6.6.13	emNotify - EC_NOTIFY_SLAVE_STATECHANGED	292
6.6.14	emNotify - EC_NOTIFY_SLAVES_STATECHANGED	293
6.6.15	emWriteSlaveRegister	293
6.6.16	emWriteSlaveRegisterReq	295
6.6.17	emReadSlaveRegister	296
6.6.18	emReadSlaveRegisterReq	297
6.6.19	emNotify - EC_NOTIFY_SLAVE_REGISTER_TRANSFER	299
6.6.20	emReadSlaveEEPROM	300
6.6.21	emReadSlaveEEPROMReq	301
6.6.22	emWriteSlaveEEPROM	302
6.6.23	emWriteSlaveEEPROMReq	303
6.6.24	emAssignSlaveEEPROM	304
6.6.25	emAssignSlaveEEPROMReq	305
6.6.26	emActiveSlaveEEPROM	306
6.6.27	emActiveSlaveEEPROMReq	307
6.6.28	emReloadSlaveEEPROM	308
6.6.29	emReloadSlaveEEPROMReq	309
6.6.30	emNotify - EC_NOTIFY_EEPROM_OPERATION	310
6.6.31	emResetSlaveController	312
6.6.32	emIoctl - EC_IOCTL_ALL_SLAVES_MUST_REACH_MASTER_STATE	313
6.6.33	emGetCfgSlaveInfo	313
6.6.34	emGetCfgSlaveEoeInfo	317
6.6.35	emGetCfgSlaveSmInfo	319
6.6.36	emGetBusSlaveInfo	320
6.6.37	emReadSlaveIdentification	324
6.6.38	emReadSlaveIdentificationReq	325
6.6.39	emNotify - EC_NOTIFY_SLAVE_IDENTIFICATION	326
6.6.40	emIoctl - EC_IOCTL_SET_AUTO_ACK_AL_STATUS_ERROR_ENABLED	327
6.6.41	emIoctl - EC_IOCTL_SET_AUTO_ADJUST_CYCCMD_WKC_ENABLED	327
6.6.42	emSetSlaveDisabled	328
6.6.43	emIoctl - EC_IOCTL_SET_SLAVE_MAX_STATE	329
6.6.44	emSetSlaveDisconnected	329
6.6.45	emSetSlavesDisconnected	330
6.6.46	emGetSlavePortState	331
6.6.47	emSetSlavePortState	332
6.6.48	emSetSlavePortStateReq	333
6.6.49	emNotify - EC_NOTIFY_PORT_OPERATION	334
6.6.50	emIoctl - EC_IOCTL_SET_NEW_BUSSLAVES_TO_INIT	335
6.7	Diagnosis, error detection, error notifications	335
6.7.1	emSetLogParms	337
6.7.2	emEthDbgMsg	337
6.7.3	emIoctl - EC_IOCTL_GET_SLVSTATISTICS	338
6.7.4	emGetSlaveStatistics	339
6.7.5	emIoctl - EC_IOCTL_CLR_SLVSTATISTICS	339
6.7.6	emClearSlaveStatistics	340
6.7.7	emIoctl - EC_IOCTL_GET_SLVSTAT_PERIOD	340
6.7.8	emIoctl - EC_IOCTL_SET_SLVSTAT_PERIOD	341
6.7.9	emIoctl - EC_IOCTL_FORCE_SLVSTAT_COLLECTION	341
6.7.10	emIoctl - EC_IOCTL_CLEAR_MASTER_INFO_COUNTERS	341
6.7.11	emIoctl - EC_IOCTL_SET_FRAME_RESPONSE_ERROR_NOTIFY_MASK	343
6.7.12	emIoctl - EC_IOCTL_SET_FRAME_LOSS_SIMULATION	344
6.7.13	emIoctl - EC_IOCTL_SET_RXFRAME_LOSS_SIMULATION	344
6.7.14	emIoctl - EC_IOCTL_SET_TXFRAME_LOSS_SIMULATION	345
6.7.15	Error notifications - general information	345

6.7.16	emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR	347
6.7.17	emNotify - EC_NOTIFY_MASTER_INITCMD_WKC_ERROR	348
6.7.18	emNotify - EC_NOTIFY_SLAVE_INITCMD_WKC_ERROR	348
6.7.19	emNotify - EC_NOTIFY_FOE_MBSLAVE_ERROR	349
6.7.20	emNotify - EC_NOTIFY_EOE_MBXSNW_WKC_ERROR	349
6.7.21	emNotify - EC_NOTIFY_COE_MBXSNW_WKC_ERROR	349
6.7.22	emNotify - EC_NOTIFY_FOE_MBXSNW_WKC_ERROR	349
6.7.23	emNotify - EC_NOTIFY_VOE_MBXSNW_WKC_ERROR	349
6.7.24	emNotify - EC_NOTIFY_S2SMBX_ERROR	349
6.7.25	emNotify - EC_NOTIFY_FRAME_RESPONSE_ERROR	349
6.7.26	emNotify - EC_NOTIFY_SLAVE_INITCMD_RESPONSE_ERROR	352
6.7.27	emNotify - EC_NOTIFY_MBSLAVE_INITCMD_TIMEOUT	353
6.7.28	emNotify - EC_NOTIFY_MASTER_INITCMD_RESPONSE_ERROR	353
6.7.29	emNotify - EC_NOTIFY_NOT_ALL_DEVICES_OPERATIONAL	353
6.7.30	emNotify - EC_NOTIFY_ALL_DEVICES_OPERATIONAL	353
6.7.31	emNotify - EC_NOTIFY_STATUS_SLAVE_ERROR	353
6.7.32	emNotify - EC_NOTIFY_SLAVE_ERROR_STATUS_INFO	354
6.7.33	emNotify - EC_NOTIFY_SLAVES_ERROR_STATUS	354
6.7.34	emNotify - EC_NOTIFY_SLAVE_UNEXPECTED_STATE	355
6.7.35	emNotify - EC_NOTIFY_SLAVES_UNEXPECTED_STATE	355
6.7.36	emNotify - EC_NOTIFY_ETH_LINK_NOT_CONNECTED	356
6.7.37	emNotify - EC_NOTIFY_ETH_LINK_CONNECTED	356
6.7.38	emNotify - EC_NOTIFY_CLIENTREGISTRATION_DROPPED	356
6.7.39	emNotify - EC_NOTIFY_EEPROM_CHECKSUM_ERROR	357
6.7.40	emNotify - EC_NOTIFY_MBXRCV_INVALID_DATA	357
6.7.41	emNotify - EC_NOTIFY_PDIWATCHDOG	357
6.7.42	ecatGetText	358
6.7.43	emLogFrameEnable	358
6.7.44	emLogFrameDisable	359
6.7.45	emGetMasterInfo	360
6.7.46	emGetMemoryUsage	363
6.7.47	emGetMasterDump	364
6.7.48	emGetMasterSyncUnitInfoNumOf	365
6.7.49	emGetMasterSyncUnitInfo	365
6.7.50	emBadConnectionsDetect	366
6.7.51	emBadConnectionsReset	367
6.7.52	emNotify - EC_NOTIFY_BAD_CONNECTION	367
6.7.53	emSelfTestScan	368
6.8	Performance Measurement	370
6.8.1	Enabling performance measurements	370
6.8.2	Retrieving overall performance statistics (min/avg/max)	370
6.8.3	Recording performance histograms	371
6.8.4	Special benchmark types	372
6.8.5	Application benchmarks	372
6.8.6	API	372
6.9	EtherCAT® Mailbox Transfer	380
6.9.1	Mailbox transfer object states	381
6.9.2	emMbxTferCreate	382
6.9.3	emMbxTferAbort	386
6.9.4	emMbxTferDelete	386
6.9.5	emNotify - EC_NOTIFY_MBOXRCV	386
6.10	Automation Device Specification over EtherCAT® (AoE)	387
6.10.1	emAoeGetSlaveNetId	387
6.10.2	emAoeRead	388
6.10.3	emAoeReadReq	389
6.10.4	emNotify - eMbxTferType_AOE_READ	390
6.10.5	emAoeWrite	391
6.10.6	emAoeWriteReq	392

6.10.7	emNotify - eMbxTferType_AOE_WRITE	393
6.10.8	emAoeReadWrite	393
6.10.9	emAoeWriteControl	394
6.10.10	emConvertEcErrorToAdsError	396
6.11	CAN application protocol over EtherCAT® (CoE)	396
6.11.1	emCoeSdoDownload	396
6.11.2	emCoeSdoDownloadReq	398
6.11.3	emNotify - eMbxTferType_COE_SDO_DOWNLOAD	400
6.11.4	emCoeSdoUpload	401
6.11.5	emCoeSdoUploadReq	403
6.11.6	emNotify - eMbxTferType_COE_SDO_UPLOAD	405
6.11.7	emCoeGetODList	406
6.11.8	emCoeGetODListReq	408
6.11.9	emNotify - eMbxTferType_COE_GETODLIST	410
6.11.10	emCoeGetObjectDesc	411
6.11.11	emCoeGetObjectDescReq	412
6.11.12	emNotify - eMbxTferType_COE_GETOBDESC	414
6.11.13	emCoeGetEntryDesc	415
6.11.14	emCoeGetEntryDescReq	418
6.11.15	emNotify - eMbxTferType_COE_GETENTRYDESC	420
6.11.16	emCoeProfileGetChannelInfo	421
6.11.17	emNotify - EC_NOTIFY_COE_INIT_CMD	423
6.11.18	CoE Emergency (emNotify - eMbxTferType_COE_EMERGENCY)	424
6.11.19	CoE Abort (emNotify - EC_NOTIFY_MBSLAVE_COE_SDO_ABORT)	425
6.11.20	emConvertEcErrorToCoeError	426
6.12	File access over EtherCAT® (FoE)	426
6.12.1	Specification	427
6.12.2	emFoeFileDownload	429
6.12.3	emFoeFileUpload	431
6.12.4	emFoeDownloadReq	433
6.12.5	emFoeSegmentedDownloadReq	435
6.12.6	emFoeUploadReq	436
6.12.7	emFoeSegmentedUploadReq	437
6.12.8	emConvertEcErrorToFoeError	441
6.12.9	emNotify - EC_NOTIFY_FOE_MBXSNB_WKC_ERROR	441
6.12.10	emNotify - EC_NOTIFY_FOE_MBSLAVE_ERROR	441
6.12.11	Extending EC_T_MBX_DATA	442
6.13	Servo Drive Profile according to IEC61491 over EtherCAT® (SoE)	442
6.13.1	SoE ElementFlags	443
6.13.2	SoE IDN coding	444
6.13.3	emSoeWrite	444
6.13.4	emSoeWriteReq	445
6.13.5	emSoeRead	446
6.13.6	emSoeReadReq	448
6.13.7	emSoeAbortProcCmd	449
6.13.8	emConvertEcErrorToSoeError	450
6.13.9	emNotify - EC_NOTIFY_SOE_MBXSNB_WKC_ERROR	450
6.13.10	emNotify - EC_NOTIFY_SOE_WRITE_ERROR	450
6.14	Vendor specific protocol over EtherCAT® (VoE)	450
6.14.1	emVoeWrite	450
6.14.2	emVoeWriteReq	451
6.14.3	emVoeRead	452
6.14.4	emNotify - eMbxTferType_VOE_READ	453
6.14.5	emNotify - eMbxTferType_VOE_WRITE	454
6.15	Raw command transfer	454
6.15.1	emTferSingleRawCmd	454
6.15.2	emClntSendRawMbx	456
6.15.3	emClntQueueRawCmd	456

6.15.4	emQueueRawCmd	458
6.15.5	emNotify - EC_NOTIFY_RAWCMD_DONE	459
6.16	EtherCAT® Bus Scan	460
6.16.1	emIoCtl - EC_IOCTL_SB_ENABLE	460
6.16.2	emIoCtl - EC_IOCTL_SB_RESTART	460
6.16.3	emIoCtl - EC_IOCTL_SB_STATUS_GET	461
6.16.4	emIoCtl - EC_IOCTL_SB_SET_TOPOLOGY_CHANGED_DELAY	461
6.16.5	emIoCtl - EC_IOCTL_SB_SET_TOPOLOGY_CHANGED_DELAYS	461
6.16.6	emIoCtl - EC_IOCTL_SB_SET_ERROR_ON_CROSSED_LINES	462
6.16.7	emIoCtl - EC_IOCTL_SB_SET_ERROR_ON_LINE_BREAK	463
6.16.8	emIoCtl - EC_IOCTL_SB_SET_TOPOLOGY_CHANGE_AUTO_MODE	463
6.16.9	emIoCtl - EC_IOCTL_SB_ACCEPT_TOPOLOGY_CHANGE	464
6.16.10	emNotify - EC_NOTIFY_SB_STATUS	464
6.16.11	emNotify - EC_NOTIFY_SB_MISMATCH	465
6.16.12	emNotify - EC_NOTIFY_SB_DUPLICATE_HC_NODE	466
6.16.13	emNotify - EC_NOTIFY_SLAVE_PRESENCE	467
6.16.14	emNotify - EC_NOTIFY_SLAVES_PRESENCE	468
6.16.15	emNotify - EC_NOTIFY_LINE_CROSSED	468
6.16.16	emNotify - EC_NOTIFY_SLAVE_NOTSUPPORTED	469
6.16.17	emNotify - EC_NOTIFY_FRAMELOSS_AFTER_SLAVE	469
6.16.18	emNotify - Bus Scan notifications for Feature Packs	470
6.16.19	emIoCtl - EC_IOCTL_SB_NOTIFY_UNEXPECTED_BUS_SLAVES	470
6.16.20	emIsTopologyChangeDetected	470
6.16.21	emNotify - EC_NOTIFY_HC_TOPOCHGDONE	471
6.16.22	emIoCtl - EC_IOCTL_SB_SET_NO_DC_SLAVES_AFTER_JUNCTION	471
7	RAS-Server for EC-Lyser and EC-Engineer	472
7.1	Integration Requirements	472
7.2	Application programming interface, reference	472
7.2.1	emRasSrvStart	472
7.2.2	emRasSrvStop	475
7.2.3	emRasNotify - xxx	475
7.2.4	emRasNotify - EC_RAS_NOTIFY_CONNECTION	475
7.2.5	emRasNotify - EC_RAS_NOTIFY_REGISTER	476
7.2.6	emRasNotify - EC_RAS_NOTIFY_UNREGISTER	476
7.2.7	emRasNotify - EC_RAS_NOTIFY_MARSHALERROR	477
7.2.8	emRasNotify - EC_RAS_NOTIFY_ACKERROR	477
7.2.9	emRasNotify - EC_RAS_NOTIFY_NONOTIFYMEMORY	478
7.2.10	emRasNotify - EC_RAS_NOTIFY_STDNOTIFYMEMORYSMALL	478
7.2.11	emRasNotify - EC_RAS_NOTIFY_MBXNOTIFYMEMORYSMALL	478
8	Error Codes	480
8.1	Groups	480
8.2	Generic Error Codes	480
8.3	DCM Error Codes	491
8.4	ADS over EtherCAT® (AoE) Error Codes	493
8.5	CAN application protocol over EtherCAT® (CoE) SDO Error Codes	495
8.6	File Transfer over EtherCAT® (FoE) Error Codes	498
8.7	Servo Drive Profil over EtherCAT® (SoE) Error Codes	500
8.8	Remote API Error Codes	504

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 MainDevice Classes

Feature name	Short description	EC-Master Class A	EC-Master Class B	Feature ID
Basic Features				
Service Commands	Support of all commands	✓	✓	101
IRQ field in datagram	Use IRQ information from SubDevice in datagram header	✓	✓	102
SubDevices with Device Emulation	Support SubDevices with and without application controller	✓	✓	103
EtherCAT® State Machine	Support of ESM special behavior	✓	✓	104
Error Handling	Checking of network or SubDevice 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	?	108
Process Data Exchange				
Cyclic PDO	Cyclic process data exchange	✓	✓	201
Multiple Tasks	Different cycle tasks Multiple update rates for PDO	✓	✓	202

continues on next page

Table 1 – continued from previous page

Feature name	Short description	EC-Master Class A	EC-Master Class B	Feature ID
Frame repetition	Send cyclic frames multiple times to increase immunity	?	?	203
Network Configuration				
Online scanning	Network configuration functionality included in EtherCAT® MainDevice	✓	✓	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 ID	Identification used for Hot Connect and prevention against cable swapping	✓	✓	303
Station Alias Addressing	Support configured station alias in SubDevice, i.e. enable 2nd Address and use it	✓	✓	304
Access to EEPROM	Support routines to access EEPROM via ESC register	✓	✓	305
Mailbox Support				
Support Mailbox	Main functionality for mailbox transfer	✓	✓	401
Mailbox Resilient Layer	Support underlying resilient layer	✓	✓	402
Multiple Mailbox channels		✓	✓	403
Mailbox polling	Polling Mailbox state in SubDevices	✓	✓	404
CAN application layer over IETHERCAT_RI (CoE)				
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

continues on next page

Table 1 – continued from previous page

Feature name	Short description	EC-Master Class A	EC-Master Class B	Feature ID
Emergency Message	Receive Emergency messages	✓	✓	505
PDO in CoE	PDO services transmitted via CoE	?	?	506
EoE				
EoE protocol	Services for tunneling Ethernet frames. includes all specified EoE services	✓	✓	601
Virtual Switch	Virtual Switch functionality	✓	✓	602
EoE Endpoint to Operation Systems	Interface to the Operation System on top of the EoE layer	FP	?	603
FoE				
FoE Protocol	Support FoE Protocol	✓	✓	701
Firmware Up-/Download	Password, File-Name should be given by the application	✓	✓	702
Boot State	Support Boot-State for Firmware Up/Download	✓	✓	703
SoE				
SoE Protocol	Support SoE Services	✓	✓	801
AoE				
AoE Protocol	Support AoE Protocol	✓	✓	901
VoE				
VoE Protocol	External Connectivity supported	✓	✓	1001
Synchronization with Distributed Clock (DC)				
DC support	Support of Distributed Clocks	✓	?	1101
Continuous Propagation Delay compensation	Continuous Calculation of the propagation delay	✓	?	1102
Sync window monitoring	Continuous monitoring of the Synchronization difference between SubDevices	✓	?	1103
SubDevice-to-SubDevice Communication				

continues on next page

Table 1 – continued from previous page

Feature name	Short description	EC-Master Class A	EC-Master Class B	Feature ID
via MainDevice	Information is given in ENI file or can be part of any other network configuration. Copying of the data can be handled by MainDevice stack or MainDevice's application	✓	✓	1201
MainDevice information				
MainDevice Object Dictionary		FP	?	1301

² : According to ETG.1500 MainDevice Classes not mandatory for Class B

¹ : According to ETG.1500 MainDevice Classes not mandatory for Class A

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

Dongled

Encrypted binary

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
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
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
```

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

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

1.4 Dongled version

The Master library is encrypted and requires a dongle to execute. The following dongles are supported:

- USB dongle
- ASIC dongle
- Virtual dongle using a WibuCmRaU file

1.4.1 Licensing using a virtual WibuCmRaU File on Linux

To create a virtual dongle using a WibuCmRaU file on a Linux machine, the following steps are required:

- Download the codemeter package from Wibu website and install it on the linux machine:
- **Import the LIF file provided by acontis**

```
cmu --import --file *.WibuCmLIF
```

- **List the Wibu licenses, to get the serial number**

```
cmu --list
```

- **Use the serial number to generate a WibuCmRaC file**

```
cmu --serial your_serial_number --context 6001978 --file *.WibuCmRaC
```

- Send the WibuCmRaC to acontis and acontis will provide a WibuCmRaU file.
- **Import the WibuCmRaU file**

```
cmu --import --file *.WibuCmRaU
```

1.4.2 Maintain the license after a linux image update

In case of a complete Linux image update, the following steps are required:

- Make a backup of the WibuCmRaU file and the following folder /var/lib/Codemeter/CmAct from the old image.
- On the new image install Codemeter
- Restore the 6001978_* files from /var/lib/Codemeter/CmAct
- **Change the user rights of the 4 files to daemon**

```
sudo chown daemon:daemon 6001978_*
```

- **Import the WibuCmRaU backup file**

```
cmu --import --file *.WibuCmRaU
```

1.5 License

1.5.1 EC-Master license

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

1.5.2 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 Waclawek <karl@wacławek.net>
Copyright (c) 2016-2024 Sebastian Pipping <sebastian@pipping.org>
Copyright (c) 2016      Cristian Rodriguez <crrodriguez@opensuse.org>
Copyright (c) 2016      Thomas Beutlich <tc@tbeu.de>
Copyright (c) 2017      Rhodri James <rhodri@wildebeest.org.uk>
Copyright (c) 2022      Thijs Schreijer <thijs@thijsschreijer.nl>
Copyright (c) 2023      Hanno Böck <hanno@gentoo.org>
Copyright (c) 2023      Sony Corporation / Snild Dolkow <snild@sony.com>
Licensed under the MIT license:
```

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

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

```
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
```

(continues on next page)

(continued from previous page)

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.

1.5.3 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.

1.6 Versioning

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

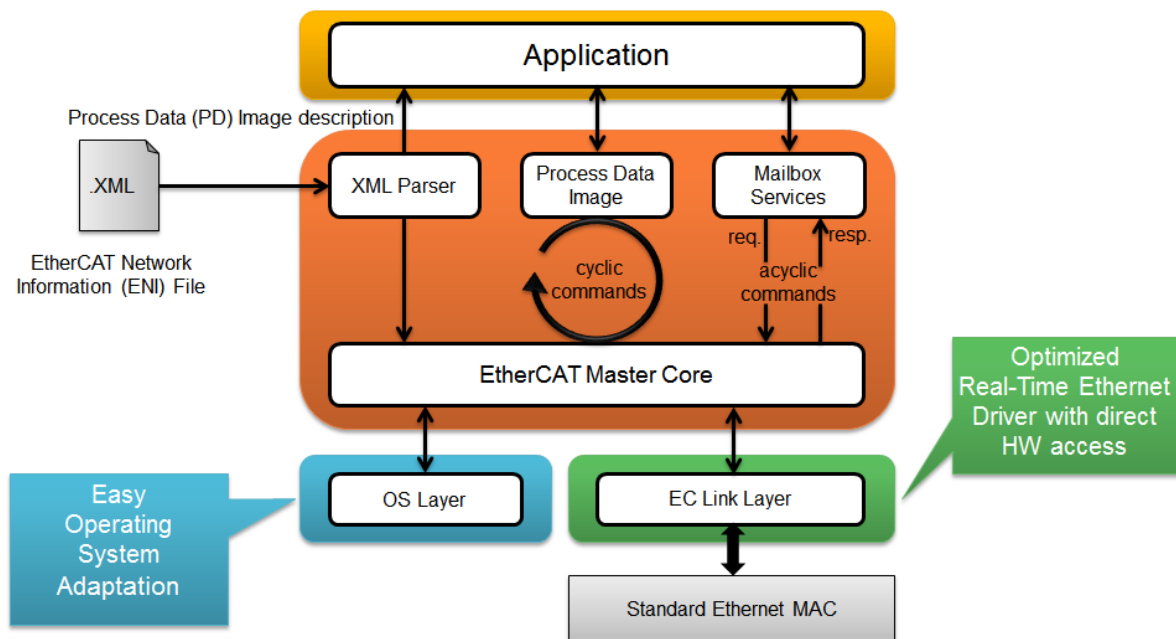
The libraries are binary compatible by unchanged *MAJOR* and *MINOR* digits. If *SERVICEPACK* increments, *BUILD* restarts with 01. *BUILD* 99 is reserved for test builds that have not been officially released for productive usage.

2 Getting Started

2.1 EC-Master Architecture

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

The MainDevice stack is divided into modules, see diagram and descriptions below:



- **EtherCAT® MainDevice 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® MainDevice is configured using an 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 MainDevice and the SubDevices. 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® MainDevice has to know about the EtherCAT® bus topology and the cyclic/acyclic frames to exchange with the SubDevices. This configuration is determined in a configuration file which has to be available in the EtherCAT® Network Information Format (ENI). This format is completely independent from EtherCAT® SubDevice vendors, from EtherCAT® MainDevice 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® MainDevice 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
- MainDevice 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 [ms]. When the time expires the demo application exits completely.

<time>
Time [ms], 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 μ s for all EtherCAT® jobs (e.g. ProcessAllRxFrames).
- <level>**
Depending on level the performance histogram can be activated as well.
- log** <prefix> [<msg cnt>]
Use given file name prefix for log files.
- <prefix>**
- <msg cnt>**
Messages count for log buffer allocation
- lic** <key>
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 = SubDevice 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

-cpsw <instance> <mode> <portpriority> <masterflag> <refboard>**Hardware: TI CPSW****<Instance>**

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

<Mode>

0 = Interrupt mode, 1 = Polling mode

<PortPriority>

Low priority (0) or high priority (1)

<MasterFlag>

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

<RefBoard>

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

If custom:**<CpswType>**

am33XX | am437X | am57X | am387X

<PhyAddress>

0 ... 31

<PhyInterface>

rmii | gmii | rgmii | osdriver

<NotUseDmaBuffers>

0 = FALSE | 1 = TRUE

-dpdk <instance> <mode> <port>**Hardware: Hardware independent, only available for Linux.****<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>.

-dpdkenetc4 <instance> <mode> <port>

Hardware: NXP iMX95 ENETC4, only available for Linux.

<Instance>

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

<Mode>

0 = Interrupt mode (not implemented), 1 = Polling mode

<Port>

DPDK port ID, e.g. 0

-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 | sgmmii | rgmmii | 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 | sgmmii | rgmmii | 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 | polarfire | 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 | polarfire

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
Enable Real-time Ethernet Driver Time Triggered Send (TTS)

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

or

--tmr
Enable 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> [--mii <phyaddress>|--osdriver]
\ [--nophyreset] [--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:

--mii
PhyInterface mii

<PhyAddress>
0 ... 31

or

--osdriver
PhyInterface osdriver

Optional:

--nophyreset
No PHY Reset

Optional:

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

<SendOffset>
TTS cyclic frame send offset from cycle start [us]

-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 <adapternam>] [--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**

Disable force broadcast

-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

Multiplier port in use

<Port>

0 = X1, 1 = X2, ...

--link

Link parameters of network adapter connected to the uplink port

<Link parms>

e.g. -intelgbe ...

Optional:**--type**

Multiplier type

<Type>

cu2508 = CU2508 Ethernet-Port-Multiplier | et2000 = ET2000 Multichannel Ethernet-Probe

-remote <instance> <mode> <src ip> <src port> <dst ip> <dst port> [--mac <address>]
 \ [--rxbuffercnt <count>]

Hardware: Hardware independent. Tunnels EtherCAT frames through a TCP socket

<Instance>

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

<Mode>

1 = Polling mode

<Src ip>

Source adapter IP address (listen)

<Src port>
Source port number (listen)

<Dst ip>
Destination adapter IP address (connect)

<Dst port>
Destination port number (connect)

Optional:

--mac
MAC Address

<Address>
formatted as xx:xx:xx:xx:xx:xx or xx-xx-xx-xx-xx-xx

Optional:

--rxbuffercnt
Frame buffer count for interrupt service thread (IST)

<Count>
Buffer count

-r6040 <instance> <mode>

Hardware: RDC R6040

<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

-rzt1 <instance>

Hardware: Renesas RZ/T1

<Instance>

Device instance 1 = Port 0 | 2 = Port 1

-sheth <instance> <mode> <refboard>**Hardware: Renesas RZG1 or Armadillo-800 EVA or MYIR Remi Pi (Renesas RZ/G2L) or Renesas RX72N Envision Kit****<Instance>**

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

<Mode>

1 = Polling mode

<RefBoard>

rzg1e | a800eva | rzg2l | rx72n

-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 adapter, e.g. eth1

Optional:**<Mode>**

0 = Interrupt mode | 1 = Polling mode

Optional:**--nommaprx**

Disable PACKET_MMAP for receive

Optional:**--promiscuousmode**

Enable promiscuous mode

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

Network adapter, e.g. eth1

<Mode>

0 = Interrupt mode | 1 = Polling mode

Optional:

--queue
Set Queue Id

<Id>
Queue Id, e.g. 0, 1, ... (Default 0)

Optional:

--xdpmode
XDP Mode

<Mode>
SKB(1), DRV(2), HW(3) or DRV_ZEROCOPY(4). Default is SKB(1)

-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> is a placeholder for the architecture. Different architectures are supported.

2.5.1 EtherCAT® MainDevice Software Development Kit (SDK)

The EtherCAT® MainDevice development kit is needed to write applications based on the MainDevice stack. The MainDevice 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 MainDevice 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 are located in the following directories:

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

3 Software Integration

3.1 Timing

3.1.1 JobTask: Interaction between application and EtherCAT® network

EtherCAT® is a cycle-based fieldbus technology. The EC-Master has no internal tasks and the timing of the cyclic operation is fully controlled by the application. To implement this, the application contains a thread that cyclically calls `emExecJob()` / `ecatExecJob()`. Within the `emExecJob()` calls, the EC-Master sends and processes the EtherCAT® frames and handles internal state machines. This thread is referred to as the *JobTask* .

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

Note: For real-time EtherCAT® applications, this task must be scheduled in real-time.

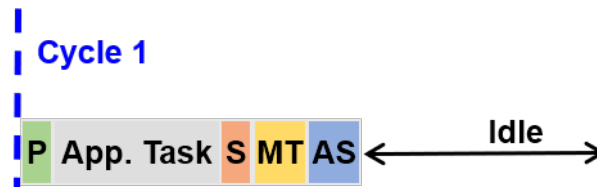
Note: Blocking APIs such as `emCoeSdoDownload()`, which depend on the JobTask and wait for results, must not be called within the JobTask as it would result in a lock-up of the JobTask.

Note: It is highly recommended that there is a single task calling all required jobs to run the stack as demonstrated in the `EcMasterDemo`.

3.1.2 Timing models

From the application perspective, the 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 MainDevice and SubDevice 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 `emExecJob(eUsrJob_SendAllCycFrames)` no further output process data synchronization is necessary. The application is responsible to (cyclically) call 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 MainDevice in every cycle
- Defined by the configuration tool (which data to read and to write)

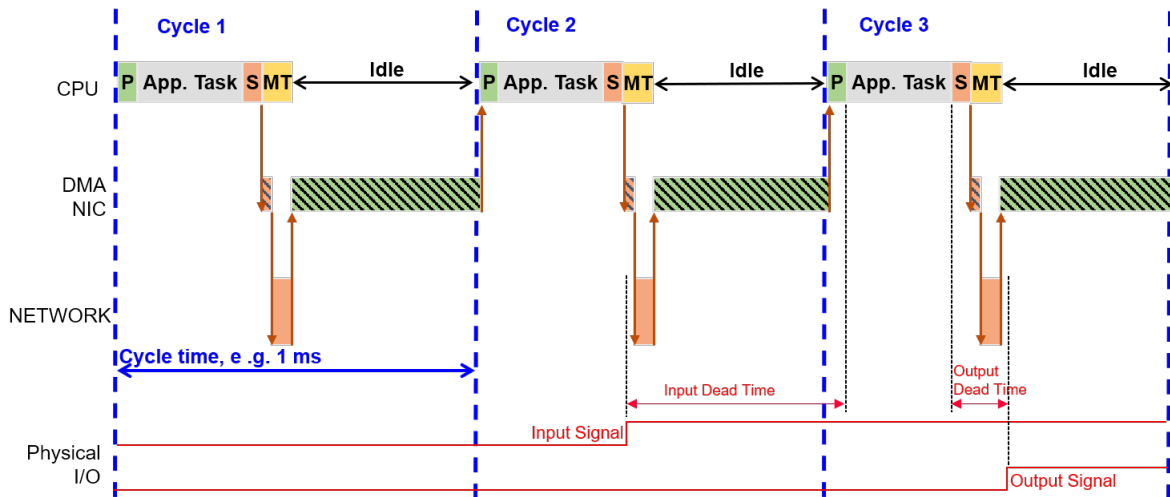
2. Acyclic frames

- Asynchronous, event triggered communication

- Mailbox communication (CoE, FoE, EoE)
- Status requests (e. g. read SubDevice state information)
- Raw EtherCAT® datagrams requested by application

3.1.3 Standard EtherCAT® Timing: Short output dead time

Cyclic frames



Application has to perform:

```

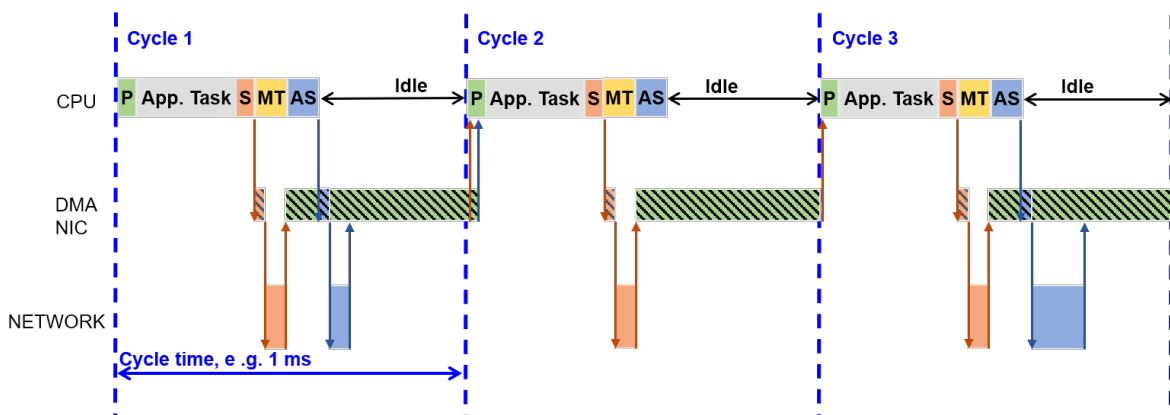
/* Job P: Process data are saved in the process data image */
emExecJob(dwInstanceId, eUsrJob_ProcessAllRxFrames, &oJobParms);

/* 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);

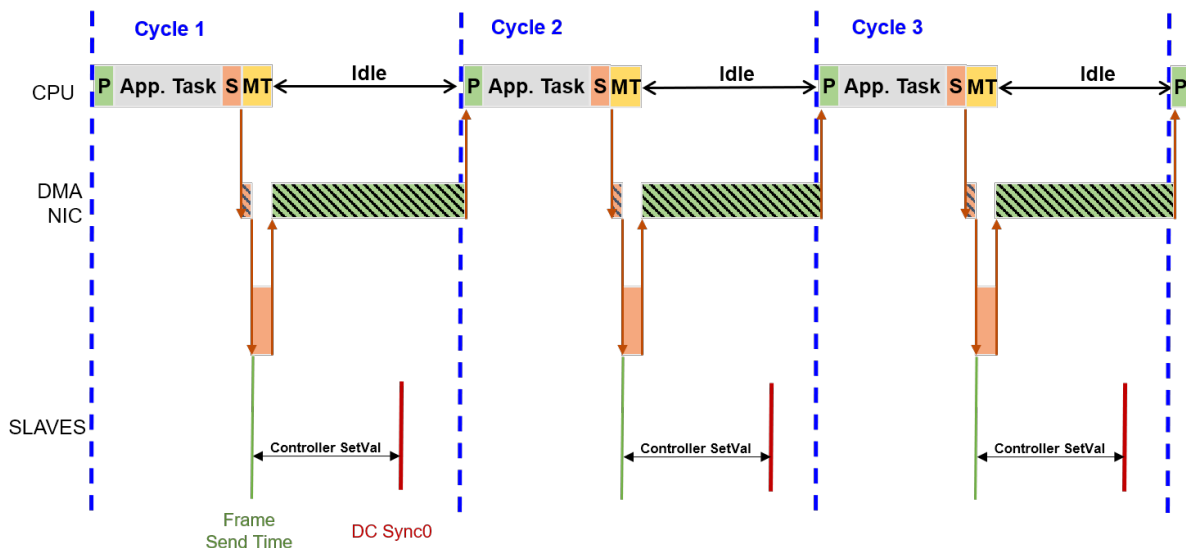
/* App. Task */

/* 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:

```

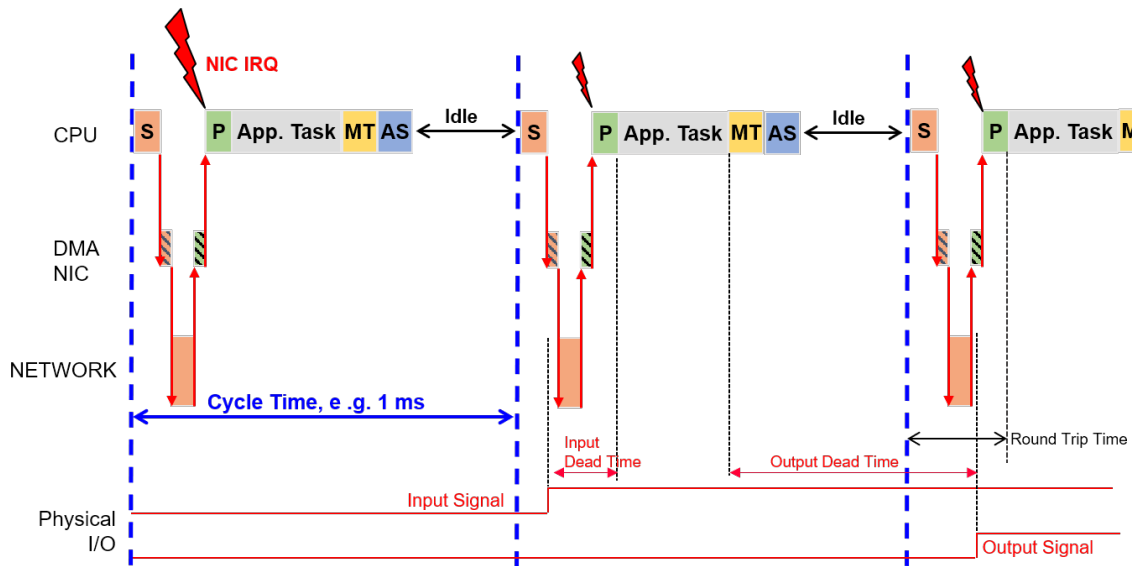
/* Job P: Process data are saved in the process data image */
emExecJob(dwInstanceId, eUsrJob_ProcessAllRxFrames, &oJobParms);

/* 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);
    
```

3.1.4 Alternative EtherCAT® 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, eUserJob_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`

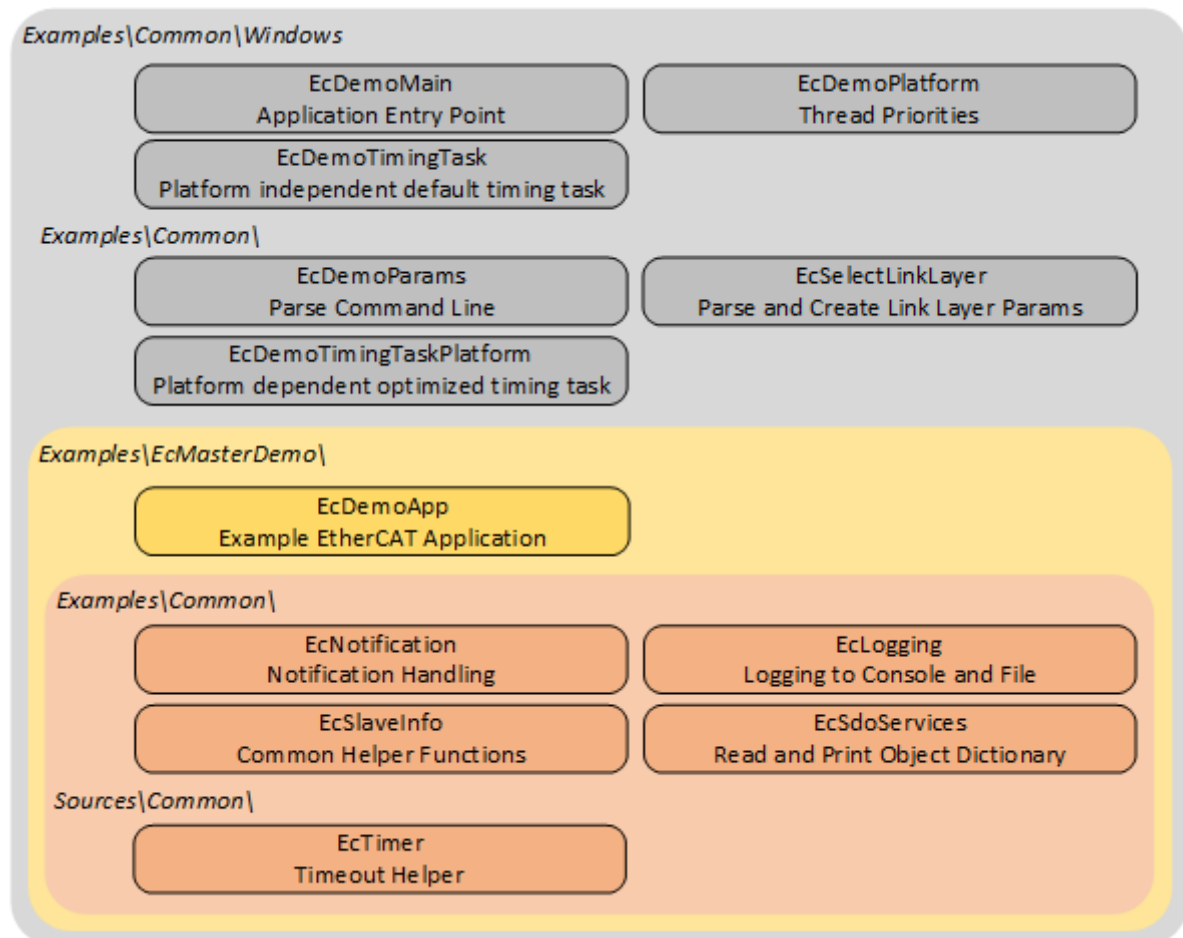
3.2 Example application

The example application EcMasterDemo handles the following tasks:

- Showing basic EtherCAT® communication
- EC-Master stack initialization
- Start (set all SubDevices 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

Entrypoint 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® MainDevice

EcDemoApp.h

Application specific settings for EcDemoApp

EcDemoParams.cpp

Parsing of command line parameters

EcDemoParams.h

Basic configuration structs and EC-Master parameters

EcSelectLinkLayer.cpp

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

EcNotification.cpp

SubDevice monitoring and error detection (function `emNotify()`)

EcSdoServices.cpp

CoE object dictionary example

EcSlaveInfo.cpp

SubDevice information services (bus scan, SubDevice properties, getting information of SubDevices connected to the EtherCAT® bus)

EcLogging.cpp

Message logging functions

EcTimer.cpp

Start and monitor timeouts

3.2.2 EC-Master lifecycle

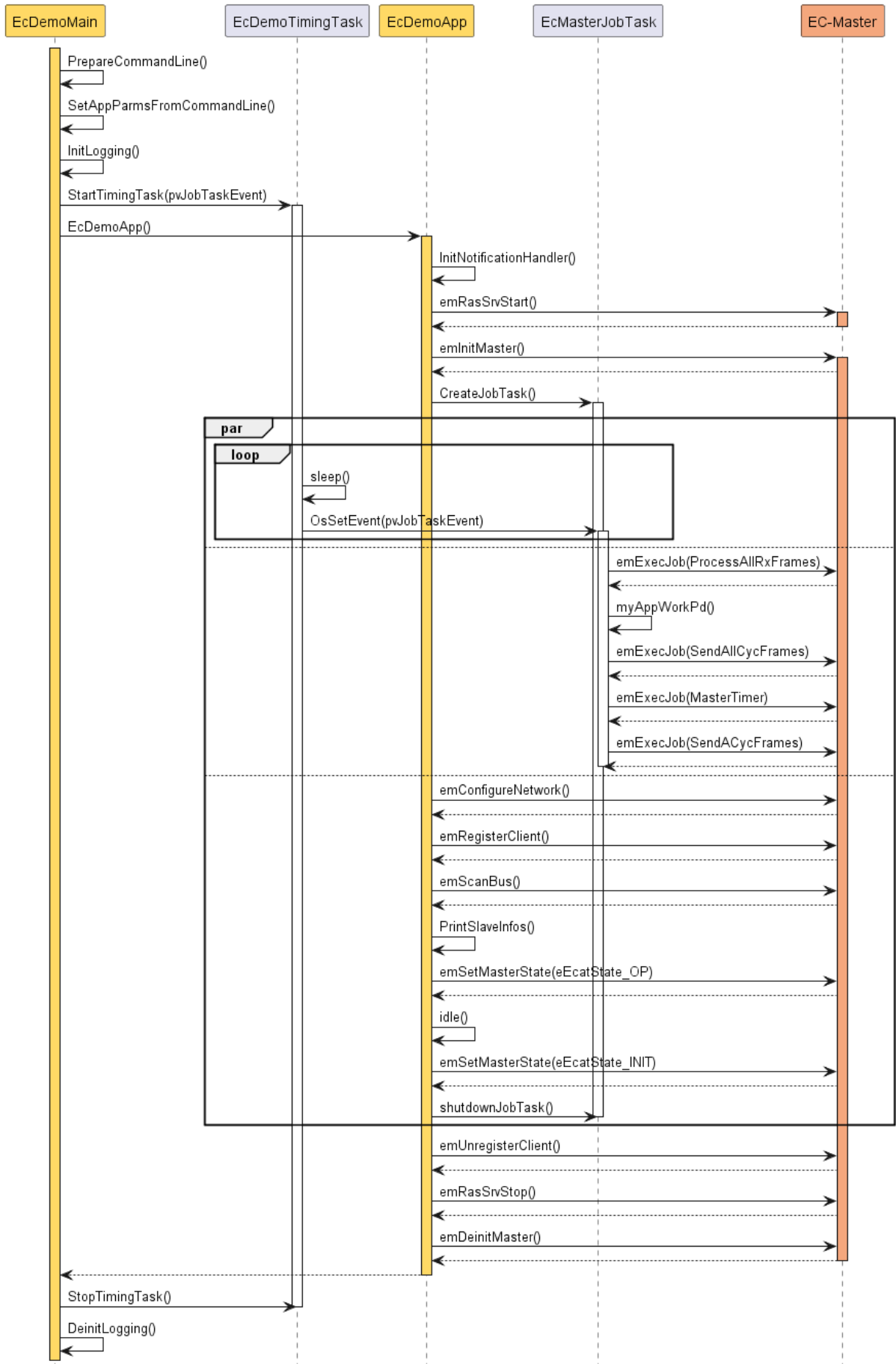
This chapter provides a brief overview of starting and stopping the EC-Master. Basically the operation of the EC-Master is wrapped between the functions:

- `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 MainDevice 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 myAppWorkpd() 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 EC-Master for operation and set operational parameters, e.g. used Real-time Ethernet Driver, buffer sizes, maximum number of SubDevices,

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()

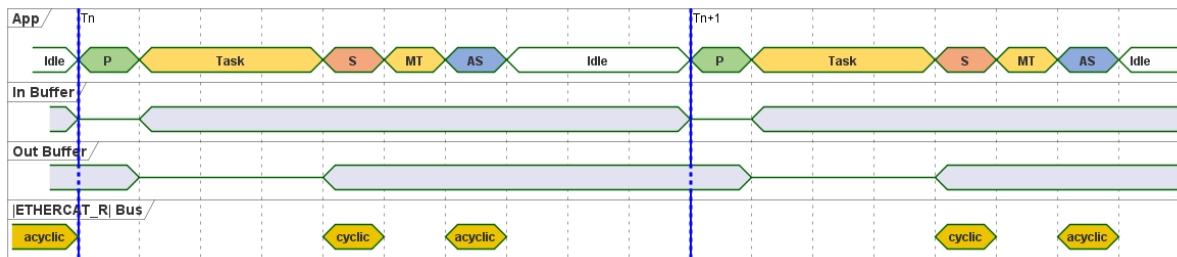
EC-Master API function: Start the EC-Master and switch the bus to the different states from INIT to OPERATIONAL.

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® MainDevice 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® MainDevice 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 as 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 EC-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

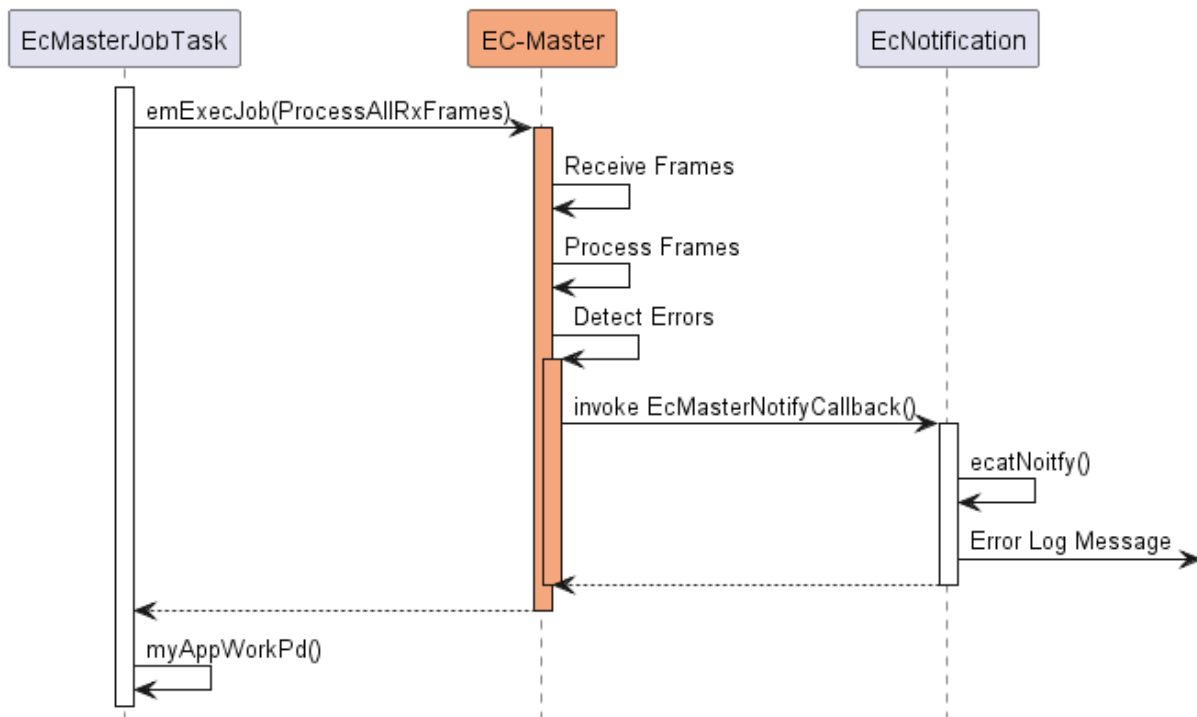
```
EC_T_DWORD emRegisterClient(EC_T_DWORD dwInstanceID, EC_PF_NOTIFY pfnNotify, EC_T_VOID *pCallerData, EC_T_REGISTERRESULTS *pRegResults)
```

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

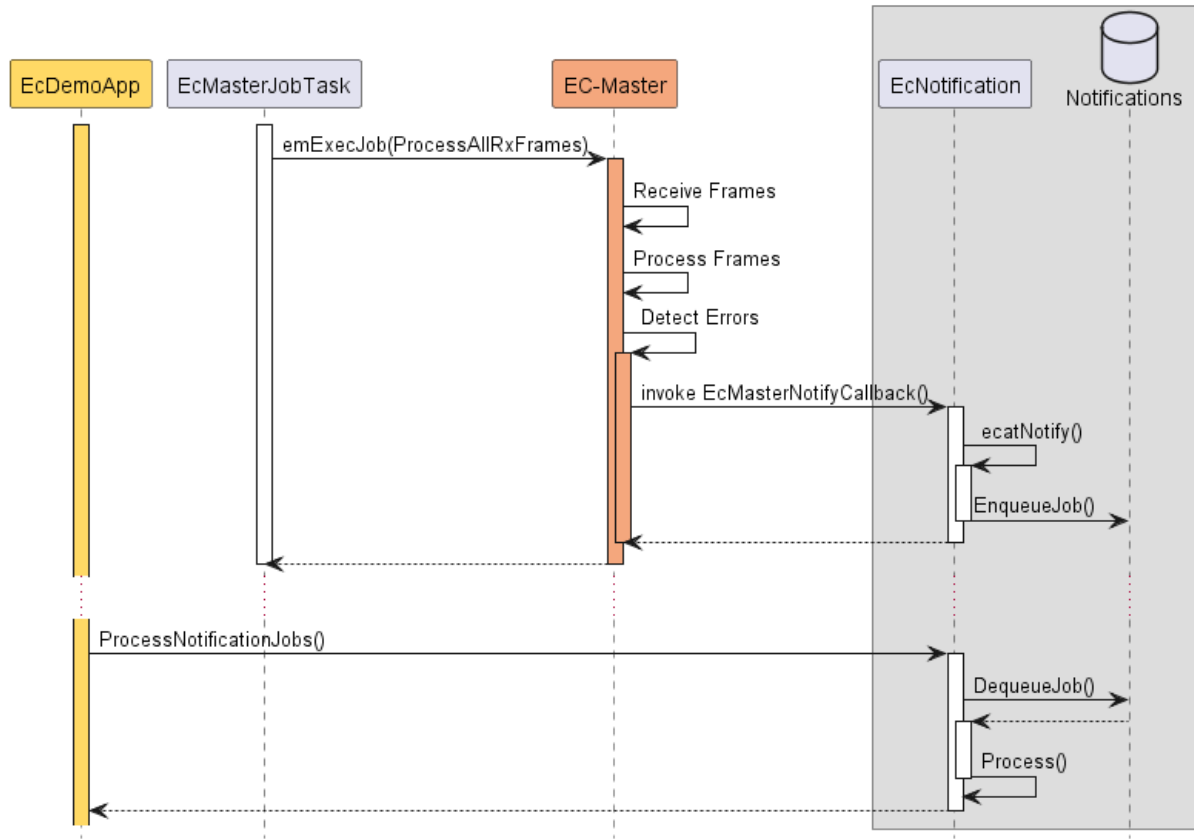
Smaller events can be handled directly in the context in which they are detected. A possible example of such an event is the detection of a false work counter (WKC). The procedure is as follows:



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 `EcMasterDemo` 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 `EcMasterDemo` 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_PARAMS::dwLogLevel`. `EcLogging.cpp` has various parameters beside the log level, like Roll Over setting, log task prio and affinity, log buffer size, etc. See `EcMasterDemo` for reference.

Configure EC-Master logging

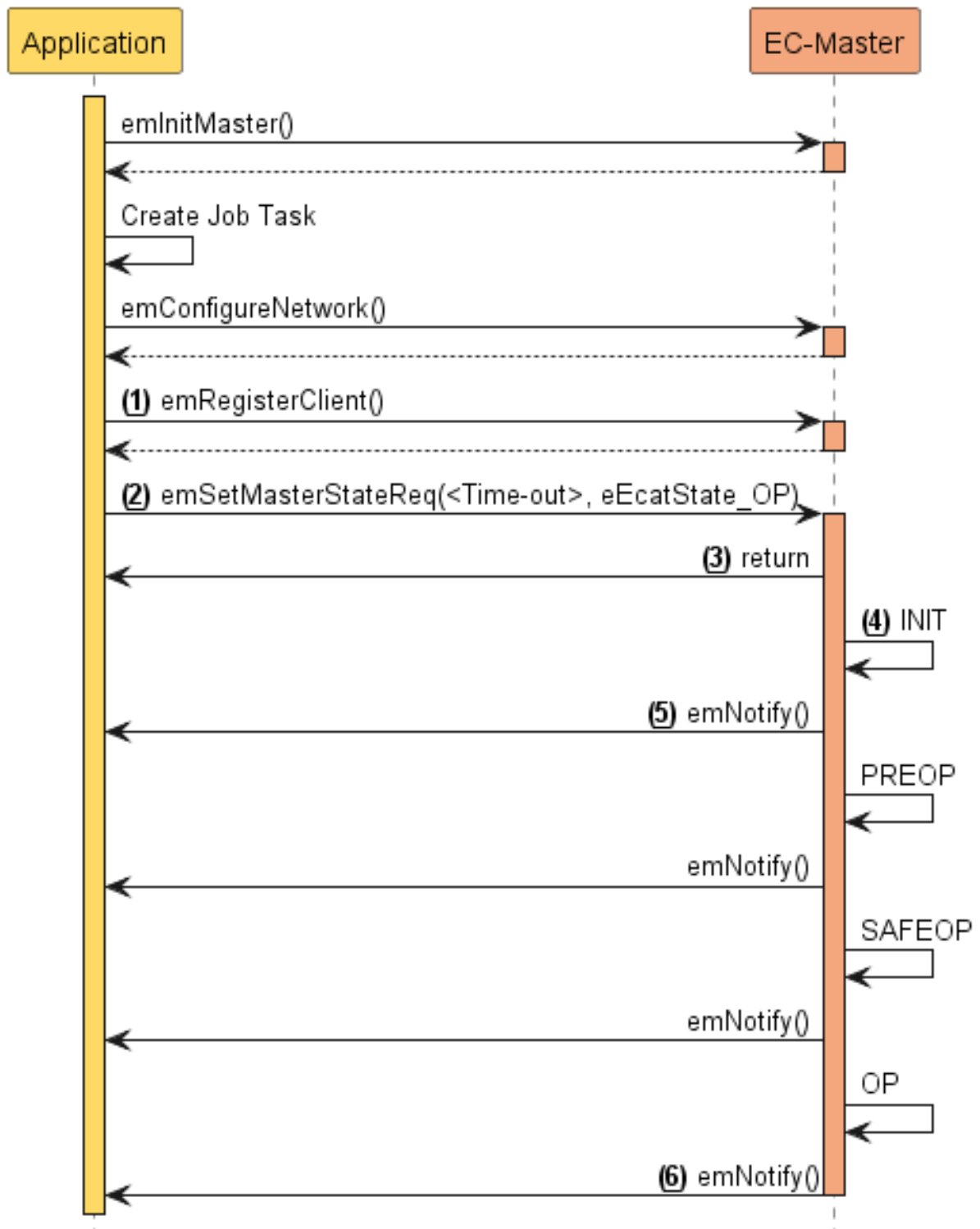
The EC-Master logging is configured on initialization, see `EC_T_INIT_MASTER_PARAMS::LogParams` in `em-InitMaster()`. The application can provide customized log message handlers of type `EC_PF_LOGMSGHK` if the default handler in `EcLogging.cpp` does not fulfill the needs of the application.

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

3.3 EC-Master startup

The EC-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 EC-Master. Finally, after all clients are registered the `MainDevice` can be started. Starting the `MainDevice` means that all `SubDevices` will be set into the operational state. Every time the state of the `MainDevice` has changed the clients are notified about this state-change.

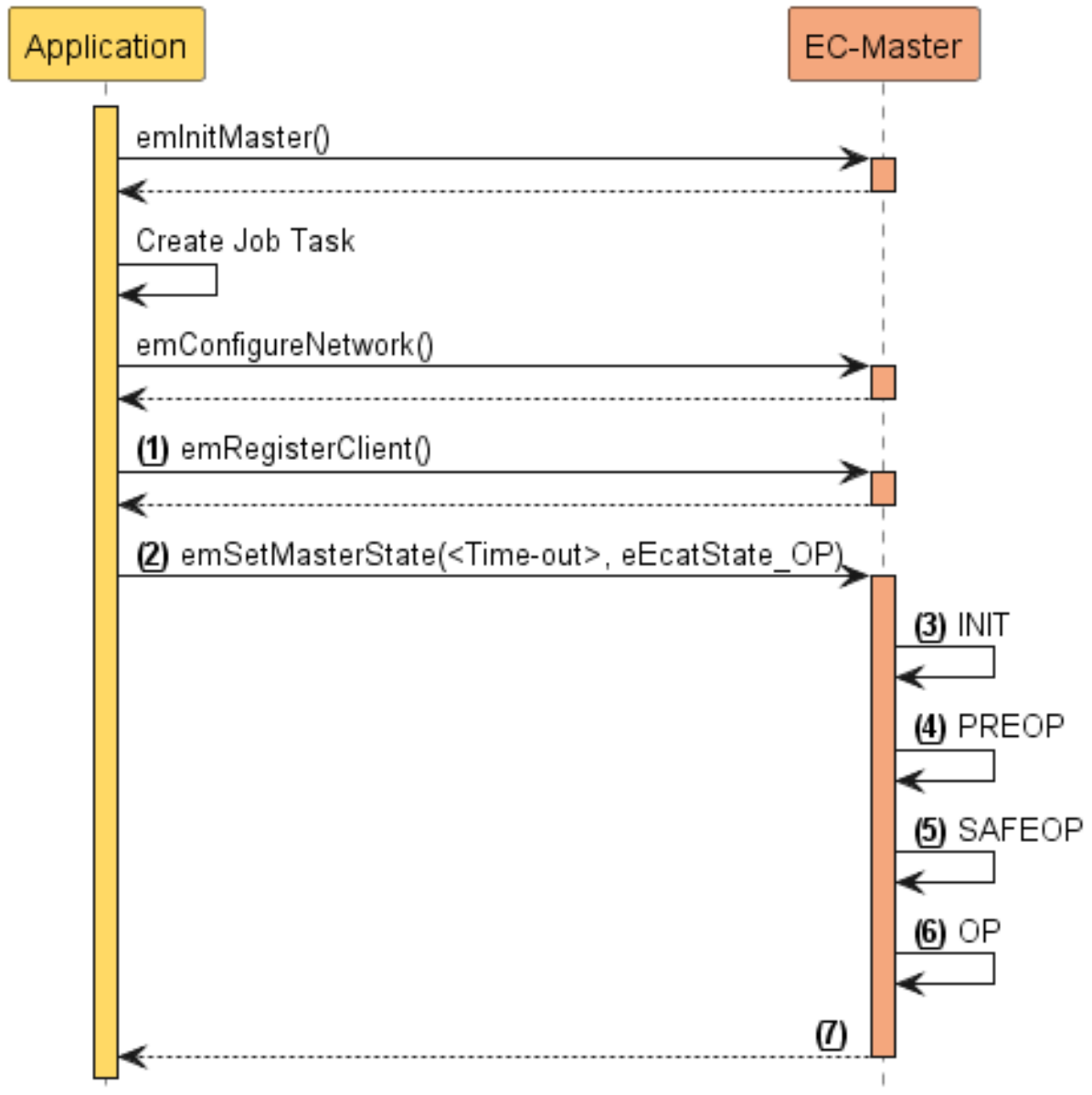
3.3.1 Asynchronous (deferred) startup



- Application calls `emInitMaster()` (...)
- Application creates Job Task. See *EC-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 MainDevice startup procedure (See “4”)
- The MainDevice initializes all SubDevices until all SubDevices 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 *EC-Master lifecycle*.
- Application calls `emConfigureNetwork()` (...)
- Application calls `emRegisterClient()` (...) (See “1”)
- Application calls `emSetMasterState()` (...) with an appropriate timeout value (See “2”)

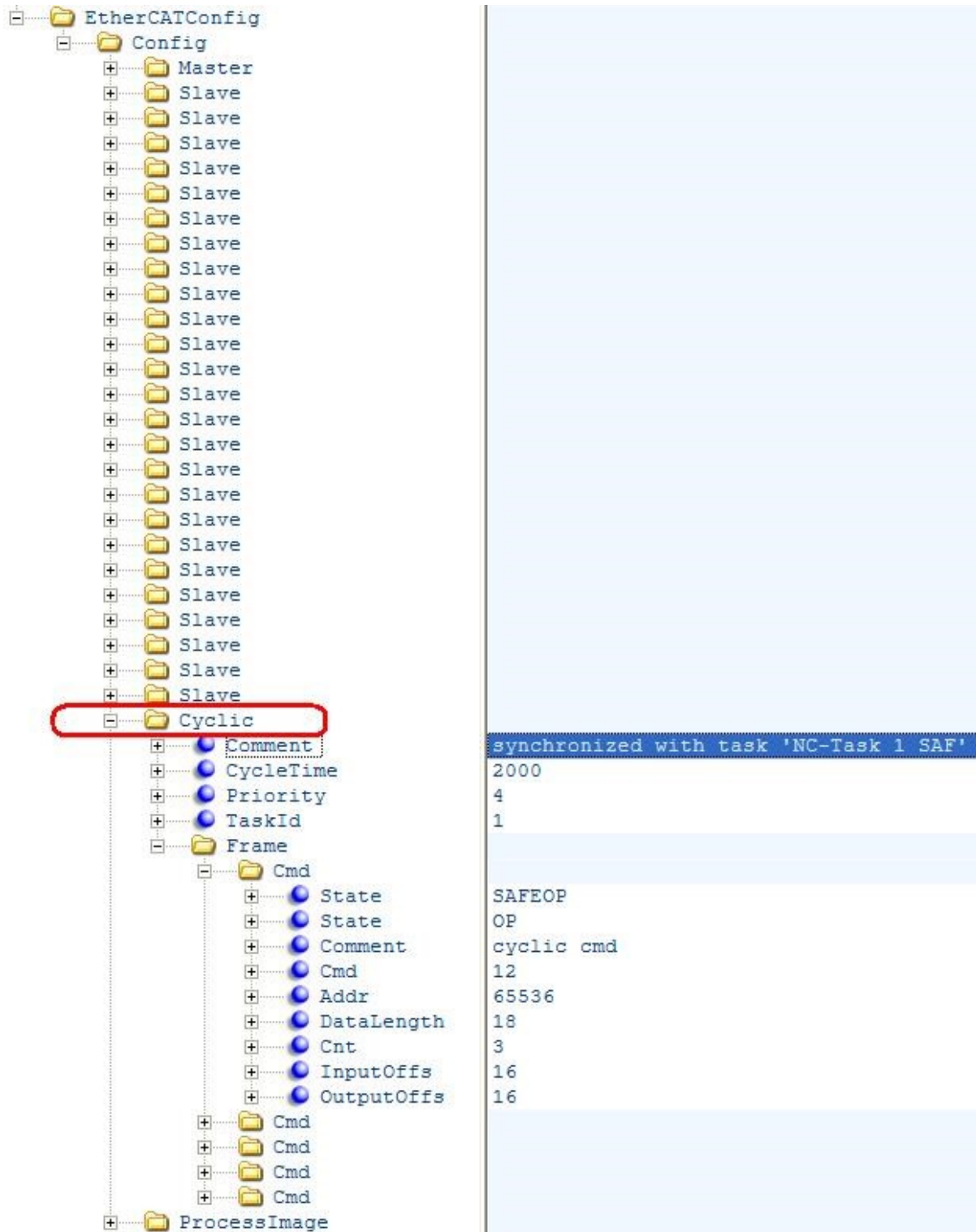
- Inside `emSetMasterState () (...)` the MainDevice startup procedure will be initiated (See “3”)
- The application is blocked until the whole startup has finished (See “7”)
- The MainDevice initializes all SubDevices until all SubDevices reach OPERATIONAL state (See “3-6”)
- After reaching the OPERATIONAL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



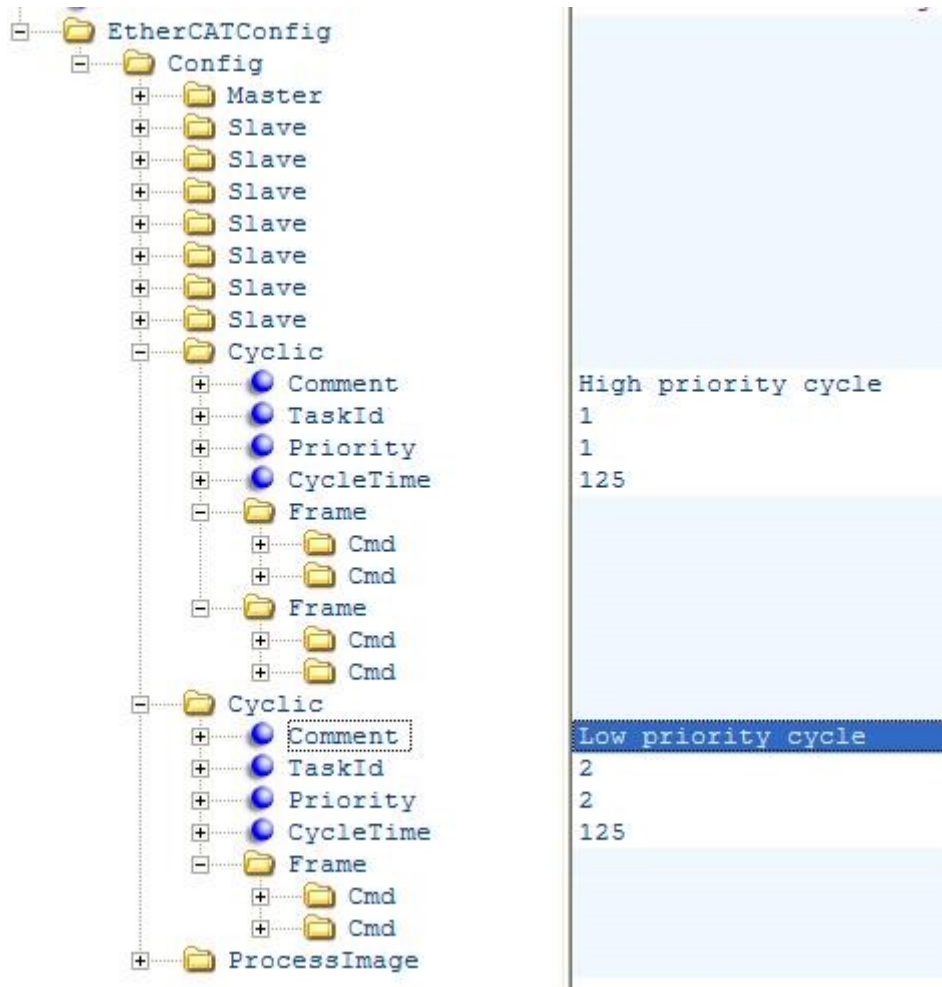
The screenshot displays the configuration interface for EtherCAT. On the left, a tree view shows the hierarchy: EtherCATConfig > Config > Master, Slave (1-20), Cyclic (highlighted with a red box), and ProcessImage. The 'Cyclic' folder is expanded to show parameters: Comment, CycleTime, Priority, TaskId, Frame, and four 'Cmd' folders. The 'Frame' folder is further expanded to show parameters: State, Comment, Cmd, Addr, DataLength, Cnt, InputOffs, and OutputOffs. The right pane shows the configuration details for the selected 'Cyclic' task, with the following data:

synchronized with task 'NC-Task 1 SAF'	
CycleTime	2000
Priority	4
TaskId	1
State	SAFEOP
Comment	OP
Cmd	cyclic cmd
Addr	12
DataLength	65536
Cnt	18
InputOffs	3
OutputOffs	16

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 EC-Master to send the appropriate cyclic frame.

See also:

`emExecJob ()`

3.4.3 Copy Information for SubDevice-to-SubDevice 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 `emExecJob(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.

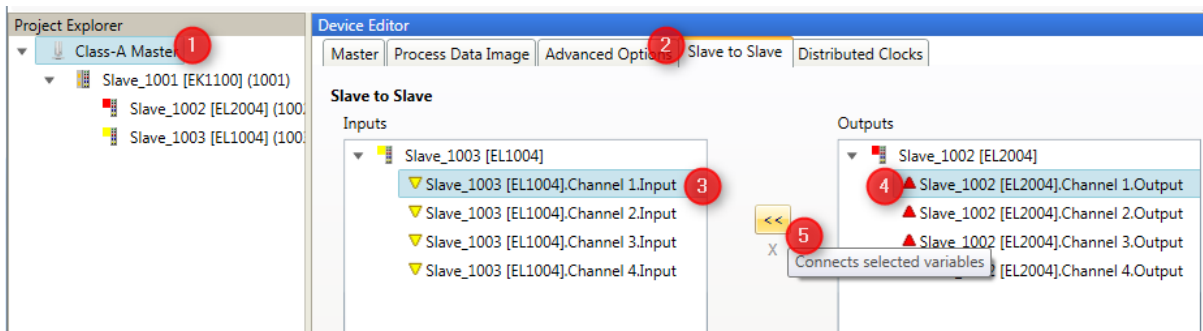
The copy info declarations are located at `/EtherCATConfig/Config/Cyclic/Frame/Command/CopyInfos` in the ENI file.

See also:

- *Cyclic cmd WKC validation and Frame Loss*
- CopyInfosType in ETG.2100

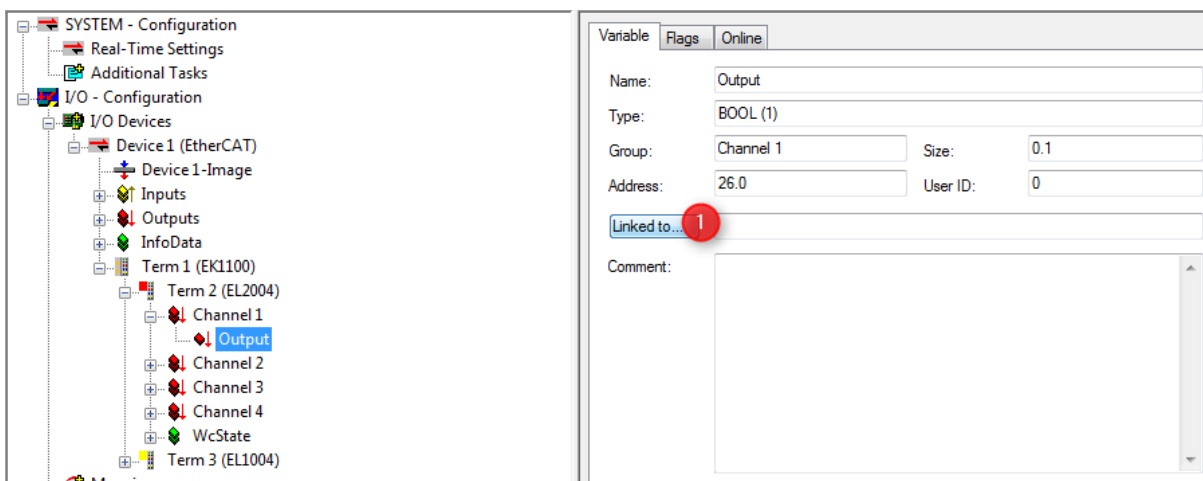
Configuration with EC-Engineer

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

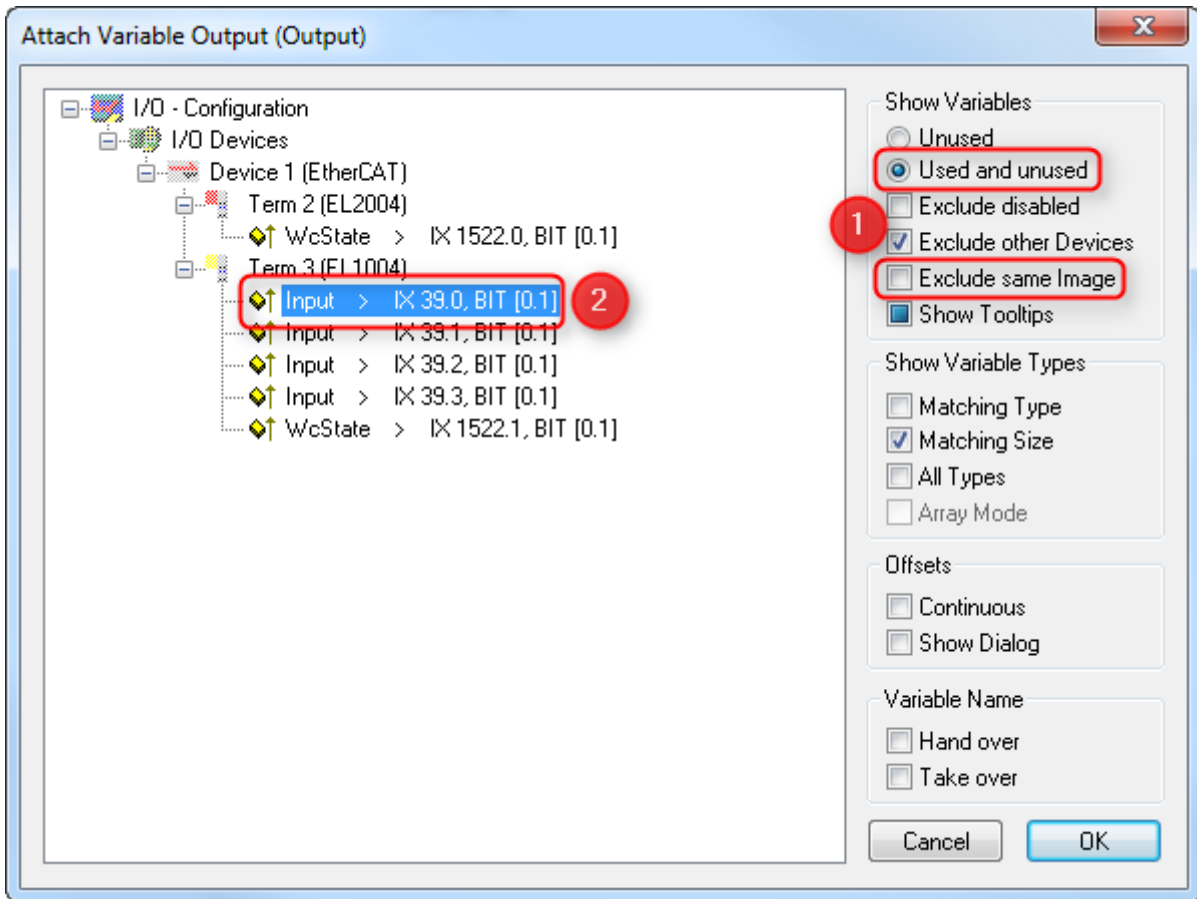


Configuration with ET9000

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



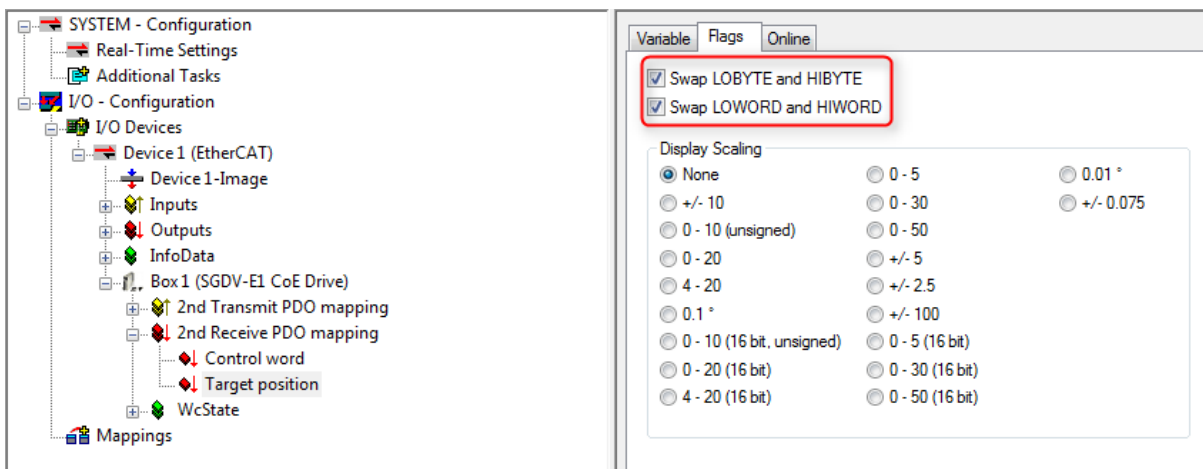
2. Select Input Variable to be attached to the Output Variable:



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

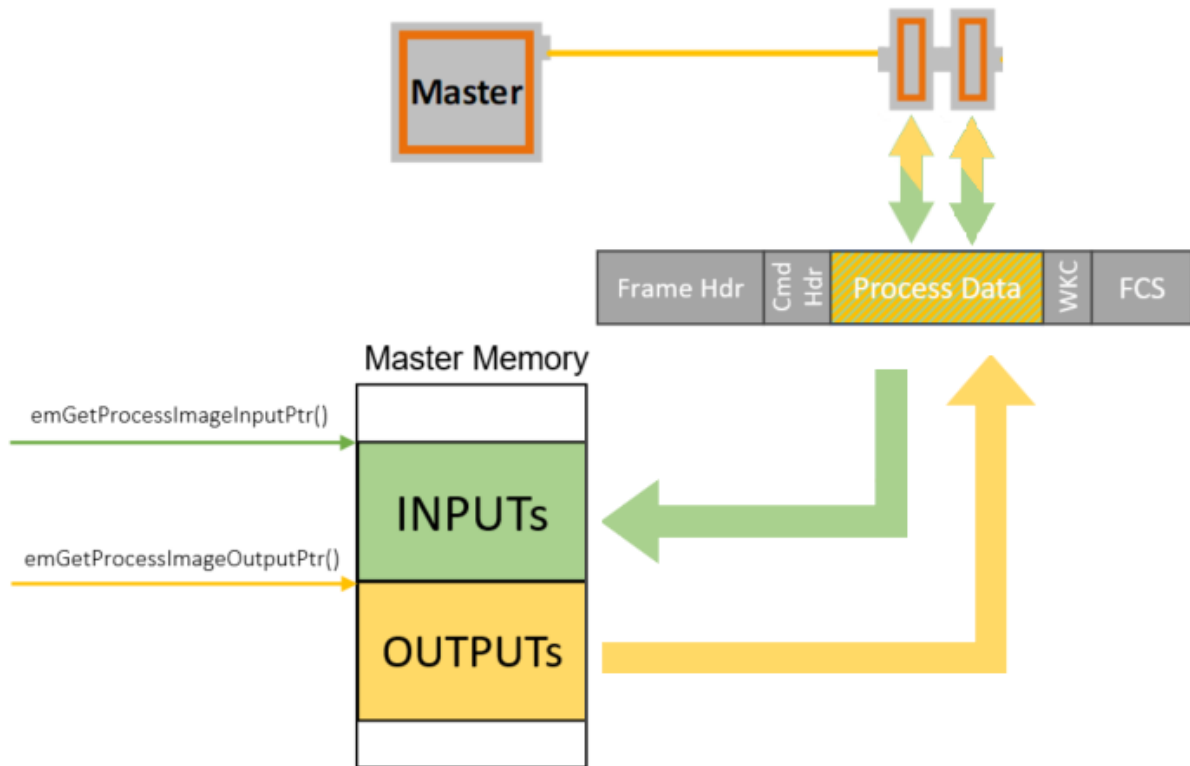
the EC-Master to swap the PDO variable.

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® MainDevice and the SubDevices with EtherCAT® commands every cycle.

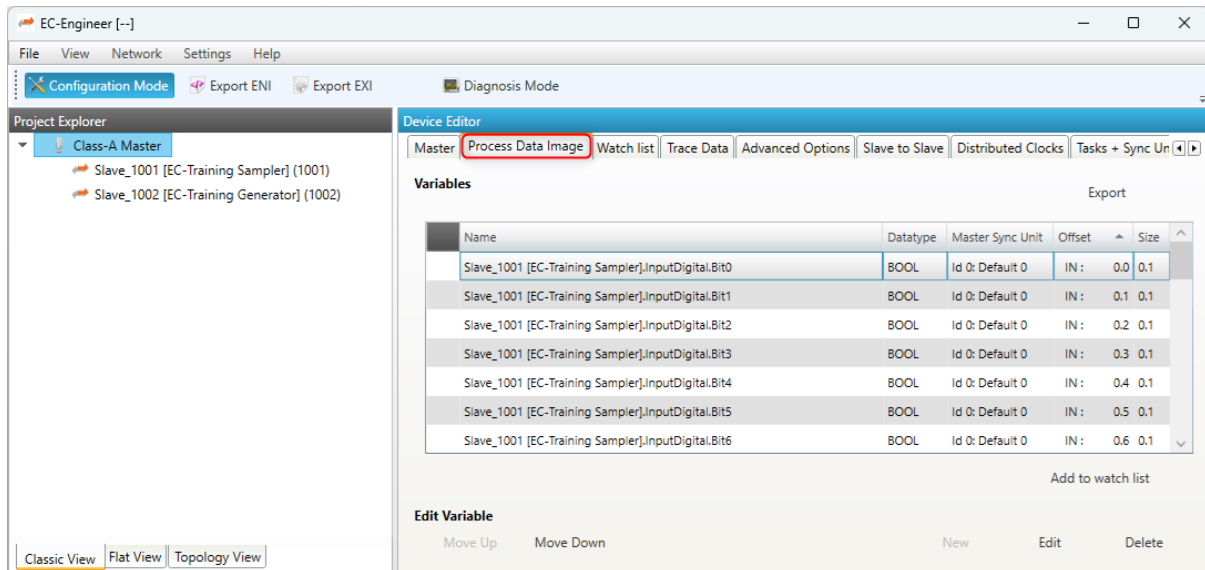
The MainDevice Memory contains the Process Data Image separated in two memory areas, one for input data and another one for output data:



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::dwPDInSize`.

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_BYTE*)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.

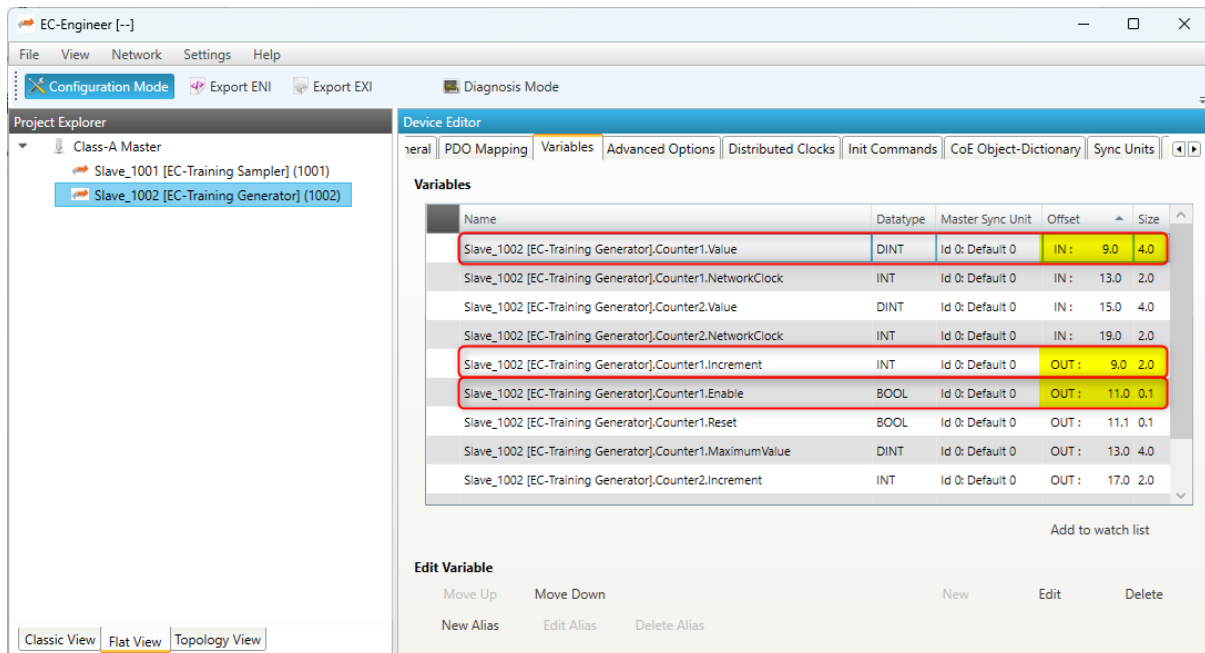
Due to padding bits within the Process Data Objects (PDOs) defined in the EtherCAT® SubDevice description (ESI file) and the calculation algorithms within the configuration tool, process data variables are typically starting at a byte boundary.

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 variable 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 information 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()` are 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 SubDevice 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;
}

```

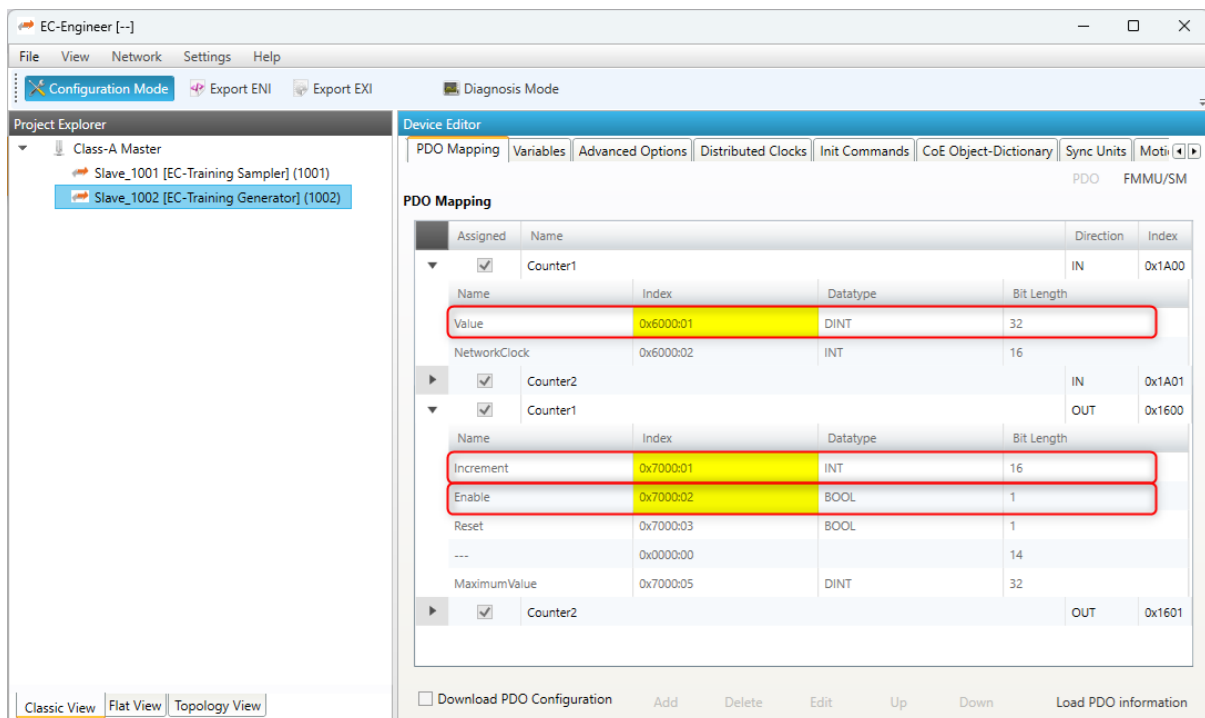
See also:

- *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 screenshot shows the EC-Engineer software interface. The 'PDO Mapping' tab is active, displaying a table of PDO mappings. The table has columns for 'Assigned', 'Name', 'Index', 'Datatype', and 'Bit Length'. The 'Value' variable of Counter1 is highlighted with a red box, showing an index of 0x6000:01 and a datatype of DINT. The 'Increment' and 'Enable' variables of Counter1 are also highlighted with red boxes, showing indices of 0x7000:01 and 0x7000:02, and datatypes of INT and BOOL, respectively.

Assigned	Name	Index	Datatype	Bit Length
<input checked="" type="checkbox"/>	Counter1			
	Name	Index	Datatype	Bit Length
	Value	0x6000:01	DINT	32
	NetworkClock	0x6000:02	INT	16
<input checked="" type="checkbox"/>	Counter2			
<input checked="" type="checkbox"/>	Counter1			
	Name	Index	Datatype	Bit Length
	Increment	0x7000:01	INT	16
	Enable	0x7000:02	BOOL	1
	Reset	0x7000:03	BOOL	1
	---	0x0000:00		14
	MaximumValue	0x7000:05	DINT	32
<input checked="" type="checkbox"/>	Counter2			

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 SubDevice station address

Based on the unique station address of a specific SubDevice 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 SubDevice's process data by its station address from the EC-Master stack using `emGetCfgSlaveInfo ()`.

Process data access using SubDevice 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_BYTE* pbyPdIn = emGetProcessImageInputPtr (dwInstanceId);
    EC_T_BYTE* 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:

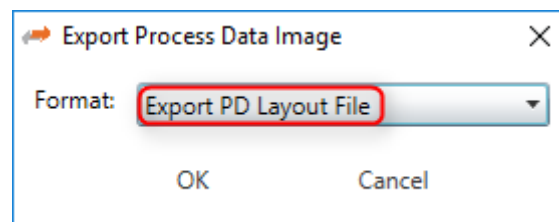
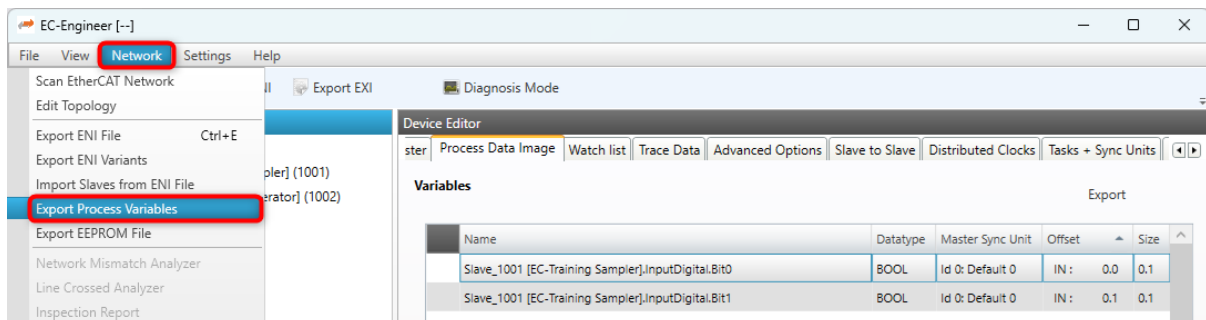
- `emGetCfgSlaveInfo()`

Note: A SubDevice 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 SubDevices 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;

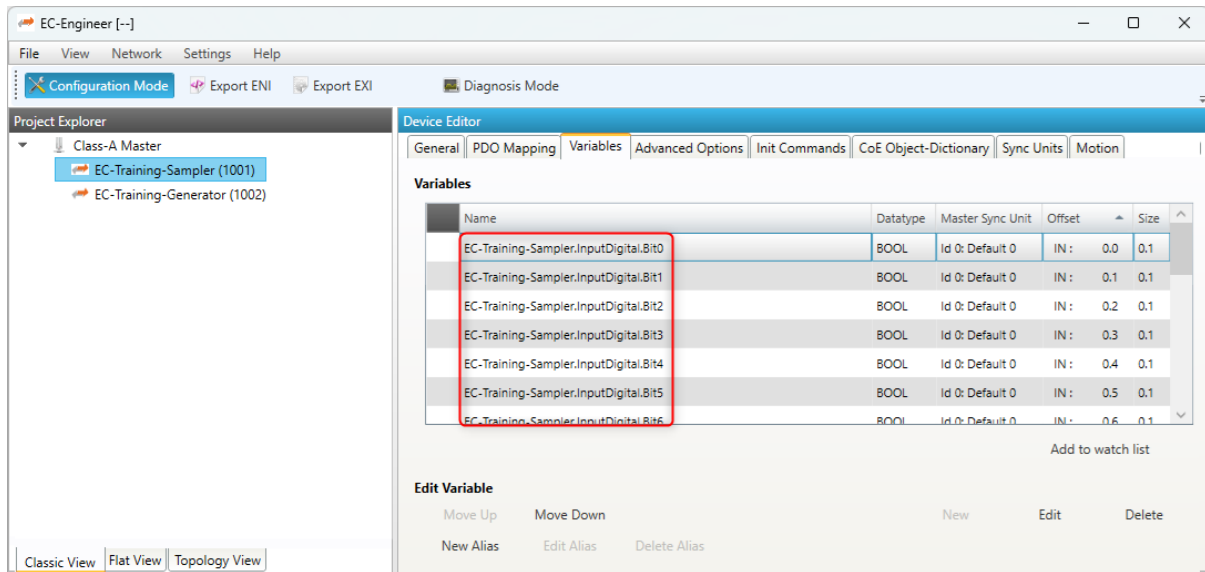
    /* Slave_1002 [EC-Training Generator].Counter1.Increment (Offset: 9.0) */
    EC_SET_FRM_WORD(&poPdOut->swSlave_1002_Counter1_Increment, 100);

    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 SubDevices 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 SubDevices 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 SubDevices

```
#include "pdlayoutcustom.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);
    ↪
    ↪
    /* 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)


```

EC_T_LINK_VAR_INFO LinkVarInfo;
LinkVarInfo.pvContext = EC_NULL;
LinkVarInfo.pbyApplVar = (EC_T_BYTE*) &S_bSlaveChannel3Out;
LinkVarInfo.wApplVarBitSize = sizeof(S_bSlaveChannel3Out) * 8;
LinkVarInfo.dwApplVarBitOffs = 0;
LinkVarInfo.pfnConverter = EC_NULL;
/* link application variable to process output */
dwRes = emLinkOutputVarByName(dwInstanceId, "Slave_1004 [EL2004].Channel 3.Output",
↪ &LinkVarInfo);

```

For more complicated tasks it is possible to specify a callback function which is called after the input process image was updated resp. before the output process image is sent. The first argument will provide all needed information about the application and the process variable. The second parameter is a pointer to the input resp. output process image. Then the callback has to perform the transition from process to application variable or vice versa and additionally can make further instructions like e.g. a conversion.

Example of callback function inverting output before writing to process image:

```

EC_T_VOID InvertOutput(const EC_T_LINK_VAR_INFO* pApplVarInfo, EC_T_BYTE*
↪ pbyProcessImage)
{
    EC_T_BYTE byOut = *pApplVarInfo->pbyApplVar;
    byOut ^= 0x01;
    EC_SETBITS(pbyProcessImage, &byOut,
        pApplVarInfo->pProcVarInfoEx->nBitOffs, pApplVarInfo->pProcVarInfoEx->
↪ nBitSize);
}

```

Example of how “Slave_1004 [EL2004].Channel 3.Output” can be linked to an application variable with callback function:

```

EC_T_LINK_VAR_INFO LinkVarInfo;
LinkVarInfo.pvContext = EC_NULL;
LinkVarInfo.pbyApplVar = (EC_T_BYTE*)&S_bSlaveChannel3Out;
LinkVarInfo.wApplVarBitSize = sizeof(S_bSlaveChannel3Out) * 8;
LinkVarInfo.dwApplVarBitOffs = 0;
LinkVarInfo.pfnConverter = InvertOutput;
/* link application variable to process output */
dwRes = emLinkOutputVarByObject(dwInstanceId, EC_TRUE, 1004, 0x7010, 0x01, &
↪ LinkVarInfo);

```

Hint: To link an application variable to a process variable the EC-Master must be within the state INIT resp. PREOP.

Hint: Depending on the main task cycle time linking application variables can have an impact to the performance of the EtherCAT® job task.

Hint: As soon as an application variable has been linked it should stay accessible until a call to *emDeinitMaster()* or *emConfigureNetwork()*.

3.6 Process Data Memory

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

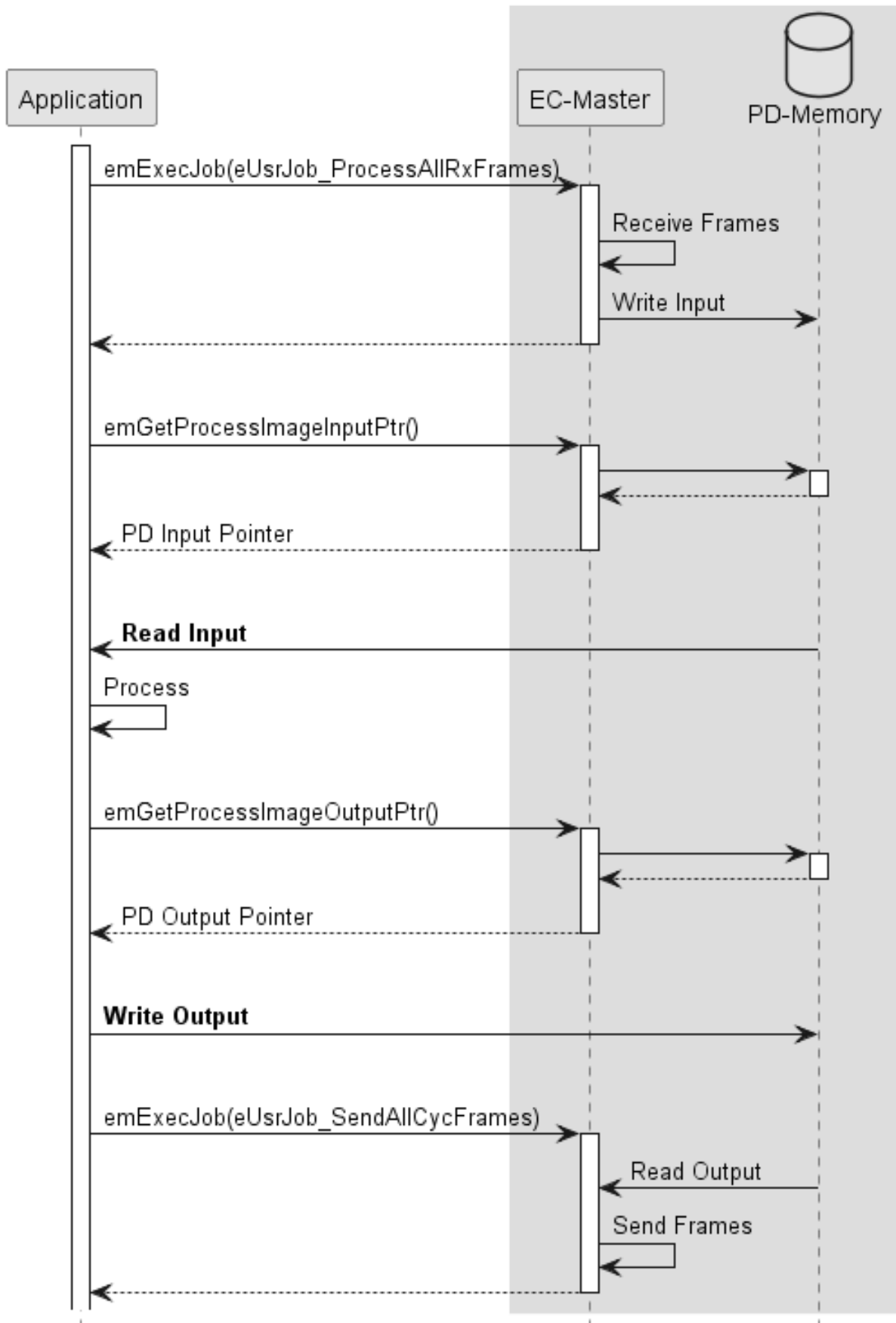
The EC-Master uses two separate buffers where process data input values and process data output values are stored. The buffers used may either be always the same (fixed buffers) or be changed on every process data transfer cycle (dynamic buffers).

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

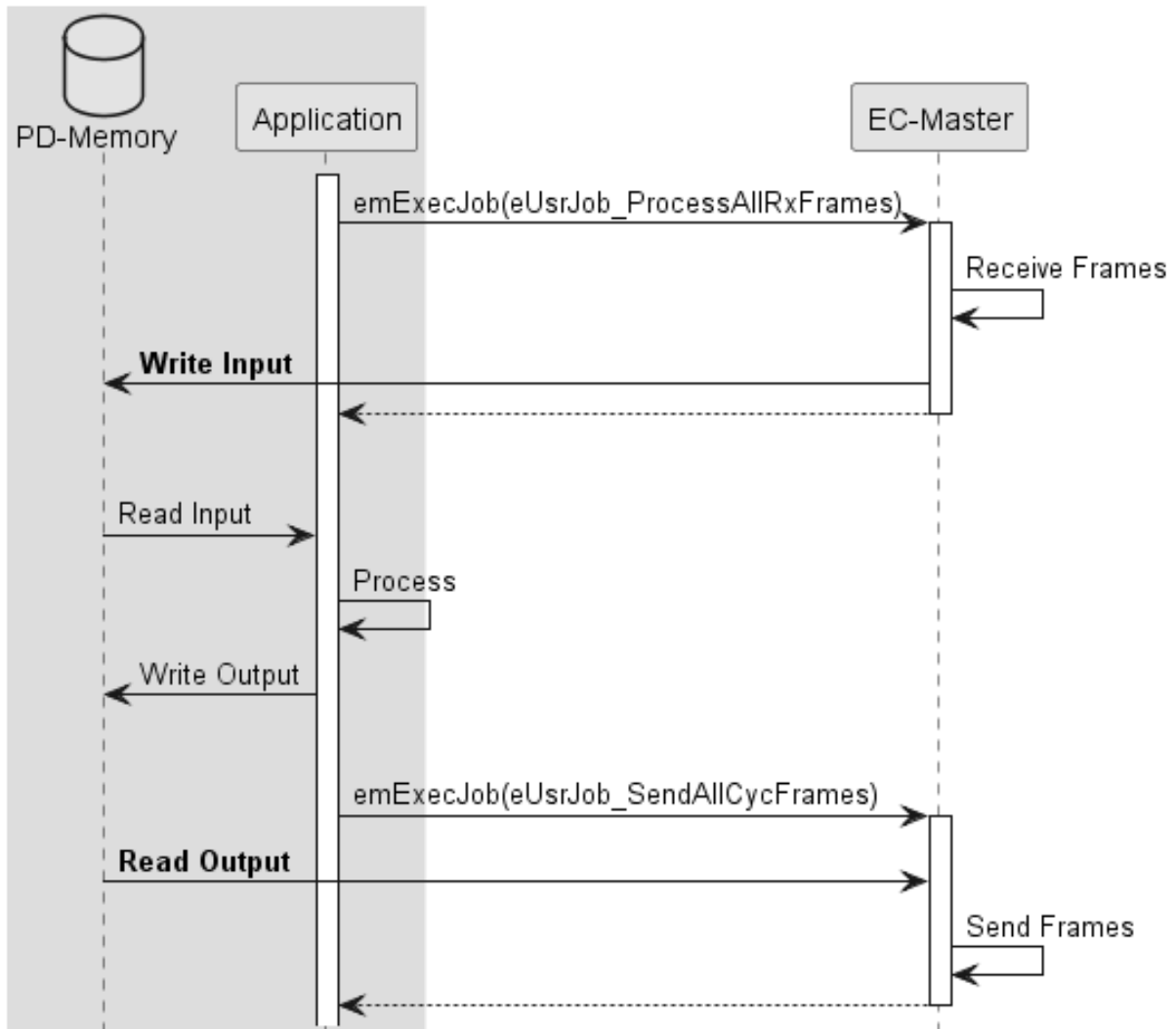
3.6.1 EC-Master as process data memory provider

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 `emIoCtl - EC_IOCTL_REGISTER_PDMEMORYPROVIDER` in case the EC-Master shall use externally allocated memory to store input and output process data values.



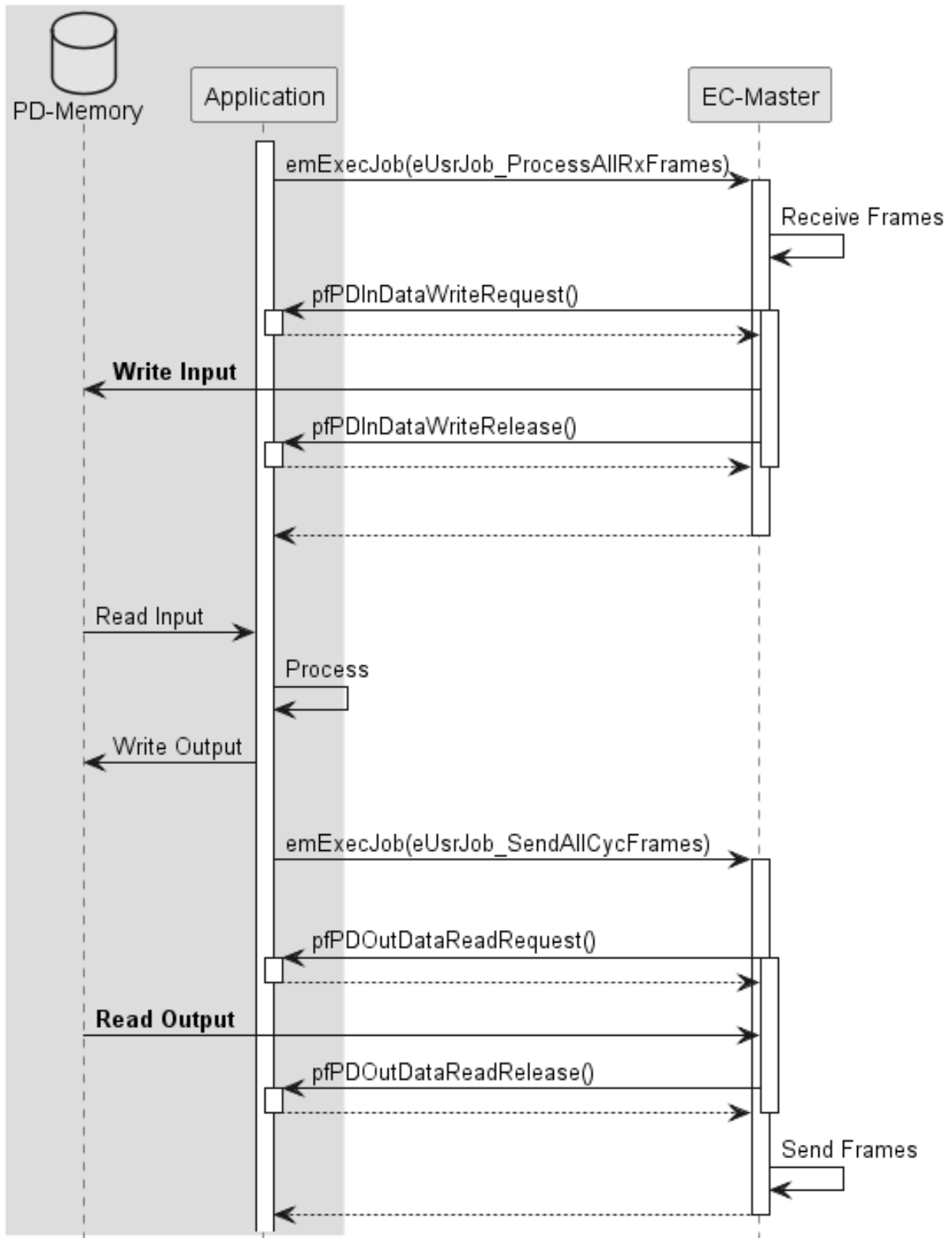
The memory provider may optionally supply callback functions to synchronize memory access between the application and the EC-Master.

Receiving new input process data:

- `EC_T_MEMPROV_DESC::pfPDInDataWriteRequest`
- `EC_T_MEMPROV_DESC::pfPDInDataWriteRelease`

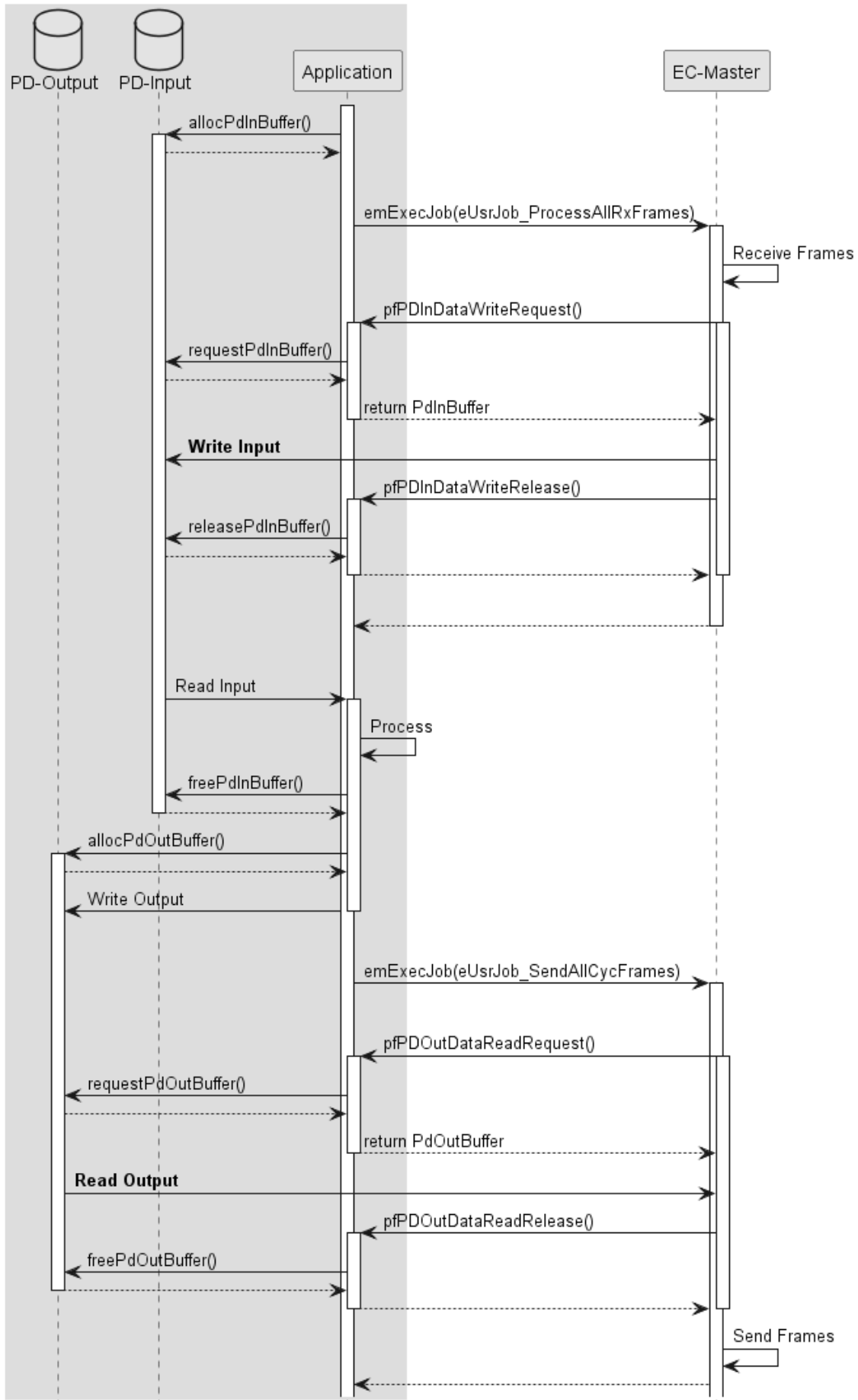
Sending new output process data:

- `EC_T_MEMPROV_DESC::pfPDOutDataReadRequest`
- `EC_T_MEMPROV_DESC::pfPDOutDataReadRelease`



3.6.3 Application as process data memory provider with dynamic buffers

The application registers an external memory provider without fixed buffers via *emIoCtl* - *EC_IOCTL_REGISTER_PDMEMORYPROVIDER* with the parameters *EC_T_MEMPROV_DESC::pbyPDInData* and *EC_T_MEMPROV_DESC::pbyPDOutData* 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 functions 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 EC-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 and Frame Loss

New input values received from the SubDevices 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

See also:

Working Counter (WKC) State in Diagnosis Image

It is possible to clear INPUTs in the process data image if an error is detected using the following IOCTLS:

See also:

emIoctl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ERROR

See also:

emIoctl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ZERO

See also:

emIoctl - EC_IOCTL_SET_ZERO_INPUTS_ON_FRAME_LOSS

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 MainDevice 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.wkcStateDiagOffsOut[0],
        ↪ 1);
    }
}

```

(continues on next page)

(continued from previous page)

```

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

See also:

Cyclic cmd WKC validation and Frame Loss

Behavior in case of automatically adjusted expected WKC value

Optionally, the expected WKC value can be automatically adjusted according to the state and the presence of the Sub-Devices. See *emIoctl - EC_IOCTL_SET_AUTO_ADJUST_CYCCMD_WKC_ENABLED*. The WKC State bits change synchronized to the corresponding notification, e.g. on link disconnection all SubDevices disappear and the behavior is as follows:

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

See also:

- *emNotify - EC_NOTIFY_ETH_LINK_NOT_CONNECTED*
- *emNotify - EC_NOTIFY_SLAVE_PRESENCE*

3.7.3 MainDevice 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)

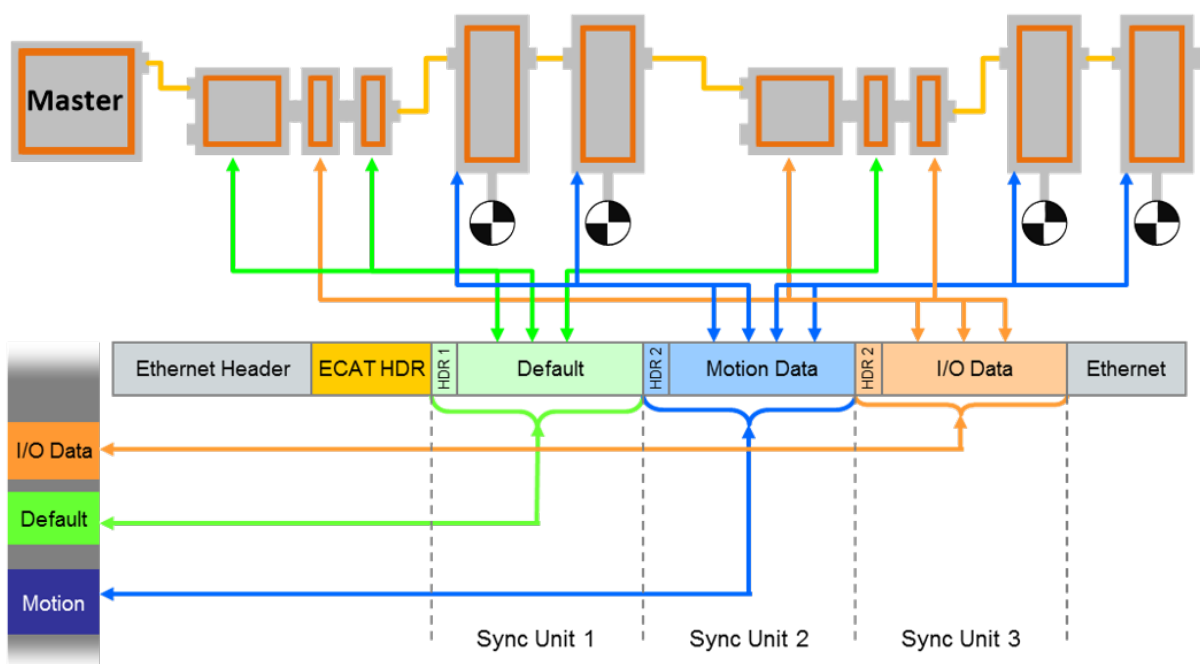


Figure out the MSU offsets by calling the functions `emGetMasterSyncUnitInfoNumOf()` and `emGetMasterSyncUnitInfo()`, as described in the corresponding documentation.

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));
}
```

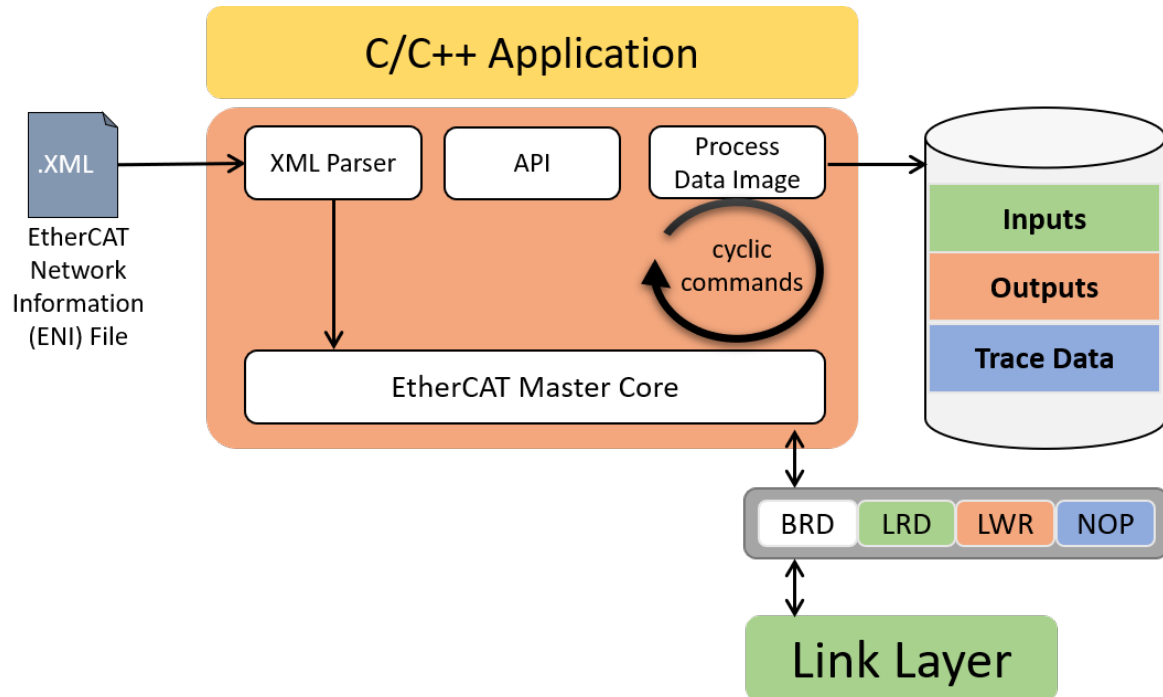
```
/* setup callback function to log EtherCAT network traffic */
EC_T_VOID* pvMyAppContext = this;
dwRes = emLogFrameEnable(dwInstanceId, LogFrameHandler, pvMyAppContext);
```

```
/* 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 SubDevice 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.

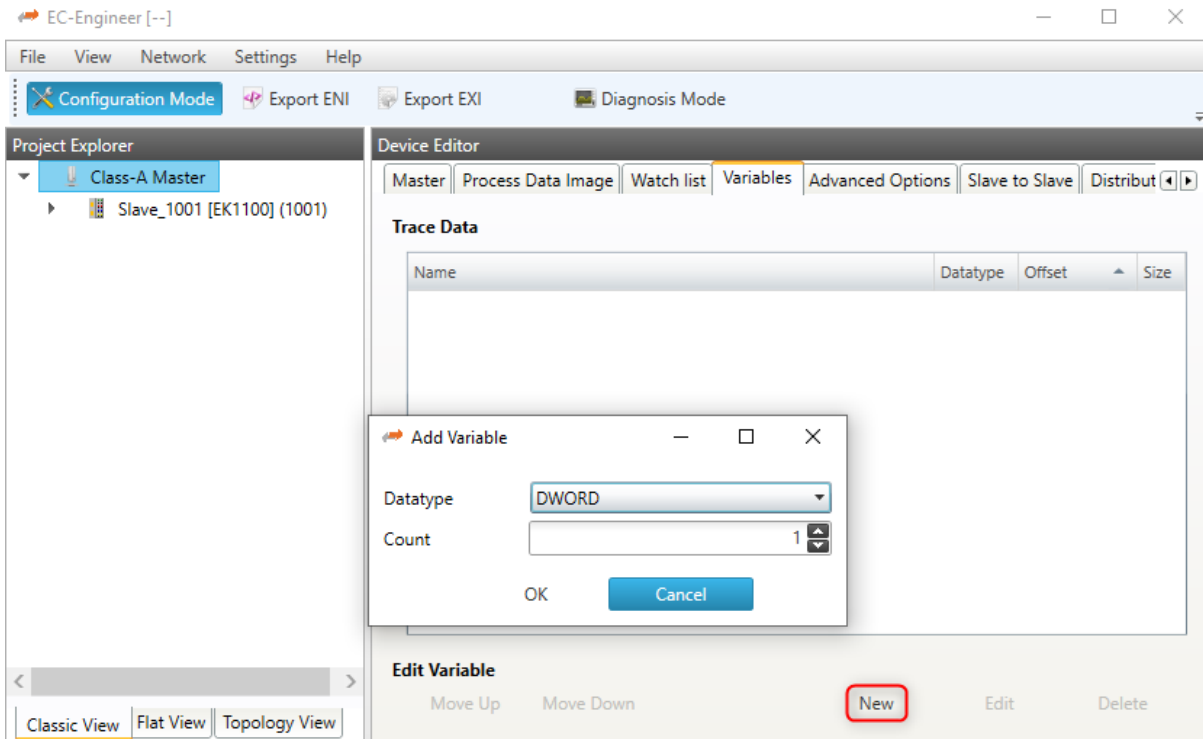


To transfer the data, an additional NOP cmd is appended to the end of the cyclic EtherCAT® frame. The NOP cmd has ADP 0 and ADO 0x4154. The EC-Master automatically fills the data area of the NOP Cmd with the current trace data when sending cyclic frames. Since the trace data are transferred to the network 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.

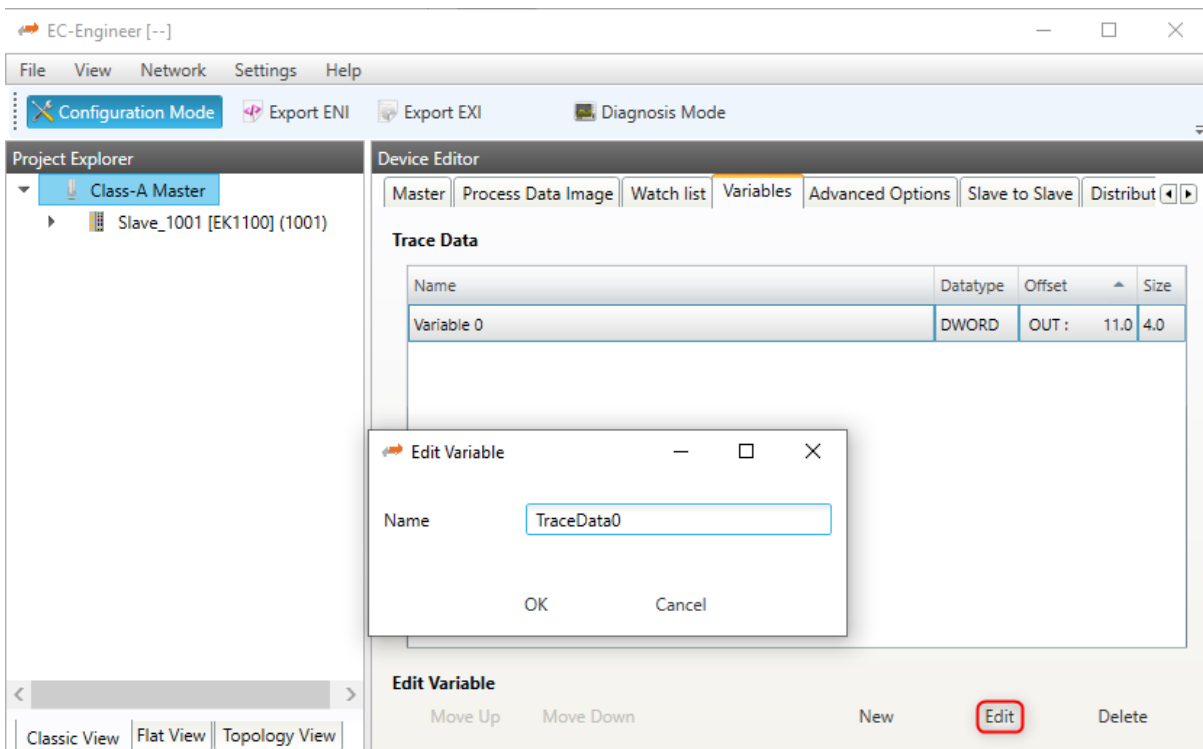
3.9.1 Trace Data configuration via EC-Engineer

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:

<ul style="list-style-type: none"> [-] ProcessImage <ul style="list-style-type: none"> [+] Inputs [-] Outputs <ul style="list-style-type: none"> [+] ByteSize [+] Variable [+] Variable [-] Variable <ul style="list-style-type: none"> [+] Name [+] DataType [+] BitSize [+] BitOffs 	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="background-color: #e0f0ff;"> </td></tr> <tr><td style="background-color: #e0f0ff;">15</td></tr> <tr><td style="background-color: #e0f0ff;"> </td></tr> <tr><td style="background-color: #e0f0ff;">TraceData0</td></tr> <tr><td style="background-color: #e0f0ff;">DWORD</td></tr> <tr><td style="background-color: #e0f0ff;">32</td></tr> <tr><td style="background-color: #e0f0ff;">88</td></tr> </table>		15		TraceData0	DWORD	32	88
15								
TraceData0								
DWORD								
32								
88								

As well as the NOP cmd:

<ul style="list-style-type: none"> [-] Cyclic <ul style="list-style-type: none"> [+] CycleTime [-] Frame <ul style="list-style-type: none"> [+] Cmd [+] Cmd [-] Cmd <ul style="list-style-type: none"> [+] State [+] State [+] State [+] Comment [+] Cmd [+] Adp [+] Ado [+] DataLength [+] Cnt [+] InputOffs [+] OutputOffs 	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="background-color: #e0f0ff;">1000</td></tr> <tr><td style="background-color: #e0f0ff;"> </td></tr> <tr><td style="background-color: #e0f0ff;"> </td></tr> <tr><td style="background-color: #e0f0ff;">PREOP</td></tr> <tr><td style="background-color: #e0f0ff;">SAFEOP</td></tr> <tr><td style="background-color: #e0f0ff;">OP</td></tr> <tr><td style="background-color: #e0f0ff;">PD; NOP; Trace Data</td></tr> <tr><td style="background-color: #e0f0ff;">0</td></tr> <tr><td style="background-color: #e0f0ff;">0</td></tr> <tr><td style="background-color: #e0f0ff;">16724</td></tr> <tr><td style="background-color: #e0f0ff;">4</td></tr> <tr><td style="background-color: #e0f0ff;">0</td></tr> <tr><td style="background-color: #e0f0ff;">11</td></tr> <tr><td style="background-color: #e0f0ff;">11</td></tr> </table>	1000			PREOP	SAFEOP	OP	PD; NOP; Trace Data	0	0	16724	4	0	11	11
1000															
PREOP															
SAFEOP															
OP															
PD; NOP; Trace Data															
0															
0															
16724															
4															
0															
11															
11															

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 EC-Master MainDevice Stack Source Code

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

The MainDevice 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 MainDevice 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 MainDevice 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 module 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. MainDevice 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 `emllXxxx.dll` where `Xxxx` shall be replaced by the Real-time Ethernet Driver type (e.g. `emllI8255x.dll` for the I8255x Real-time Ethernet Driver).
- VxWorks: a downloadable kernel module (.out) has to be created. The name of the module has to be `emllXxxx.out` where `Xxxx` shall be replaced by the Real-time Ethernet Driver type (e.g. `emllI8255x.out` for the I8255x Real-time Ethernet Driver). `sysLoSalAdd.c` should be included in the 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:** an RTX dynamic link library (`.rt.dll`) has to be created. The name of the DLL has to be `emllXXXX.dll` where `XXXX` shall be replaced by the Real-time Ethernet Driver type (e.g. `emllI8255x.dll` for the I8255x Real-time Ethernet Driver).
- **INtime:** a shared library (`.rsl`) has to be created. The name of the RSL has to be `emllXXXX.rsl` where `XXXX` shall be replaced by the Real-time Ethernet Driver type (e.g. `emllI8255x.rsl` for the I8255x Real-time Ethernet Driver).

MainDevice Stack Binaries

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

- MainDevice 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 MainDevice 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

```
emIoctl - EC_IOCTL_SET_FRAME_LOSS_SIMULATION  
emIoctl - EC_IOCTL_SET_RXFRAME_LOSS_SIMULATION  
emIoctl - 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

```
emIoctl - EC_IOCTL_REGISTER_PDMEMORYPROVIDER
```

EXCLUDE_MULTIPLE_CYC_ENTRIES

```
emIoctl - 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 performance or to reduce the footprint. They are defined incrementally as follows. 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 SubDevices 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 the file `ENI.c` by the generated one. The 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
emllCmsisEth.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 or download from 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 the demo application the following environment variables (either system or project variables) have 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® SubDevice(s).
- Power on EtherCAT® SubDevice(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 an 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 built 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 options	
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 another example with different parameters, the kernel has to be rebuilt.

For the EcMasterDemo example the 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] MainDevice 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  
libemllIntelGbe.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>/libsrc/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 %BUILDOUTPUT%\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/libsrc/
↳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. Or MCU-PLUS-SDK-AM64X 11.00.00.15 and Code Composer Studio 12.8 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

In 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. Or MCU-PLUS-SDK-AM243X 11.00.00.15 and Code Composer Studio 12.8 or newer.

How to create the demo applications for 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 with MCU-PLUS-SDK-AM243X 08.01.00.36

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 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.

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_mmcscd example

4.4 tenAsys INtime

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

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

4.4.1 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 the 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 an 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 call-backs, or network adapter IRQs. To fully isolate a CPU for real-time workloads, `nohz_full`, `rcu_nocbs`, and `irqaffinity` should be used together to eliminate kernel noise and ensure deterministic execution.

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

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

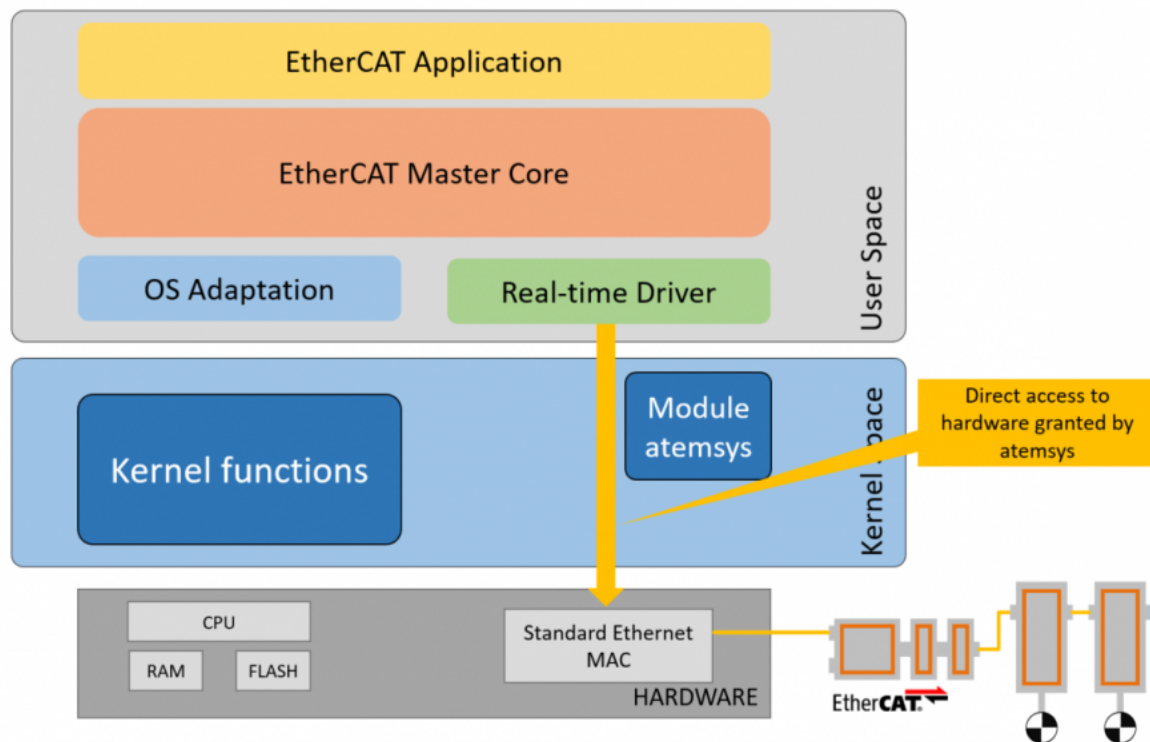
Running EcMasterDemo on the isolated CPU core by setting the CPU affinity -a:

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

4.5.2 atemsys kernel module

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

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



atemsys as Device Tree Ethernet Driver

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

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

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

atemsys and PHY OS Driver

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

4.5.3 Unbind Ethernet Driver instance

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

Unbind from kernel driver

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

PCI

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

Example:

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

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

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

SoC

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

Example:

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

Unload kernel driver

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

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

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

Example:

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

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

```
$ lspci -v
```

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

Modules can be prevented from loading with the following commands:

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

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

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

4.5.4 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 and 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 eni.xml into this directory.
3. **Start bash console in container**

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

Command line arguments:

- `-it` Allocate a pseudo-TTY and run container
- `--name atem_container` Container name
- `--device=/dev/atemsys:/dev/atemsys` Add *atemsys* device to container
- `--ulimit rtprio=99` Allow max realtime priority
- `--cap-add=sys_nice` Add Linux capability to set priority
- `--cap-add=ipc_lock` Add Linux capability to lock memory

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

4. Run 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-eabi` (ARM 32Bit)
- `armv7-vfp-eabi` (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-eabi`/`armv7-vfp-eabi` 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:

```
-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 on changing the priority range refer to the QNX documentation.

Attention: Not changing the priority range leads to poor 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 a predefined typed memory region that is used by the Real-time Ethernet Driver. The definition has to be done in the QNX BSP build file and the name must match following pattern:

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

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

```
smm_emllIntelGbe-0x00000001
```

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

4.7.4 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 application 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 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

4.8.1 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 interface may not be configured in RtxTcpIp.ini.

4.8.2 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

4.8.3 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)
```

4.9 SylixOS

4.9.1 Setting up and running EcMasterDemo on SylixOS

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 ...
```

4.10 TI-RTOS

4.10.1 Setting up and running EcMasterDemo

Prerequisites, basic settings:

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

Make sure your Code Composer Studio uses the 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. E.g.: For AM572x demo project, PDK_INSTALL_PATH=C:/ti/pdk_am57xx_1_0_9/packages

TI SDK RTOS for AM654x

At least 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 the 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 a 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
- emllICSS
- 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 pdkAppImageCreate.bat from the PSDK package.

How to run the EC-Master motion demo application

1. Create an appropriate ENI file as an 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 the target processor.
5. The demo logging is done over UART.

See also:

Running EcMasterDemo

4.10.2 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
```

4.11 μ C3 for STM32

4.11.1 Setting up and running EcMasterDemo in IAR for ARM IDE

1. Prerequisites

- IAR for ARM IDE, v9.32
 - μ C3/Compact
 - STM32H743 target
 - EtherCAT® devices
2. Connect the STM32H743 target to the PC.
 3. Connect EtherCAT® device 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 the file ENI.c by the generated one. The 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

4.11.2 OS Compiler settings

Besides the general settings from *Compiling the EcMasterDemo* the following settings are necessary to build the example application for μ C3 (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
emllCmsisEth.a
```

4.12 μ C3 for i.MX8

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

1. Prerequisites

- GCC compiler for aarch64 9.2.1
 - GNU make 4.4.1
 - μ C3/Standard for Cortex-A53 MPcore i.MX8M Plus GCC
 - NXP 8MPLUSLPD4-EVK board
 - EtherCAT® network adapters
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® network adapters to the board, use "ENET1" connector.

9. Prepare an u-boot SD card and copy Workspace\uC3_iMX8\EcMasterDemo_8MPLUSPD4-EVK\ aarch64\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
```

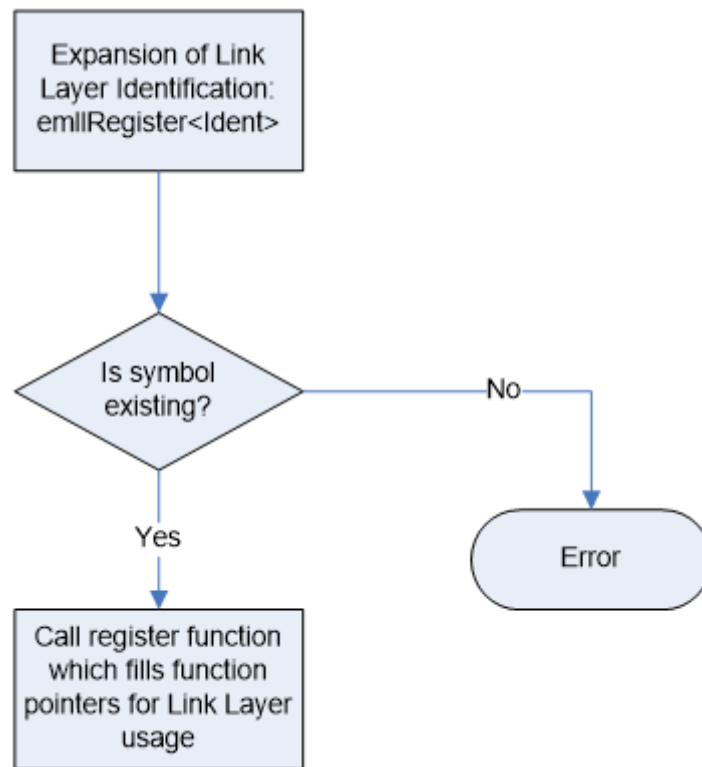
12. Start EcMasterDemo:

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

4.13 Windriver VxWorks

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

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



4.13.1 VxWorks native

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

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

```

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

```

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

See also:

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

4.13.2 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.

4.13.3 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® SubDevices

The SubDevices 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 a 100Mbit/s (!) Ethernet switch has to be used.

4. Download a Real-time Ethernet Driver module

The Real-time Ethernet Driver library (e.g. `emllIntelGbe.out`) which contains hardware support for the corresponding NIC must be downloaded. By default the Ethernet Driver `emllSnarfGpp` are contained in 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 an FTP server connection on host

The demo application needs to load an XML file (`eni.xml`) for the configuration of the EC-Master. This file can be accessed using an FTP server. The screen shot below shows 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<eml1i8254x.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

4.13.4 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.

4.14 Microsoft Windows

4.14.1 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

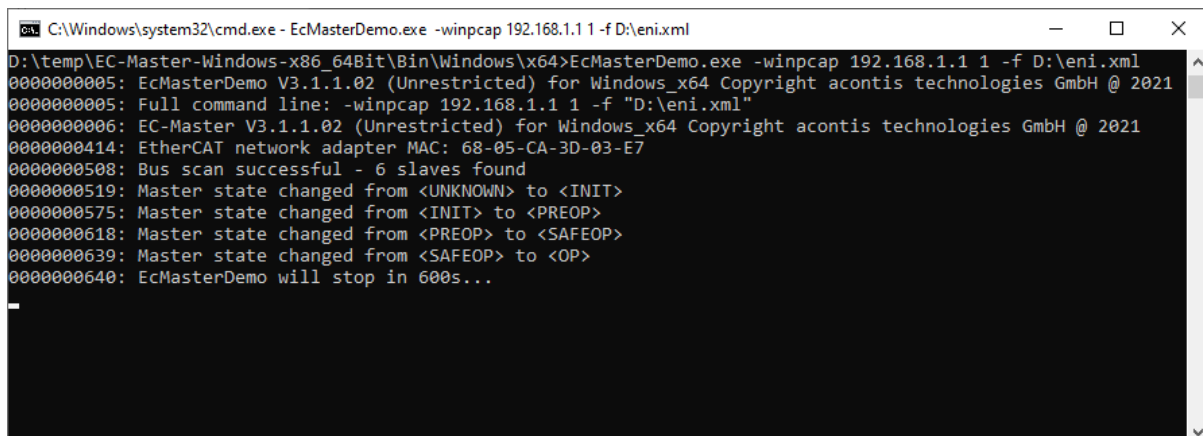
Any EtherCAT® module can be directly connected to the target system. EtherCAT® requires a 100 Mbit/s connection. If the Ethernet adapter card does not support this speed, an Ethernet switch must be used.

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
00000000414: 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.

4.14.2 EcMasterDemoDotNet (.NET) - Microsoft Windows

Install and run

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

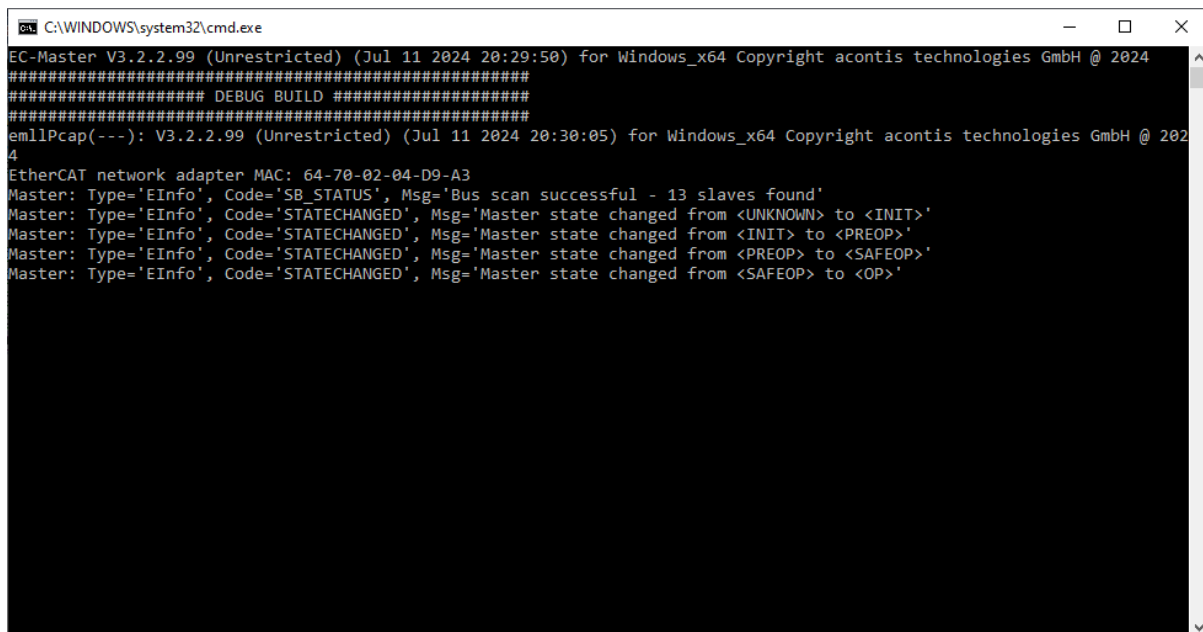
Any EtherCAT® module can be directly connected to the target system. EtherCAT® requires a 100 Mbit/s connection. If the Ethernet adapter card does not support this speed, an Ethernet switch must be used.

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 #####
emllPcap(---): 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>'
```

Compile and debug

1. Copy `*.dlls` from `<InstallPath>/Bin/Windows/<Arch>` to `<InstallPath>/Bin/Windows/<Arch>/Debug` and `<InstallPath>/Bin/Windows/<Arch>/Release`.

Hint: By copying the DLLs, they can be loaded on startup and original files are not overwritten and are retained.

1. **Open** `<InstallPath>/Workspace/WindowsVS2015/EcMasterDemoDotNet/EcMasterDemoDotNet.csproj` in VS2015 or higher
2. Specify command line arguments for the project as shown above
3. (Adapt code,) compile, run, debug as usual

4.14.3 EcMasterDemoGuiDotNet (.NET) - Microsoft Windows

1. Prerequisites

To run the `EcMasterDemoGuiDotNet.exe`, the libraries `EcMaster.dll`, `EcMasterRasServer.dll`, `EcWrapperDotNet.dll`, `EcWrapper.dll` and `emllNdis.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

4.14.4 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

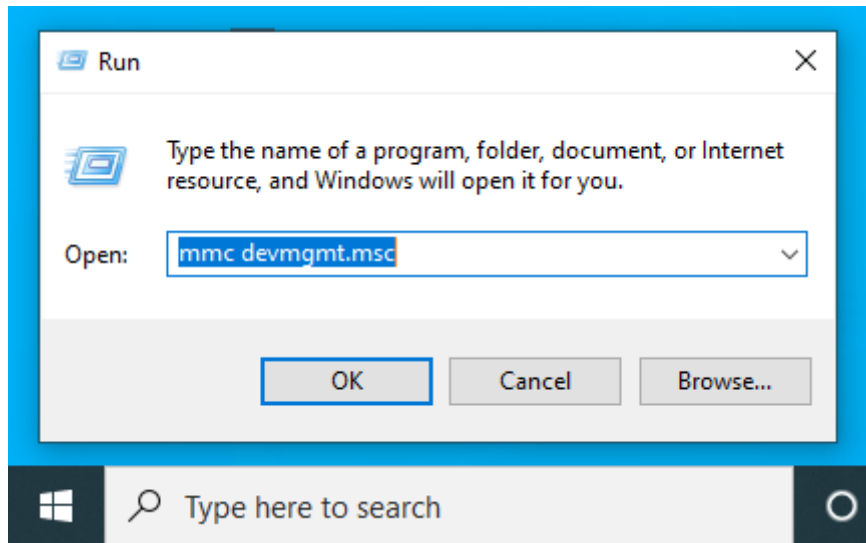
```
<InstallPath>\SDK\LIB\Windows
```

4.14.5 RtaccDevice for Real-time Ethernet Driver

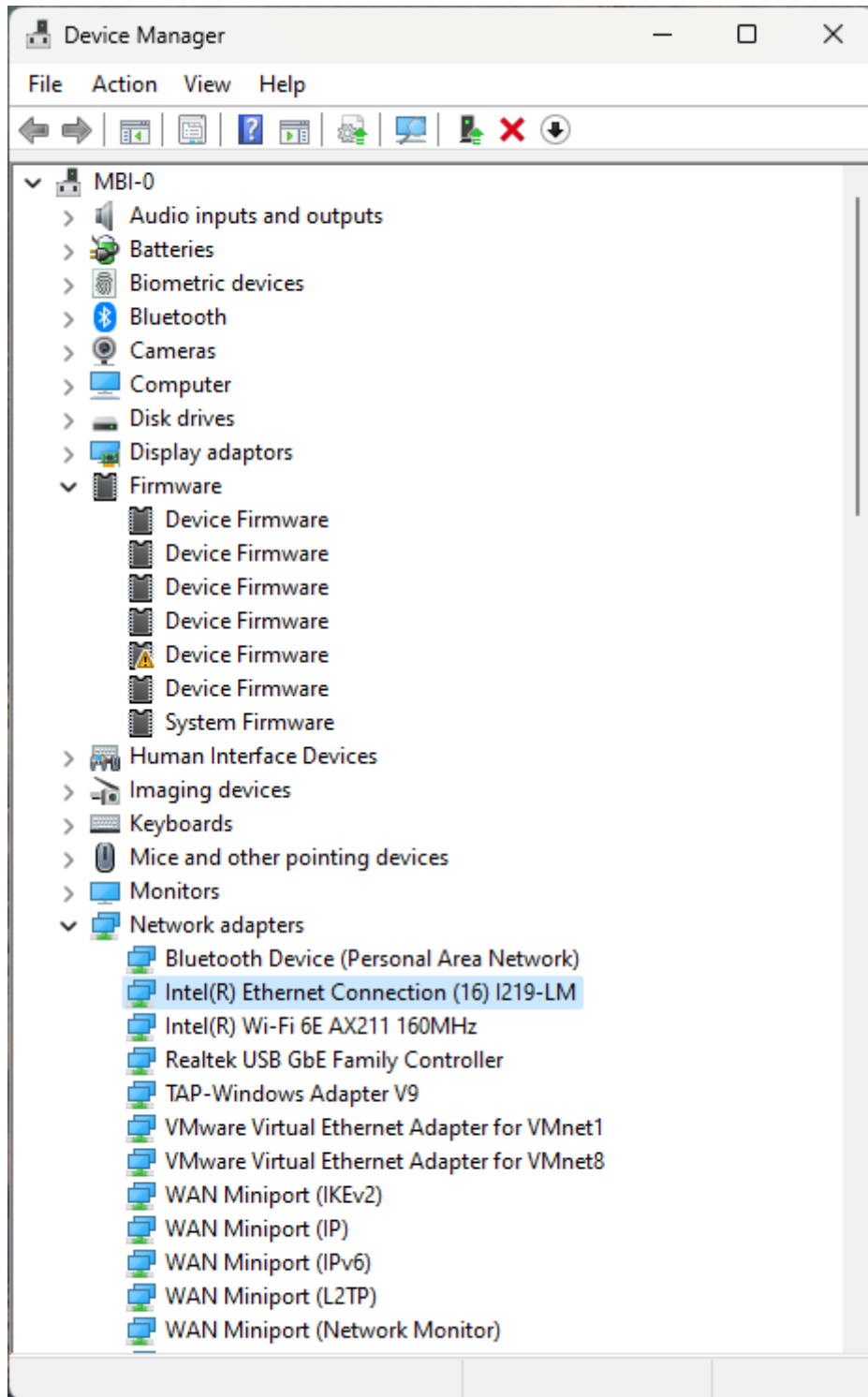
As alternative to the NDIS based or Pcap based Ethernet Driver, an optional Real-time Ethernet Driver on Windows can be installed. The Real-time Ethernet Driver replaces the original Windows driver and also requires an extra license.

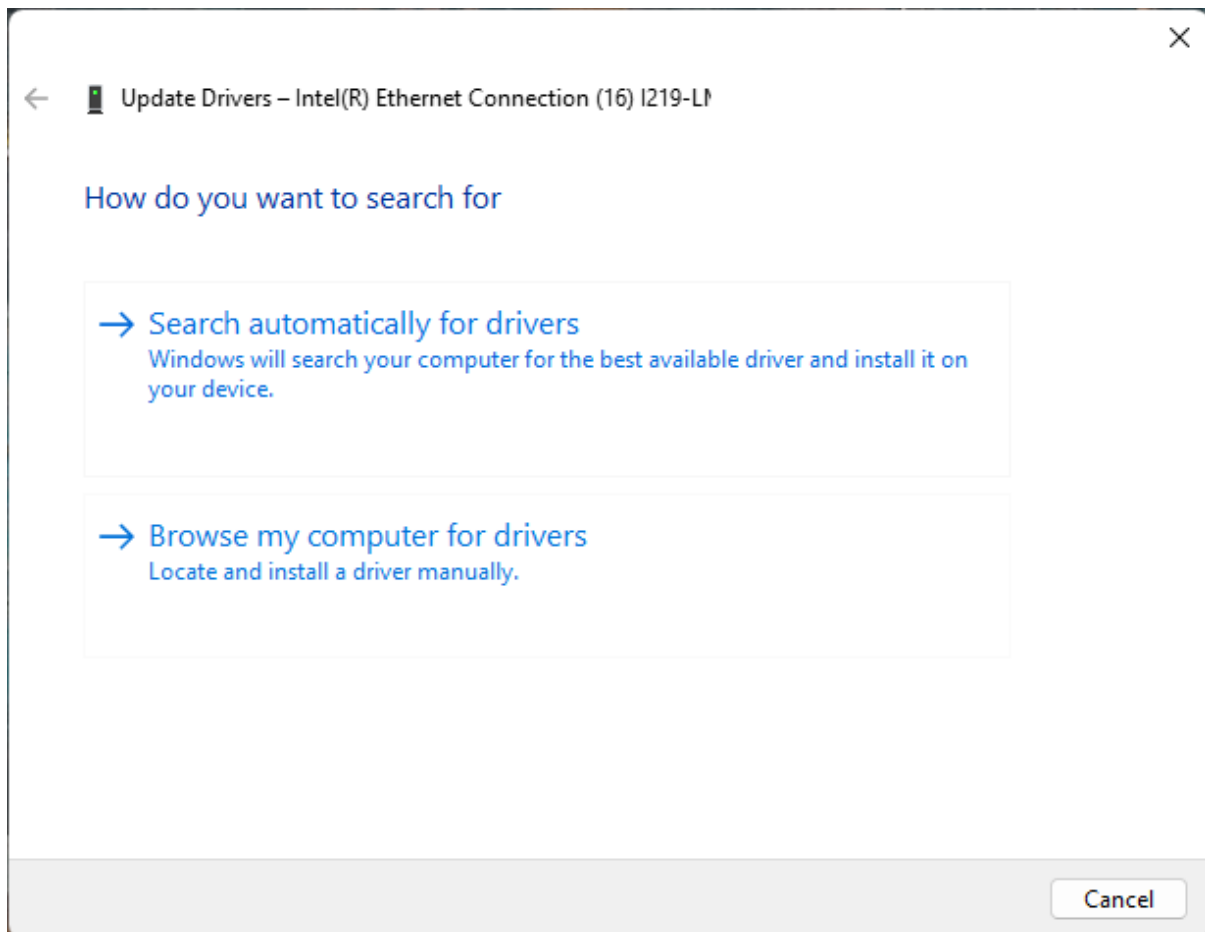
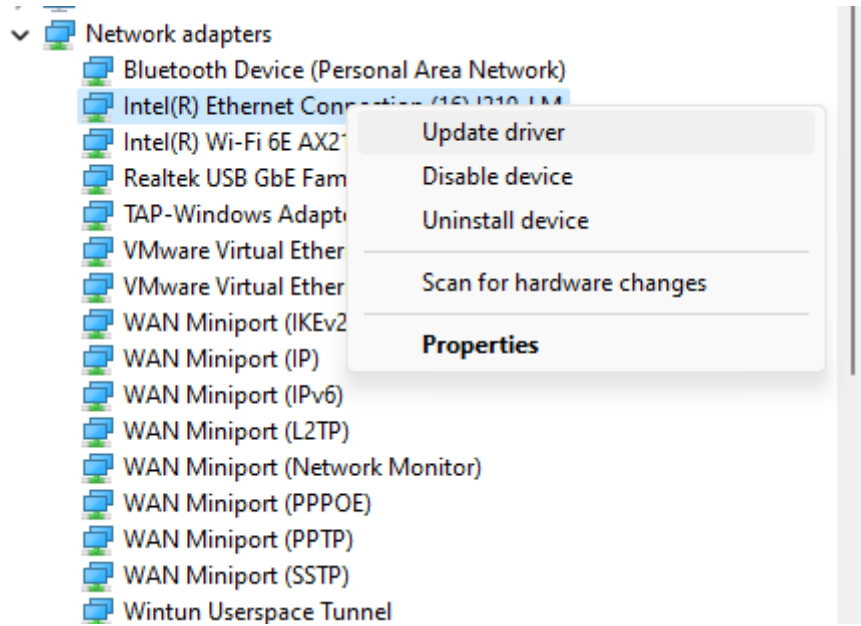
To use the Real-time Ethernet Driver under Windows, it is necessary to install the `RtaccDevice` driver included in the Real-time Ethernet Driver delivery:

1. Start the “Device Manager”

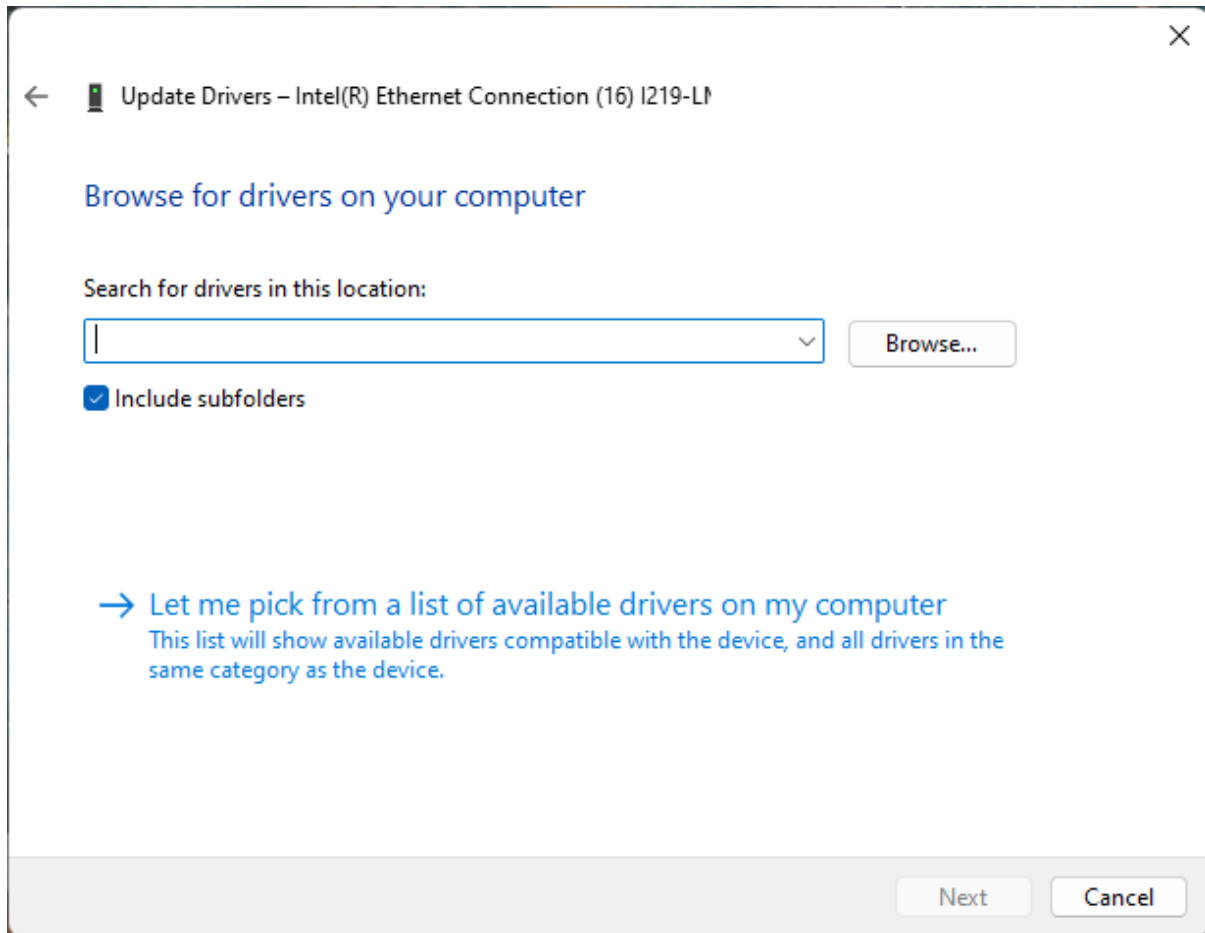


2. Assign RtaccDevice to the network adapter

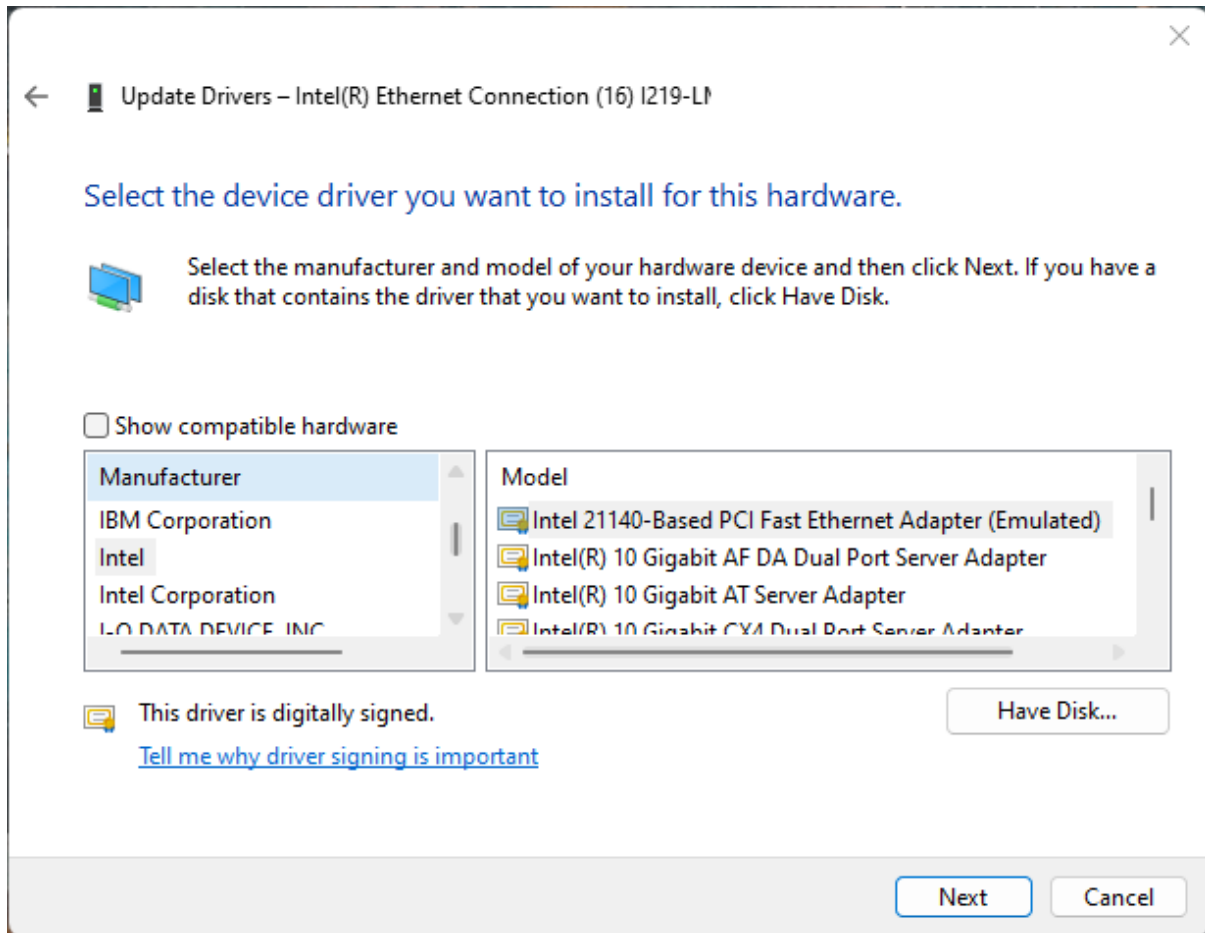




Click on “Browse my computer for drivers”



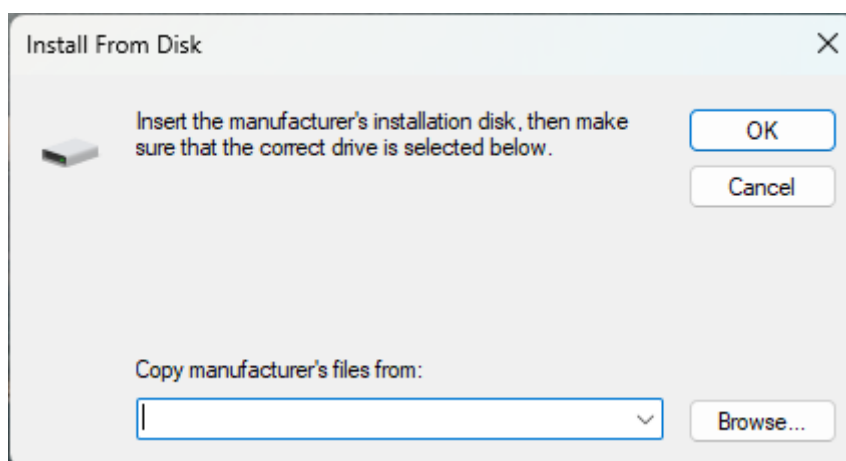
Click on “Let me pick...”



Click on “Have Disk...”

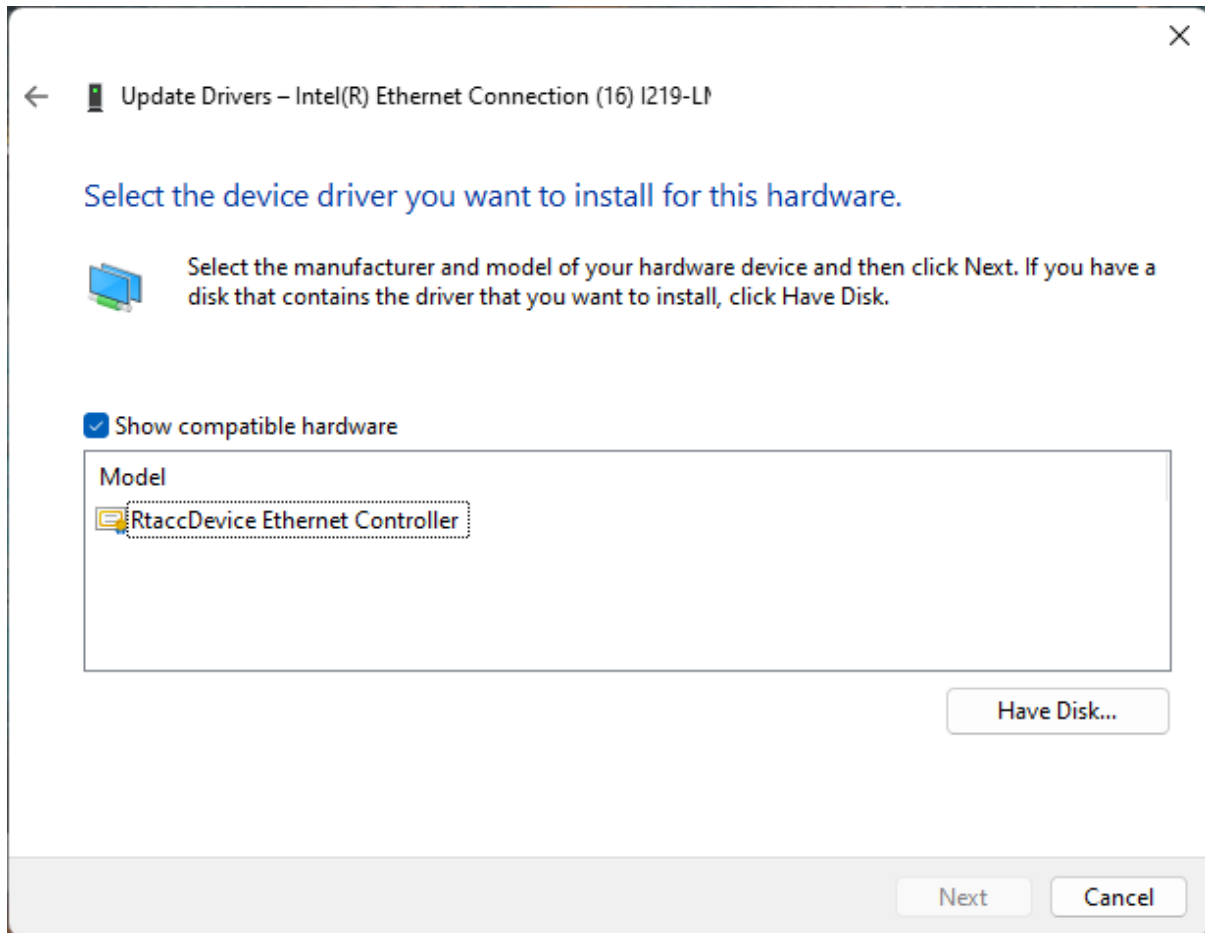
3. Enter the directory of RtaccDevice Driver

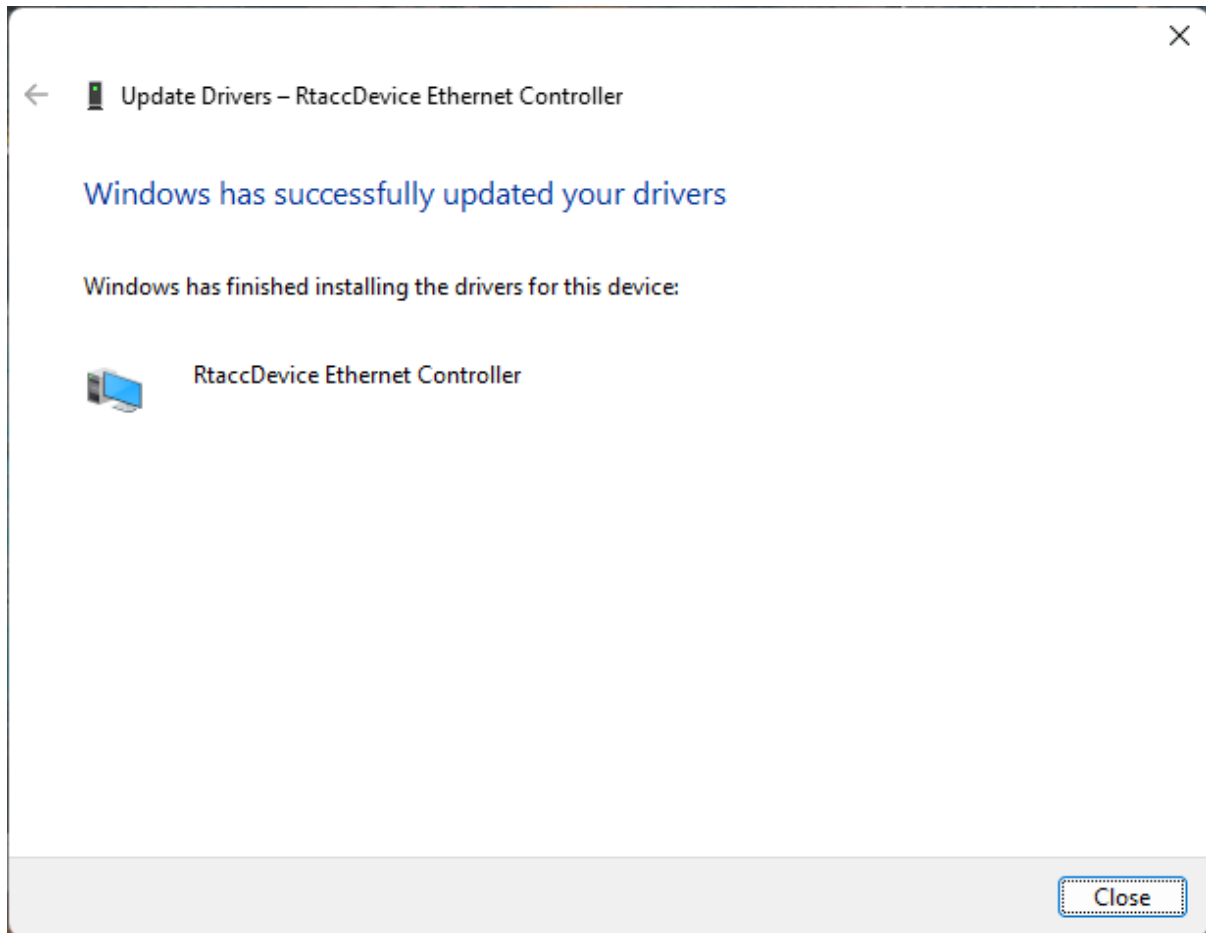
The default folder if not changed is <InstallPath>\Bin\Windows\x64



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

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



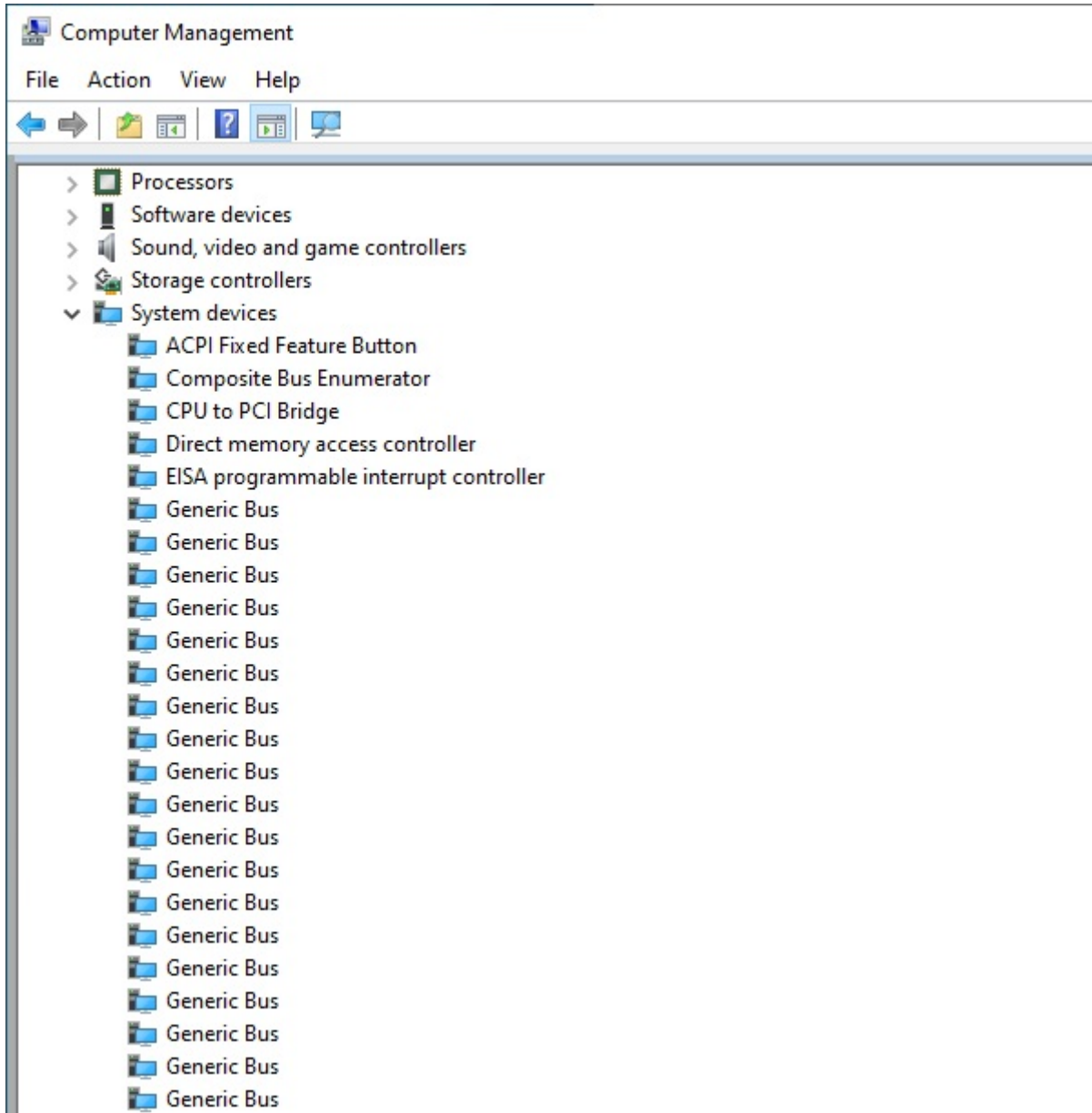


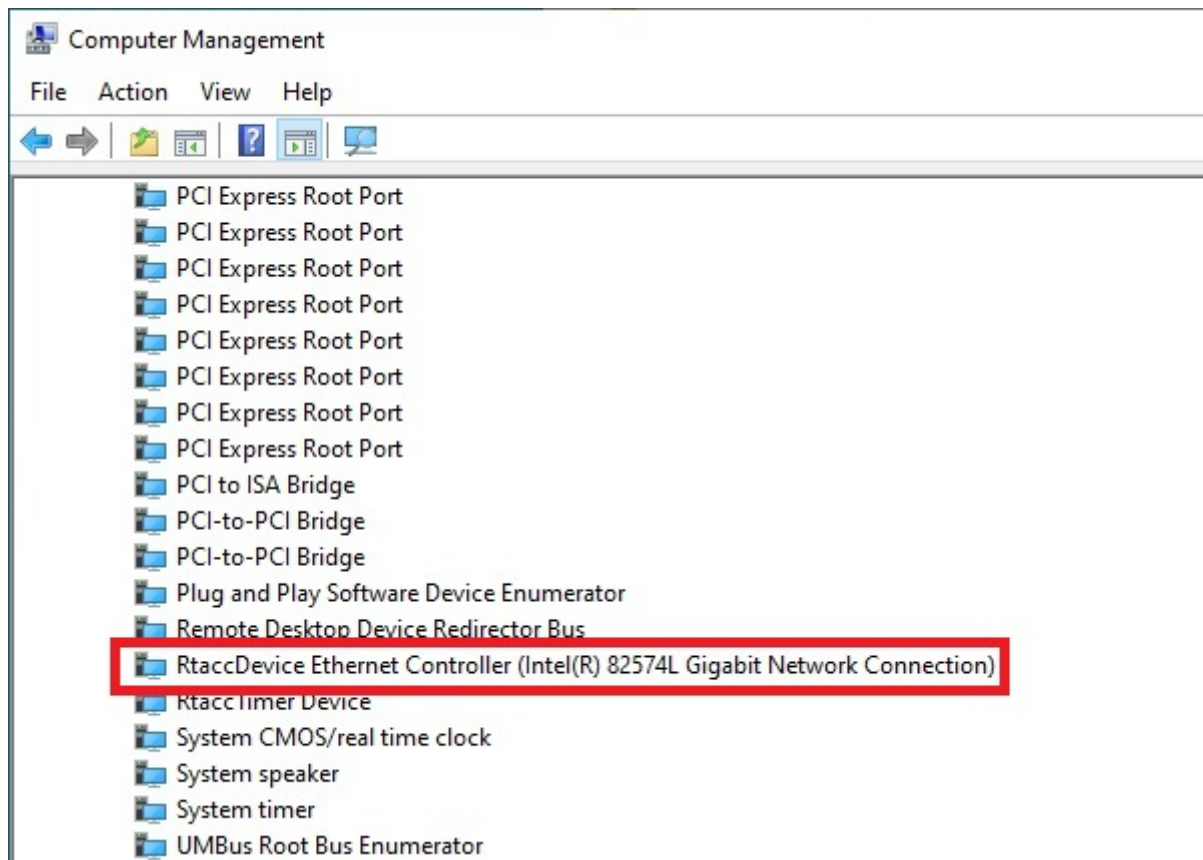
Optionally modify the search location of the Real-time Ethernet Driver

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

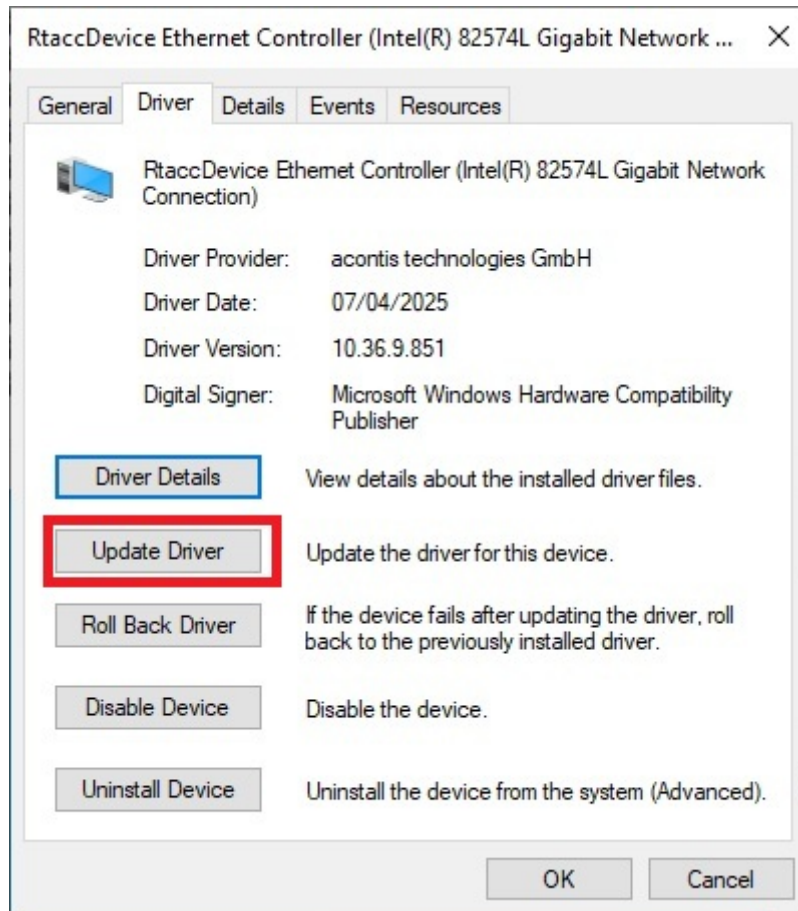
The RtaccDevice must be assigned to the network adapter for Real-time Ethernet Driver usage on Windows. In this case it is exclusively bound for EtherCAT® usage. The assignment can be reverted using the following steps:

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

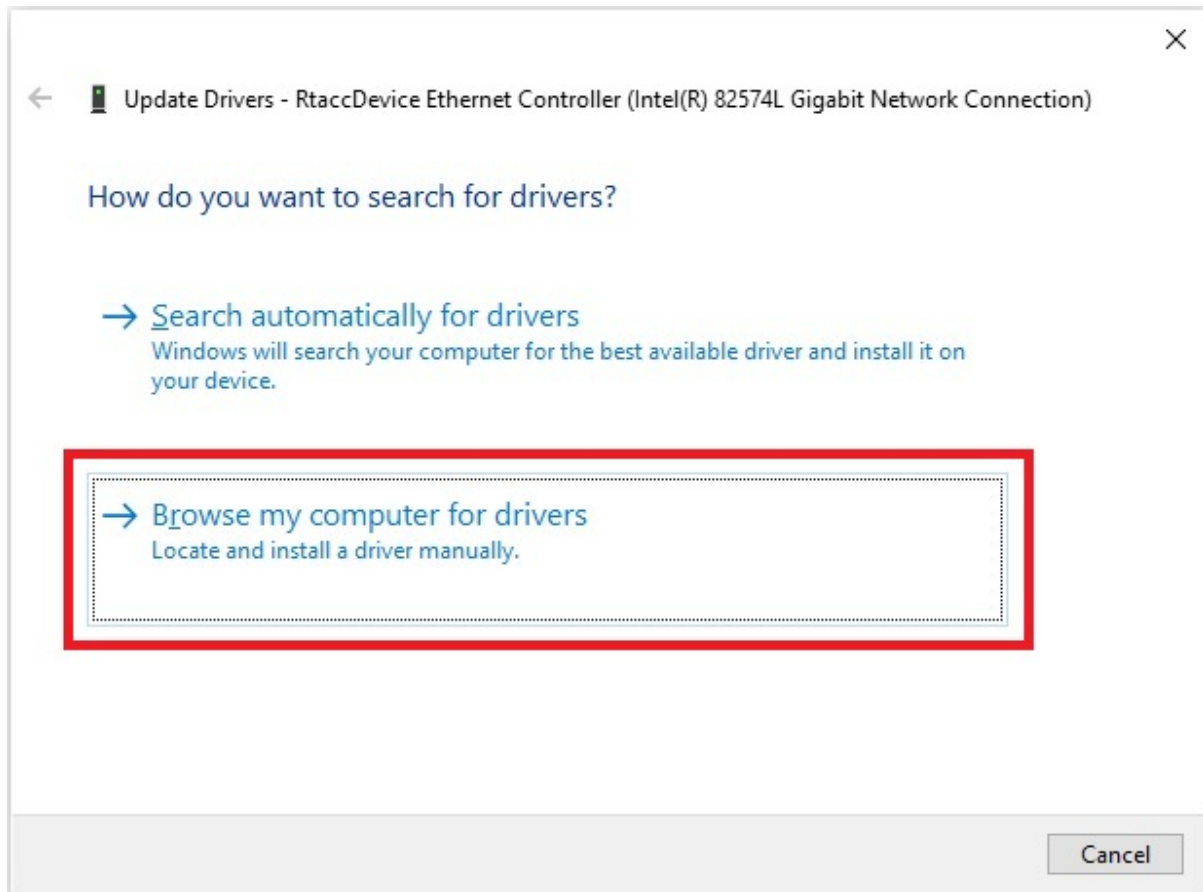




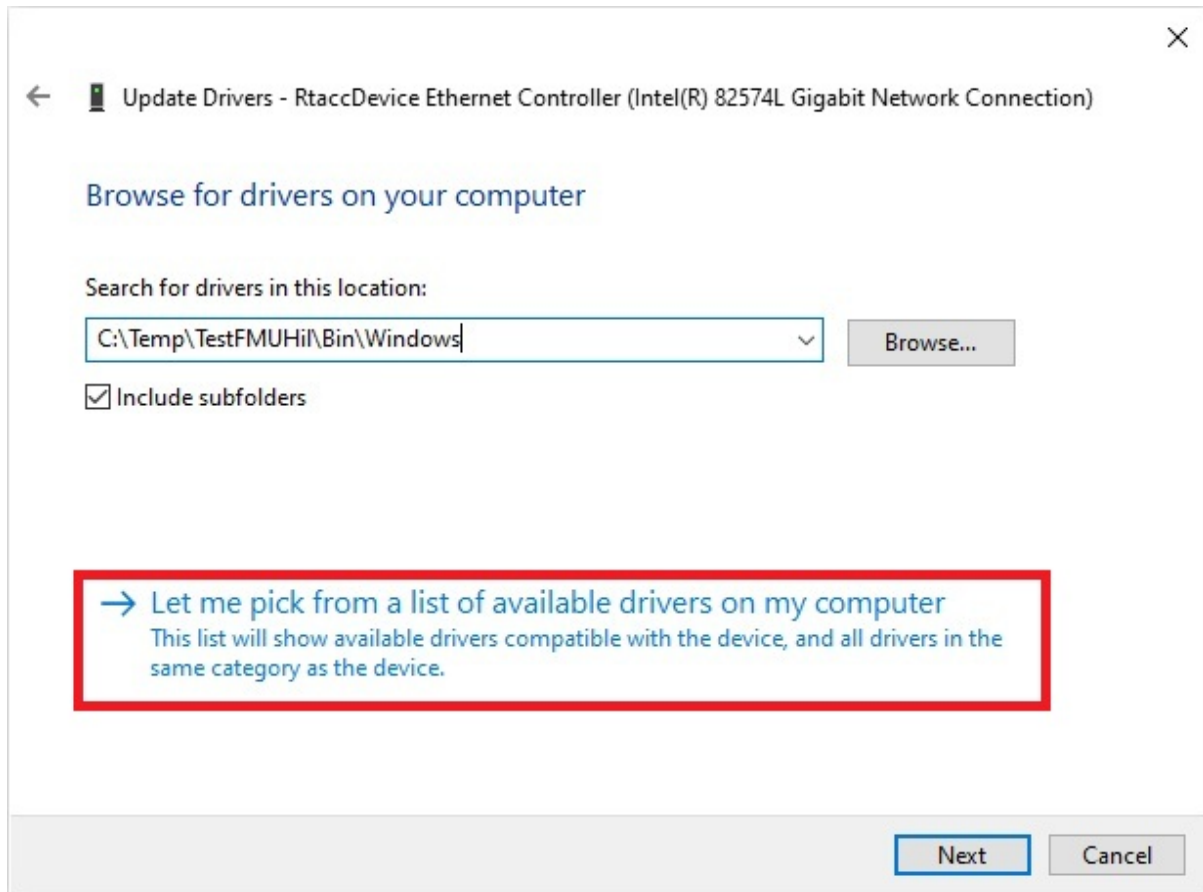
2. Right click on the RtaccDevice Ethernet Controller, then click on Properties:



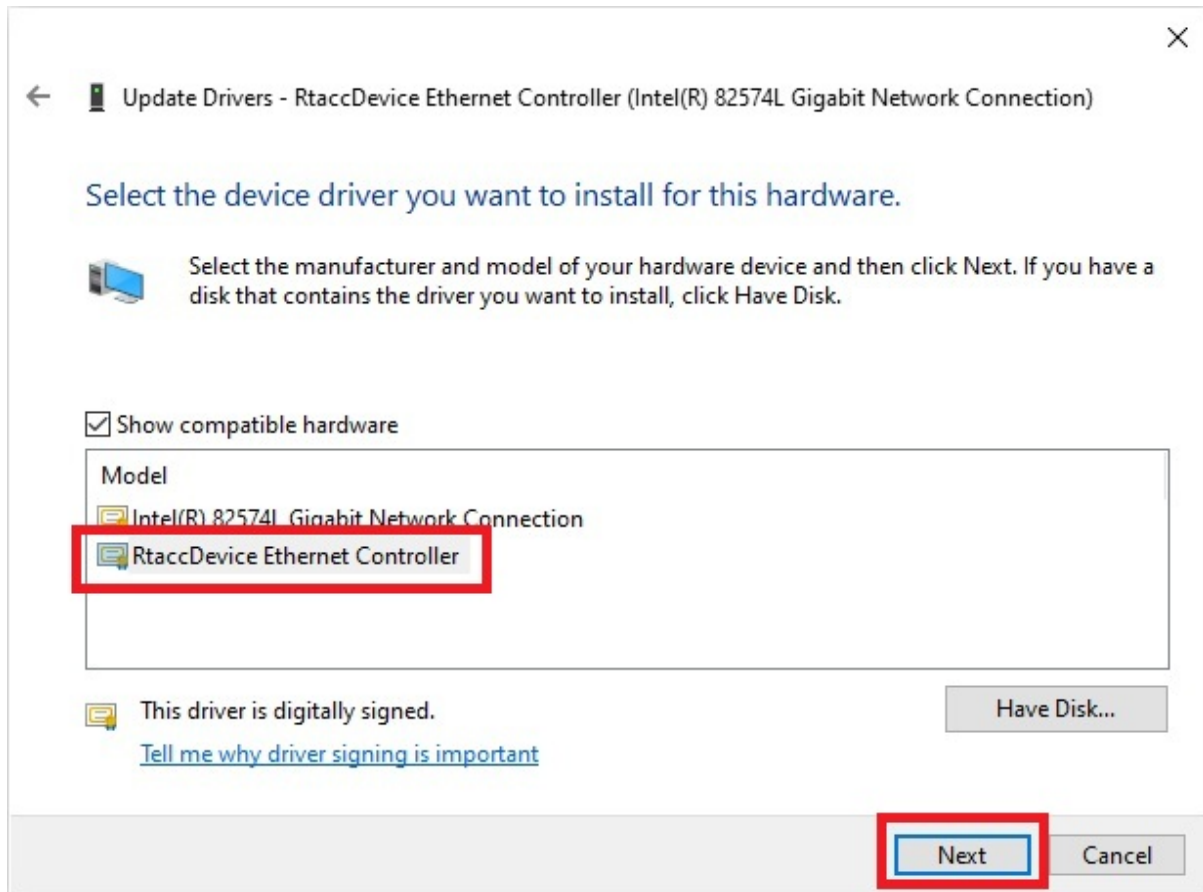
3. Switch to the tab "Driver", then click the button "Update driver":



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



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

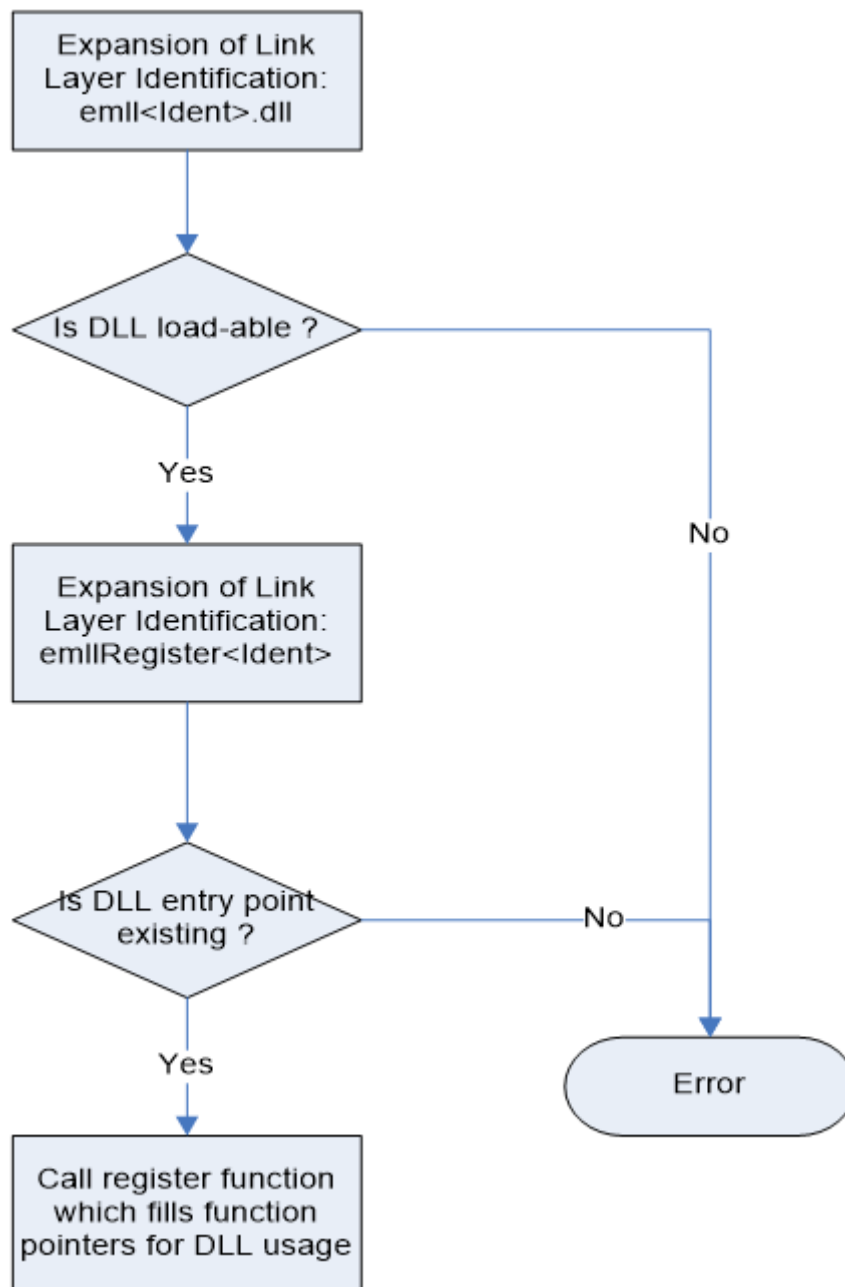


Select the network adapter and click “Next”.

The assignment is now reverted.

4.15 Microsoft Windows CE

4.15.1 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.

4.15.2 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 EC-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 EC-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

4.15.3 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.

4.15.4 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.

4.15.5 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.

4.15.6 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)
`emllI8255x.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` (EC-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"
```

```

172.17.7.148 - PuTTYtel
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

4.15.7 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
```

4.16 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)

4.16.1 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_OS_VERSION=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](#)

4.16.2 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 **LDFLAGS**. If further information is needed, please refer to <http://xenomai.org/>.

- Xenomai 4:

```
EcMasterRasServer EcMaster pthread evl dl
```

4.17 Zephyr

4.17.1 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

4.17.2 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 EC-Master currently supports a variety of different Real-time Ethernet Driver modules, each contained in a single library file, which is loaded by the core library dynamically. The EC-Master EtherCAT® MainDevice stack shipment consist of a MainDevice 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 depends on the Link Layer parameters at runtime.

Real-time means operating directly on the network adapter’s register set instead of using the operating system’s native driver.

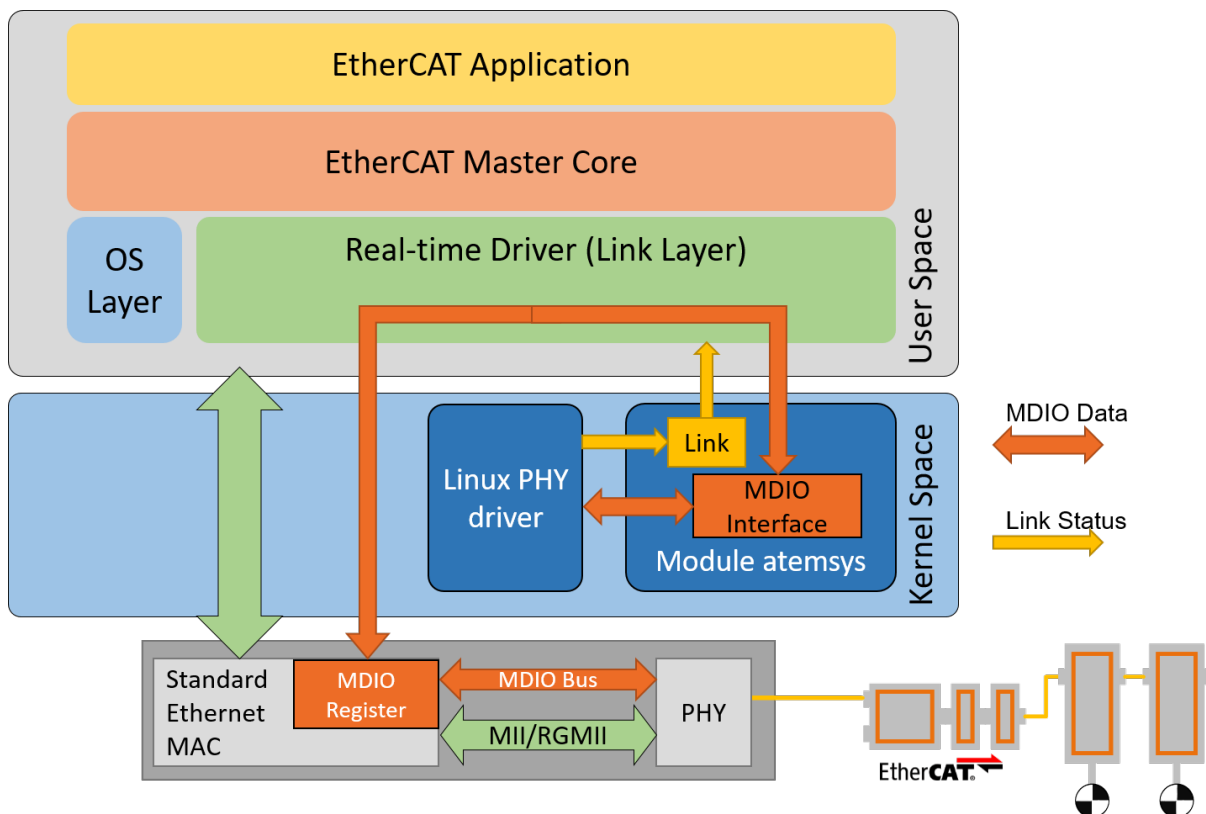
The principle of the Real-time Ethernet Driver selection is that the Real-time Ethernet Driver name (Real-time Ethernet Driver identification) is used to determine the location and name of a registration function called by the EC-Master and registers function pointers that allow access to the Real-time 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 initialization settings of the EC-Master when calling the function `emInitMaster()`.

The EC-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 EC-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. `atemsys` 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 starts 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);
OsSafeStrncpy(oLinkParmsSockRaw.linkParms.szDriverIdent, EC_LINK_PARMS_IDENT_
↳SOCKRAW, EC_DRIVER_IDENT_NAME_SIZE);

/* 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_NAME_SIZE]

[in] Name of Link Layer module (driver identification) for Link Layer Selection

EC_T_DWORD dwInstance

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

EC_T_LINKMODE eLinkMode

[in] Mode of operation

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

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

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

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

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

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

enum **EC_T_LINKMODE**

Values:

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

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

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

struct **EC_T_LINKLAYER_TIMINGTASK**

Public Members

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

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

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

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

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

EC_T_UINT64 **nSystemTime**
 [in] System

enum **EC_T_LINKLAYER_TIMING**

Values:

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

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

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

enum **EC_T_PHYINTERFACE**

Values:

enumerator **ePHY_UNDEFINED**
Undefined

enumerator **ePHY_FIXED_LINK**
No PHY access at all

enumerator **ePHY_MII**
MII 10 / 100 MBit

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

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

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

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

enumerator **ePHY OSDRIVER**
Get interface type from OS

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

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_PCILLOCATION (0x01000000)` and the PCI location must be set as `EC_T_LINK_PARAMS::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
```

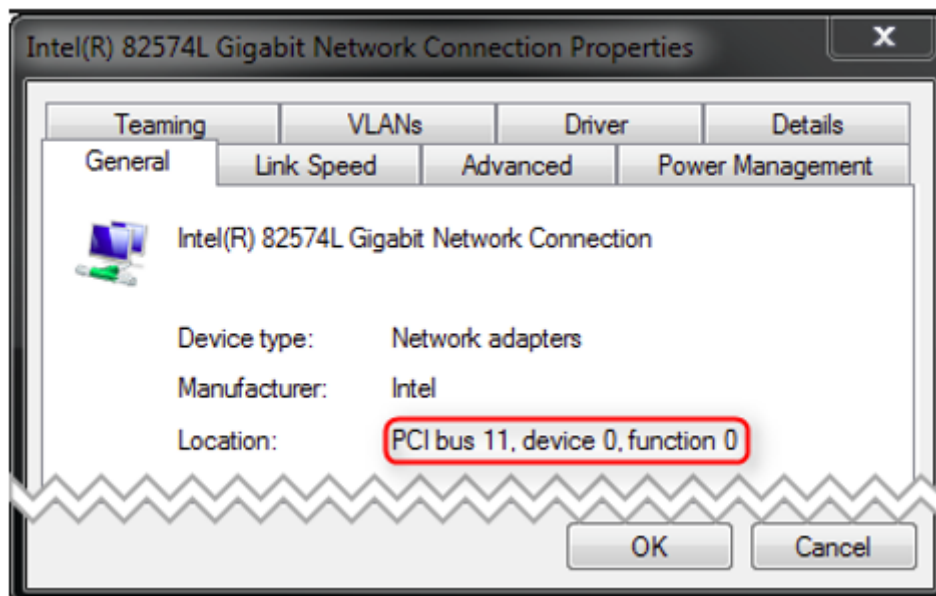
The format of `EC_T_LINK_PARAMS::dwInstance` using PCI bus address is:

`0x01bbddf`

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

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

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



5.2 Intel Pro/1000 - emllIntelGbe

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

```
struct EC_T_LINK_PARAMS_INTELGBE
```

Public Members

`EC_T_LINK_PARAMS linkParams`

Common link parameters. Signature must be set to `EC_LINK_PARAMS_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 *emInitMaster()*, before using the NIC, e.g.:

```
{
    emIoctl(dwInstanceId, EC_IOCTL_LINKLAYER_MAIN + EC_LINKIOCTL_FORCELINKMODE, EC_
    ↪NULL + 0x20103, sizeof(EC_T_DWORD), EC_NULL, 0, EC_NULL);
    OsSleep(1000);
}
```

5.2.1 TTS Feature

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

See also:

EC_T_LINKLAYER_TIMINGTASK, EC_T_LINKLAYER_TIMING

5.2.2 Supported PCI devices

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

- **PCI_DEVICE_I82540EM_DESKTOP** (0x8086, 0x100E)
- **PCI_DEVICE_I82545EM_COPPER** (0x8086, 0x100F)
- **PCI_DEVICE_I82546EB_COPPER_DUAL** (0x8086, 0x1010)
- **PCI_DEVICE_I82541EI_COPPER** (0x8086, 0x1013)
- **PCI_DEVICE_I82547GI_COPPER** (0x8086, 0x1019)
- **PCI_DEVICE_I82545GM_COPPER** (0x8086, 0x1026)
- **PCI_DEVICE_I82566MM** (0x8086, 0x1049)
- **PCI_DEVICE_I82566DM** (0x8086, 0x104A)
- **PCI_DEVICE_I82566MC** (0x8086, 0x104D)
- **PCI_DEVICE_N1E5132_SERVER** (0x8086, 0x105E)
- **PCI_DEVICE_I82547EI** (0x8086, 0x1075)
- **PCI_DEVICE_I82541GI_COPPER** (0x8086, 0x1076)
- **PCI_DEVICE_I82541GI_MOBILE** (0x8086, 0x1077)
- **PCI_DEVICE_I82541ER** (0x8086, 0x1078)
- **PCI_DEVICE_I82546GB_COPPER_DUAL** (0x8086, 0x1079)
- **PCI_DEVICE_I82541PI_DESKTOP** (0x8086, 0x107C)
- **PCI_DEVICE_I82572EI** (0x8086, 0x107D)

- **PCI_DEVICE_I82573E** (0x8086, 0x108B)
- **PCI_DEVICE_I82573** (0x8086, 0x108C)
- **PCI_DEVICE_I82573L** (0x8086, 0x109A)
- **PCI_DEVICE_I82571GB_QUAD** (0x8086, 0x10A4)
- **PCI_DEVICE_I82575_ZOAR** (0x8086, 0x10A7)
- **PCI_DEVICE_I82572GI** (0x8086, 0x10B9)
- **PCI_DEVICE_I82571GB_QUAD_2** (0x8086, 0x10BC)
- **PCI_DEVICE_I82566L** (0x8086, 0x10BD)
- **PCI_DEVICE_I82576** (0x8086, 0x10C9)
- **PCI_DEVICE_I82567V** (0x8086, 0x10CE)
- **PCI_DEVICE_I82574L** (0x8086, 0x10D3)
- **PCI_DEVICE_I82567LM3** (0x8086, 0x10DE)
- **PCI_DEVICE_I82577LM** (0x8086, 0x10EA)
- **PCI_DEVICE_I82577LC** (0x8086, 0x10EB)
- **PCI_DEVICE_I82578DM** (0x8086, 0x10EF)
- **PCI_DEVICE_I82578DC** (0x8086, 0x10F0)
- **PCI_DEVICE_I82567LM** (0x8086, 0x10F5)
- **PCI_DEVICE_I82567V3** (0x8086, 0x1501)
- **PCI_DEVICE_I82579LM** (0x8086, 0x1502)
- **PCI_DEVICE_I82579V** (0x8086, 0x1503)
- **PCI_DEVICE_I82576NS** (0x8086, 0x150A)
- **PCI_DEVICE_I82583V** (0x8086, 0x150C)
- **PCI_DEVICE_I82580_QUAD** (0x8086, 0x150E)
- **PCI_DEVICE_I350** (0x8086, 0x1521)
- **PCI_DEVICE_I82576_ET2** (0x8086, 0x1526)
- **PCI_DEVICE_I82580_QUAD_FIBRE** (0x8086, 0x1527)
- **PCI_DEVICE_I210AT** (0x8086, 0x1531)
- **PCI_DEVICE_I210AT_2** (0x8086, 0x1532)
- **PCI_DEVICE_I210_COPPER** (0x8086, 0x1533)
- **PCI_DEVICE_I210IT** (0x8086, 0x1535)
- **PCI_DEVICE_I210_SERDES** (0x8086, 0x1537)
- **PCI_DEVICE_I211AT** (0x8086, 0x1539)
- **PCI_DEVICE_I210_COPPER_FLASHLESS** (0x8086, 0x157B)
- **PCI_DEVICE_I210_BACKPLANE** (0x8086, 0x157C)
- **PCI_DEVICE_I217LM** (0x8086, 0x153A)
- **PCI_DEVICE_I217V** (0x8086, 0x153B)
- **PCI_DEVICE_I218LM** (0x8086, 0x155A)
- **PCI_DEVICE_I218V** (0x8086, 0x1559)
- **PCI_DEVICE_I218LM_2** (0x8086, 0x15A0)
- **PCI_DEVICE_I218V_2** (0x8086, 0x15A1)
- **PCI_DEVICE_I218LM_3** (0x8086, 0x15A2)
- **PCI_DEVICE_I218V_3** (0x8086, 0x15A3)
- **PCI_DEVICE_I219LM** (0x8086, 0x156F)
- **PCI_DEVICE_I219LM_2** (0x8086, 0x15B7)
- **PCI_DEVICE_I219LM_3** (0x8086, 0x15B9)
- **PCI_DEVICE_I219LM_4** (0x8086, 0x15D7)
- **PCI_DEVICE_I219LM_5** (0x8086, 0x15E3)
- **PCI_DEVICE_I219LM_6** (0x8086, 0x15BD)
- **PCI_DEVICE_I219LM_7** (0x8086, 0x15BB)
- **PCI_DEVICE_I219LM_8** (0x8086, 0x15DF)
- **PCI_DEVICE_I219LM_9** (0x8086, 0x15E1)
- **PCI_DEVICE_I219V** (0x8086, 0x1570)
- **PCI_DEVICE_I219V_2** (0x8086, 0x15B8)
- **PCI_DEVICE_I219V_4** (0x8086, 0x15D8)
- **PCI_DEVICE_I219V_5** (0x8086, 0x15D6)
- **PCI_DEVICE_I219V_6** (0x8086, 0x15BE)
- **PCI_DEVICE_I219V_7** (0x8086, 0x15BC)
- **PCI_DEVICE_I219V_8** (0x8086, 0x15E0)
- **PCI_DEVICE_I219V_9** (0x8086, 0x15E2)
- **PCI_DEVICE_I219LM_10** (0x8086, 0x0D4E)
- **PCI_DEVICE_I219V_10** (0x8086, 0x0D4F)
- **PCI_DEVICE_I219LM_11** (0x8086, 0x0D4C)
- **PCI_DEVICE_I219V_11** (0x8086, 0x0D4D)
- **PCI_DEVICE_I219LM_12** (0x8086, 0x0D53)
- **PCI_DEVICE_I219V_12** (0x8086, 0x0D55)
- **PCI_DEVICE_I219LM_18** (0x8086, 0x0DC5)
- **PCI_DEVICE_I219V_18** (0x8086, 0x0DC6)
- **PCI_DEVICE_I219LM_19** (0x8086, 0x0DC7)
- **PCI_DEVICE_I219V_19** (0x8086, 0x0DC8)
- **PCI_DEVICE_I219LM_13** (0x8086, 0x15FB)
- **PCI_DEVICE_I219V_13** (0x8086, 0x15FC)
- **PCI_DEVICE_I219LM_14** (0x8086, 0x15F9)
- **PCI_DEVICE_I219V_14** (0x8086, 0x15FA)
- **PCI_DEVICE_I219LM_15** (0x8086, 0x15F4)
- **PCI_DEVICE_I219V_15** (0x8086, 0x15F5)
- **PCI_DEVICE_I219LM_16** (0x8086, 0x1A1E)
- **PCI_DEVICE_I219V_16** (0x8086, 0x1A1F)
- **PCI_DEVICE_I219LM_17** (0x8086, 0x1A1C)
- **PCI_DEVICE_I219V_17** (0x8086, 0x1A1D)
- **PCI_DEVICE_I219LM_20** (0x8086, 0x550A)
- **PCI_DEVICE_I219V_20** (0x8086, 0x550B)
- **PCI_DEVICE_I219LM_21** (0x8086, 0x550C)
- **PCI_DEVICE_I219V_21** (0x8086, 0x550D)
- **PCI_DEVICE_I219LM_22** (0x8086, 0x550E)
- **PCI_DEVICE_I219V_22** (0x8086, 0x550F)
- **PCI_DEVICE_I219LM_23** (0x8086, 0x5510)
- **PCI_DEVICE_I219V_23** (0x8086, 0x5511)
- **PCI_DEVICE_I219LM_24** (0x8086, 0x57A0)
- **PCI_DEVICE_I219V_24** (0x8086, 0x57A1)
- **PCI_DEVICE_I225LM** (0x8086, 0x15F2)
- **PCI_DEVICE_I225V** (0x8086, 0x15F3)
- **PCI_DEVICE_I225I** (0x8086, 0x15F8)
- **PCI_DEVICE_I225K** (0x8086, 0x3100)
- **PCI_DEVICE_I225K_2** (0x8086, 0x3101)
- **PCI_DEVICE_I225LMVP** (0x8086, 0x5502)
- **PCI_DEVICE_I225IT** (0x8086, 0x0D9F)
- **PCI_DEVICE_I226LM** (0x8086, 0x125B)
- **PCI_DEVICE_I226V** (0x8086, 0x125C)
- **PCI_DEVICE_I226IT** (0x8086, 0x125D)

5.3 Intel Pro/100 - emllI8255x

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;

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

5.3.1 Supported PCI devices

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

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

5.4 Broadcom Genet - emllBcmGenet

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

```
struct EC_T_LINK_PARMS_BCMGENET
```

Public Members

EC_T_LINK_PARMS **linkParms**

Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_BCMGENET.

EC_T_BCMGENET_TYPE **eSocType**

Broadcom processor type

EC_T_BOOL **bNotUseDmaBuffers**

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

enum **EC_T_BCMGENET_TYPE**

Values:

enumerator **eBCMGENET_BCM2711**

Broadcom BCM2711, Raspberry Pi 4

5.5 Broadcom NetXtreme - emllBcmNetXtreme

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

struct **EC_T_LINK_PARMS_BCMNETXTREME**

Public Members

EC_T_LINK_PARMS **linkParms**

Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_BCMNETXTREME.

EC_T_DWORD **dwRxBuffers**

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

EC_T_DWORD **dwTxBuffers**

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

5.5.1 Supported PCI devices

Broadcom 571x PCI specific definitions (VendorId, DeviceId)

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

5.6 Berkeley Packet Filter - emIBPF

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® network adapters. If the main network adapter is used for both EtherCAT® network adapters and the local area network there may be a major impact on the local area network operation.

Note: 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

5.7 Beckhoff CCAT - emIBCCAT

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

5.7.1 Supported PCI devices

Beckhoff CCAT PCI specific definitions (VendorId, DeviceId)

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

5.8 CMSIS - emllCmsisEth

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

struct **EC_T_LINK_PARMS_CMSISETH**

Public Members

EC_T_LINK_PARMS **linkParams**
Common link parameters. Signature must be set to `EC_LINK_PARMS_SIGNATURE_CMSISETH`.

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

5.9 Texas Instruments CPSW - emII CPSW

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

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

EC_T_CPSW_TYPE **eCpswType**

CPSW type

EC_T_DWORD **dwPhyAddr**

PHY address

EC_T_DWORD **dwPortPrio**

0 (lowest), 1 (highest)

EC_T_BOOL **bMaster**

`EC_TRUE`: Initialize MAC

EC_T_BOOL **bPhyRestartAutoNegotiation**

`EC_TRUE`: Restart auto negotiation on initialization

EC_T_PHYINTERFACE **ePhyInterface**

PHY connection type

EC_T_DWORD **dwRxInterrupt**

Receive interrupt number (IRQ)

EC_T_BOOL **bNotUseDmaBuffers**

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

EC_T_BOOL **bNoPhyAccess**

Don't use MDIO to set up the PHY

EC_T_BOOL **bAleBypass**

Enable `bAleBypass`, similar to promiscuous mode

EC_T_DWORD **dwExtendRxFrameLength**

Receive longer Rx Frames

```
enum EC_T_CPSW_TYPE
```

Values:

enumerator **eCPSW_AM33XX**

TI AM33xx (e.g. Beaglebone)

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

enumerator **eCPSW_AM437X**
TI AM437x

enumerator **eCPSW_AM57X**
TI AM57x

5.9.1 CPSW usage under Linux

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

It is possible to use one CPSW port for Linux kernel (TCP/IP) and another CPSW port for EC-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 cpsw.c and davinci_mdio.c files.
- Kernel must be rebuilt and installed

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

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

5.10 Texas Instruments CPSWG for AM6x and Jacinto 7 - emIICP-SWG

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

```
struct EC_T_LINK_PARAMS_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 eCPSWG_AM654X

TI AM654x

enumerator eCPSWG_TDA4X

TI TDA4x (Jacinto 7)

enumerator eCPSWG_AM64X

TI AM64x

enumerator eCPSWG_AM62X

TI AM62x

5.10.1 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)

5.11 Linux DPDK - emllDpdk

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

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

```
struct EC_T_LINK_PARMS_DPK
```

Public Members

`EC_T_LINK_PARMS linkParms`

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

`EC_T_DWORD dwPortId`

DPDK port ID

`EC_T_WORD wRxBufferCnt`

Receive buffer count, 0: defaults to DPDK internal setting

`EC_T_WORD wTxBufferCnt`

Transmit buffer count, 0: default to DPDK internal setting

`EC_T_BOOL bDontCheckLinkStatus`

Don't check link status (forced)

EC_T_BOOL bSetPromiscuousMode

Change promiscuous mode setting

EC_T_BOOL bEalInitDeinitByApp

DPDK EAL layer is initialized and deinitialized by user App

Note:

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

5.11.1 Linux System Requirements

- Kernel configuration

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

```
HUGETLBFS
PROC_PAGE_MONITOR support
```

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

5.11.2 Huge pages setup

Create huge pages (not persistent)

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

5.11.3 DPDK for PCI Network Adapter

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

```
$ sudo modprobe uio_pci_generic
```

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

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

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

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

For each port to be bound to emllDpdk, call

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

Example:

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

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

5.11.4 DPDK for DPAA

Port specific numbering

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

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

DPDK	dts
0	1
1	5
2	6
3	10

Reassigning Ports to DPAA and Linux in DTB/DTS

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

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

The section dpaa-extended-args must be extended:

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

    dpaa-extended-args {

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

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

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

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

            fman0_tx1-extd-args {
                cell-index = <1>;
```

(continues on next page)

(continued from previous page)

```

};

...

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

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

...
};

```

5.11.5 DPDK for ENETC4

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

To use this version of `emllDpdk`, the `EC_T_LINK_PARAMS::szDriverIdent` of `EC_T_LINK_PARAMS_DPDK::linkParms` must be set to `EC_LINK_PARAMS_IDENT_DPDK_ENETC4`:

```

EC_T_LINK_PARAMS_DPDK* pLinkParmsAdapter = EC_NULL;
/* ... */
OsStrcpy(pLinkParmsAdapter->linkparms.szDriverIdent, EC_LINK_PARAMS_IDENT_DPDK_
↪ENETC4);

```

See also:

`CreateLinkParmsFromCmdLineDpdk()` in `EcSelectLinkLayer.cpp`

Additional requirements

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

Setting up the ENETC4 Ethernet interfaces

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

1. Load Kernel modules

```
$ modprobe uio_pci_generic
$ modprobe kpage_nocache
```

2. Setup hugepages

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

3. Identify the PF(Physical Function) PCI address

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

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

```

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

```

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

```

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

```

5. Create Virtual Function VF0

```

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

```

6. Identify the VF0 PCI address

```

$ dpdk-devbind.py --status

```

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

```

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

```

7. Bind VF0 to DPDK

```

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

```

8. Enable trust for VF0

```

ip link set end0 vf 0 trust on

```

See also:

[NXP RM00293](#)

5.11.6 Limitations

If the adapter needs to be used multiple times, for cable redundancy or calling `emInitMaster()` multiple times, the application must set `EC_T_LINK_PARAMS_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();

```

5.12 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

`EC_T_DWORD` **dwRxInterrupt**

Receive interrupt number (IRQ)

```
enum EC_T_DW3504_TYPE
```

Values:

enumerator **eDW3504_CycloneV**
MAC on Cyclone V SoC

enumerator **eDW3504_LCES1**
MAC on LCES1 SoC

enumerator **eDW3504_RZN1**
MAC on Renesas RZN1

enumerator **eDW3504_STM32MP15x**
MAC on STM32MP15x

enumerator **eDW3504_ATOM**
MAC on Atom 6000

enumerator **eDW3504_STM32MP13x**
MAC on STM32MP13x

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

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

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

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

enumerator **eDW3504_SemidriveD9**
MAC on Semidrive D9

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

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

enumerator **eDW3504_AllWinnerT536**
MAC on AllWinner T536

enumerator **eDW3504_RZN2**
MAC on Renesas RZN2

enumerator **eDW3504_RK3506**
MAC on Rockchip 3506

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

5.12.1 Supported PCI devices

Synopsis DW3504 PCI specific definitions (VendorId, DeviceId)

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

5.13 Freescale TSEC / eTSEC - emIIETSEC

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

```
struct EC_T_LINK_PARMS_ETSEC
```

Public Members

EC_T_LINK_PARMS linkParms

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

EC_T_DWORD dwRegisterBase

Physical base address of register block (4k)

EC_T_DWORD dwLocalMdioBase

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

EC_T_DWORD dwPhyMdioBase

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

EC_T_DWORD dwPhyAddr

PHY address on MII bus. `ETSEC_FIXED_LINK` if fixed link configuration.

EC_T_DWORD dwTbiPhyAddr

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

EC_T_DWORD dwFixedLinkVal

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

EC_T_BYTE abyStationAddress[6]

MAC address

EC_T_VOID *oMiiBusMtx

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

EC_T_DWORD dwRxInterrupt

Receive interrupt number (IRQ)

EC_T_BOOL bNotUseDmaBuffers

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

EC_T_ETSEC_TYPE eETSECType

System on Chip type

EC_T_BOOL bMaster

Full control over the MAC and need to initialize MAC and the connections to the PHYs

enum **EC_T_ETSEC_TYPE**

Values:

enumerator **eETSEC_P2020RDB**

MAC on Freescale P2020

enumerator **eETSEC_TWRP1025**

MAC on Freescale TWRP1025

enumerator **eETSEC_ISTMPC8548**

MAC on Freescale ISTMPC8548

enumerator **eETSEC_XJ_EPU20C**

MAC on Freescale XJ EPU20C

enumerator **eETSEC_TWRLS1021A**

MAC on Freescale TWRLS1021A

enumerator **eETSEC_TQMLS_LS102XA**

MAC on Freescale TQMLS LS102XA

5.13.1 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)

5.13.2 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;
```

5.13.3 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); ↵
↵ */
```

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

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

`EcLinkGetStatus` always returns the last known link status.

5.13.5 Fixed Link

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

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

Parameters for fixed link:

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

5.14 Freescale FslFec - emlIFslFec

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

```
struct EC_T_LINK_PARMS_FSLFEC
```

Public Members

EC_T_LINK_PARMS **linkParams**

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

`EC_T_DWORD` **dwRxBuffers**

Receive buffer count

`EC_T_DWORD` **dwTxBuffers**

Transmit buffer count

EC_T_FEC_TYPE **eFecType**

System on Chip type

EC_T_PHYINTERFACE **ePhyInterface**

PHY interface type

EC_T_BOOL **bUseDmaBuffers**

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

EC_T_DWORD **dwPhyAddr**

PHY Address

EC_T_BOOL **bNoPinMuxing**

No clock configuration and pin muxing

EC_T_BOOL **bDontReadMacAddr**

Read of MAC address disabled

EC_T_DWORD **dwRxInterrupt**

Receive interrupt number (IRQ)

EC_T_BOOL **bNoPhyAccess**

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

EC_T_BOOL **bFlushFramesRequired**

EC_FALSE: flush cyclic / acyclic frames queued at link layer

enum **EC_T_FEC_TYPE**

Values:

enumerator **eFEC_IMX25**

MAC on Freescale i.MX25 (ARM9; ARMv5)

enumerator **eFEC_IMX28**

MAC on Freescale i.MX28 (ARM9; ARMv5)

enumerator **eFEC_IMX53**

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

enumerator **eFEC_IMX6**

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

enumerator **eFEC_VF6**

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

enumerator **eFEC_IMX7**

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

enumerator **eFEC_IMX8**

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

enumerator **eFEC_IMX8M**

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

enumerator **eFEC_IMXRT1064**

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

enumerator **eFEC_IMX9**

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

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

5.15 Cadence GEM/MACB - emllGEM

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

struct **EC_T_LINK_PARMS_GEM**

Public Members

EC_T_LINK_PARMS **linkParms**

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

EC_T_GEM_RXSOURCE **eRxSource**

Source of RX clock, control and data signals

EC_T_DWORD dwPhyAddr

PHY address

EC_T_DWORD dwRxInterrupt

Receive interrupt number (IRQ)

EC_T_BOOL bUseDmaBuffers

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

EC_T_BOOL bNoPhyAccess

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

EC_T_BOOL bUseGmiiToRgmiiConv

Use XILINX GMIITORGMI Converter (`EC_TRUE`)

EC_T_DWORD dwConvPhyAddr

PHY address used to communicate with converter. In Linux doc it is named "reg".

EC_T_DWORD dwTxDmaDesCnt

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

EC_T_DWORD dwRxDmaDesCnt

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

EC_T_GEM_TYPE **eGemType**

System on Chip type

EC_T_PHYINTERFACE **ePhyInterface**

PHY connection type

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

EC_T_GEM_CLK_DIV_TYPE **eClkDivType**
Change Ref Clock settings

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

enum **EC_T_GEM_RXSOURCE**

Values:

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

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

enum **EC_T_GEM_TYPE**

Values:

enumerator **eGemType_Zynq7000**
Xilinx Zynq 7000

enumerator **eGemType_ZynqUltrascale**
Xilinx Zynq Ultrascale

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

enumerator **eGemType_PolarFire**
Microchip PolarFire SoC

5.16 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

5.17 Texas Instruments ICSS - emllICSS

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

struct **EC_T_LINK_PARMS_ICSS**

Public Members

EC_T_LINK_PARMS **linkParams**

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

EC_T_LINK_ICSS_BOARD **eBoardType**

TI System on Chip board type

EC_T_BOOL bMaster

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

EC_T_BOOL bNoMacAddr

`EC_TRUE`: No MAC address registers access

EC_T_DWORD dwPhyAddr

PHY address

EC_T_PHYINTERFACE **ePhyInterface**

PHY connection type

EC_T_BOOL bNoPhyReset

No hardware reset of the PHY

EC_T_BOOL bUseAllSendQueues

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

EC_T_BOOL bLegacyFirmware

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

enum **EC_T_LINK_ICSS_BOARD**

Values:

enumerator **EcLinkIcssBoard_Unsupported**

enumerator **EcLinkIcssBoard_am572x**
TI AM572x

enumerator **EcLinkIcssBoard_am571x**
TI AM571x

enumerator **EcLinkIcssBoard_am3359**
TI AM3359

enumerator **EcLinkIcssBoard_am572x_emerson**
TI AM572x on Emerson board

enumerator **EcLinkIcssBoard_am574x**
TI AM574x

5.17.1 TTS Feature

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

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

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

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

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

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

5.17.2 TI AM335x ICEV2

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

See also:

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

5.17.3 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).

5.17.4 AM5728 IDK and AM5718 IDK boards and Technical Limitations

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

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

5.18 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 the link layer is used on a single ICSSG port. This flag is also required, when the 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

5.18.1 TI AM654x IDK

Support for ICSSG on AM654x is currently limited to TI AM654x IDK/EVK with TI RTOS/Sysbios in polling mode.

5.19 Microchip LAN743x - emllan743x

The parameters to the Real-time Ethernet Driver LAN743x are setup-specific. The function `CreateLinkParamsFromCmdLineLAN743x()` 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 **linkParams**

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)

- `PCI_DEVICE_LAN743X` (0x1055, 0x7430)

5.20 Beckhoff CUxxxx Multiplier - emllMultiplier

The parameters to the Multiplier Real-time Ethernet Driver are setup-specific. The function “`CreateLinkParamsFromCmdLineMultiplier`” 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

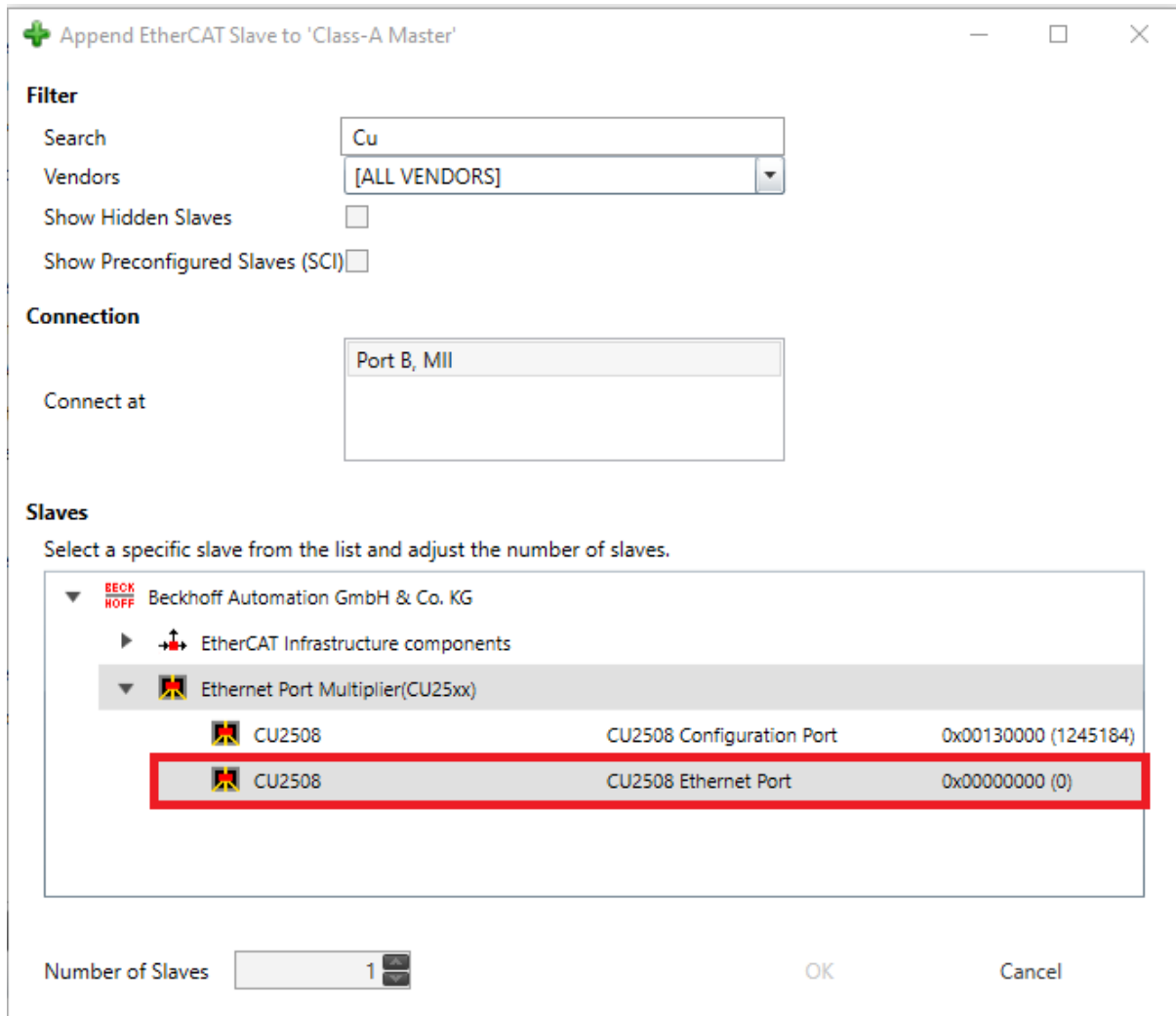
enumerator **eMultiplier_Et2000**

Beckhoff ET2000 industrial Ethernet multi-channel probe

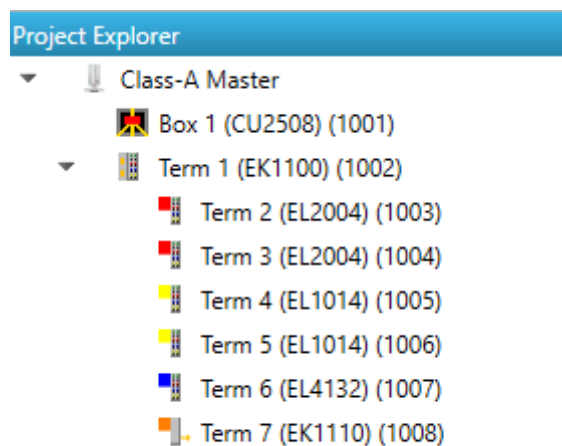
5.20.1 Configuration with EC-Engineer

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 SubDevice.



- Append the SubDevices that you are going to connect to the port.



5.21 Windows NDIS - emllNdis

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

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

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

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

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

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

struct **EC_T_LINK_PARMS_NDIS**

Public Members

EC_T_LINK_PARMS **linkParms**

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

EC_T_CHAR szAdapterName[`EC_NDIS_ADAPTER_NAME_SIZE`]

ServiceName of network adapter, see `HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\NetworkCards` in registry (zero terminated)

EC_T_BYTE abyIpAddress[4]

IP address of network adapter

EC_T_BOOL bDisablePromiscuousMode

Disable adapter promiscuous mode

EC_T_BOOL bDisableForceBroadcast

Don't change target MAC address to `FF:FF:FF:FF:FF:FF`

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

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

5.22 Windows WinPcap - emllPcap

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

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

```
struct EC_T_LINK_PARMS_WINPCAP
```

Public Members

EC_T_LINK_PARMS **linkParms**

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

`EC_T_BYTE` **abyIpAddress**[4]

IP address

`EC_T_CHAR` **szAdapterId**[`MAX_LEN_WINPCAP_ADAPTER_ID`]

Adapter ID, format: {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}

`EC_T_BOOL` **bFilterInput**

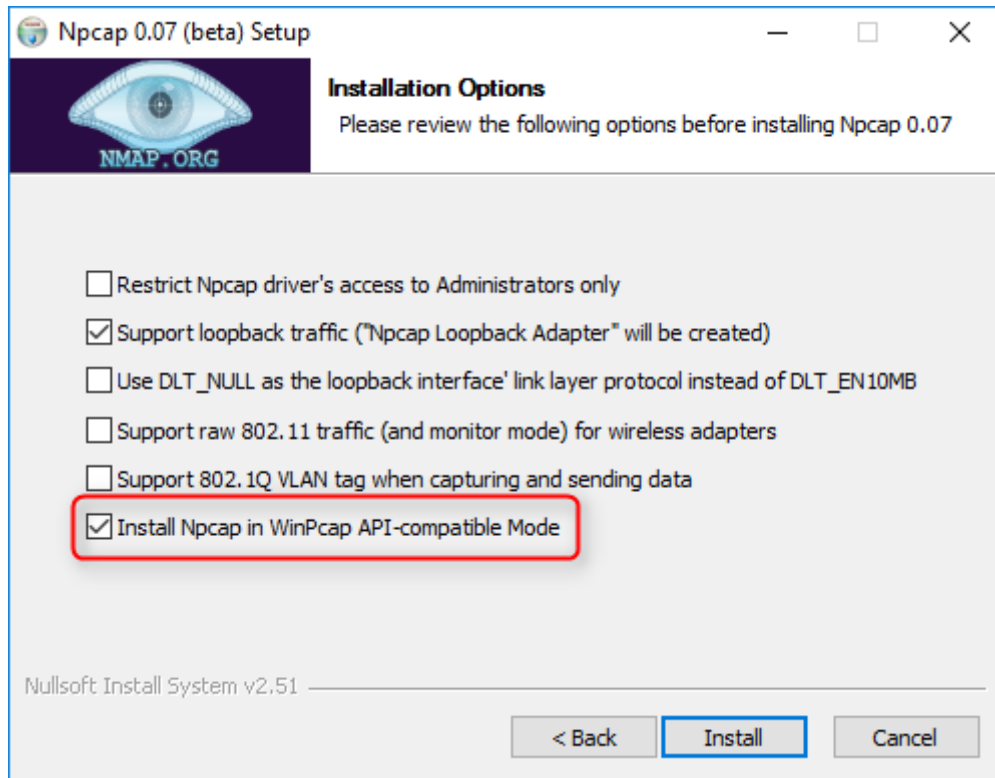
Filter input if `EC_TRUE`. This is needed on some system if the winpcap library notify the sent frames to the network adapter.

5.22.1 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)

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

struct **EC_T_LINK_PARMS_REMOTE**

Public Members

EC_T_LINK_PARMS **linkParms**

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

EC_T_DWORD **dwSocketType**

Socket type. Must be set to 1 (`emrsockettype_tcp`).

EC_T_BYTE **abySrcIpAddress[4]**

Source adapter IP address (listen)

EC_T_WORD **wSrcPort**

Source port number (listen)

EC_T_BYTE **abyDstIpAddress[4]**

Destination adapter IP address (connect)

EC_T_WORD **wDstPort**

Destination port number (connect)

EC_T_BYTE **abyMac[6]**

MAC address

EC_T_DWORD **dwRxBufferCnt**

Frame buffer count for interrupt service thread (IST)

While EC-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 - emlRTL8169

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)

- **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 - emIIIRZT1

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 - emIISEth

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

```
struct EC_T_LINK_PARMS_SHETH
```

Public Members

EC_T_LINK_PARMS **linkParms**

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

EC_T_SHETH_TYPE **eType**

System on Chip type

EC_T_BYTE **abyStationAddress[6]**

MAC address

EC_T_DWORD **dwBaseAddr**

Physical address of register block

EC_T_BYTE **byPhyAddr**

PHY address

EC_T_BOOL **bNotUseDmaBuffers**

`EC_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 the 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 not be called from the JobTask's context.

```
dwRes = emIoCtl((EC_IOCTL_LINKLAYER | EC_LINK_IOCTL_UPDATE_LINKSTATUS), EC_NULL);
```

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

5.27.2 SHEth usage under Linux

Due to the 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 - emIISNARF

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 initializing 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 and the needed CPU time is often too high to reach cycle times less than 300 to 500 microseconds. Additionally there is an impact if in parallel to EtherCAT® traffic the VxWorks application needs to use a second network card for transferring TCP/IP data. The single `tNetTask` is shared by all network drivers. Using a dedicated EtherCAT® driver these disadvantages can be overcome.

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

```
struct EC_T_LINK_PARMS_SNARF
```

Public Members

`EC_T_LINK_PARMS linkParams`

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

`EC_T_CHAR szAdapterName[EC_SNARF_ADAPTER_NAME_SIZE]`

SNARF adapter name (zero terminated)

`EC_T_DWORD dwRxBuffers`

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

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

(continues on next page)

(continued from previous page)

```

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

```

5.29 Linux SockRaw - emllSockRaw

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

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

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

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

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

```

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

```

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

```
struct EC_T_LINK_PARMS_SOCKRAW
```

Public Members

EC_T_LINK_PARMS linkParms

Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_SOCKRAW.

EC_T_CHAR szAdapterName[EC_SOCKRAW_ADAPTER_NAME_SIZE]

Native ETH device name, e.g. “eth0” (zero terminated)

EC_T_BOOL bDisableForceBroadcast

Don't change target MAC address to FF:FF:FF:FF:FF:FF

EC_T_BOOL bReplacePaddingWithNopCmd

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

EC_T_BOOL bUsePacketMmapRx

Use PACKET_MMAP PACKET_RX_RING for receive

EC_T_BOOL bSetCoalescingParms

Set Coalescing parameters to enhance the link layer performance

EC_T_BOOL bSetPromiscuousMode

Enable promiscuous mode at network adapter

5.30 Linux SockXdp - emllSockXdp

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

Note: Root privileges are required.

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

```
struct EC_T_LINK_PARMS_SOCKXDP
```

Public Members

EC_T_LINK_PARMS **linkParms**Common link parameters. Signature must be set to `EC_LINK_PARMS_SIGNATURE_SOCKXDP`.**EC_T_CHAR szAdapterName**[`EC_SOCKXDP_ADAPTER_NAME_SIZE`]

Native ETH device name, e.g. "eth0" (zero terminated)

EC_T_WORD wQueueId

Used NIC Queue Id

EC_T_XDP_MODE **eXdpMode**

XDP mode

EC_T_BOOL bDisableForceBroadcast

Don't change target MAC address to FF:FF:FF:FF:FF:FF

EC_T_BOOL bReplacePaddingWithNopCmd

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

```
enum EC_T_XDP_MODE
```

*Values:*enumerator **eXDP_SKB_MODE**

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

enumerator **eXDP_DRV_MODE**

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

enumerator **eXDP_HW_MODE**

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

enumerator **eXDP_DRV_ZEROCOPY**

Native XDP with ZEROCOPY

5.30.1 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 AMD Tri-Mode Ethernet Media Access Controller (TEMAC)

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

```
struct EC_T_LINK_PARMS_TEMAC
```

Public Members

EC_T_LINK_PARMS linkParms

Common link parameters. Signature must be set to EC_LINK_PARMS_SIGNATURE_TEMAC

EC_T_DWORD dwRxBufferCnt

Receive buffer count, 0: default to 100

5.31.1 Encustra Mercury_XU5_PE1

Support for FPGA designs with TEMAC with XAE_SOFT_TEMAC_LOW_SPEED and XPAR_AXI_FIFO.

5.32 Texas Instruments CPSWG for AM6x and Jacinto 7 based on Enet LLD - emllTiEnetCpswg

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

5.32.1 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.

5.33 Texas Instruments ICSSG for AM6x and Jacinto 7 based on Enet LLD - emllTiEnetIcssg

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_BYTE **byPhyAddr**

PHY address

5.33.1 TI AM64x EVM

Working with MCU Plus SDK AM64x 08.01.00.36 and MCU Plus SDK AM64x 11.00.00.15

5.33.2 TI AM243x LP and TI AM243x EVM

Working with MCU Plus SDK AM243x 08.01.00.36 and MCU Plus SDK AM243x 11.00.00.15

5.34 Virtual Local Area Network - emllVlan

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

5.34.1 Switch Configuration

Ensure that the Spanning Tree Protocol (STP) is disabled on the ports used for EtherCAT, because it can lead to permanent frame loss.

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.

<code>EC_E_NOERROR</code>	The function was successfully executed
<code>EC_E_NOTSUPPORTED</code>	Unsupported feature or functionality
<code>EC_E_BUSY</code>	The MainDevice is currently busy and the function has to be re-tried at a later time
<code>EC_E_NOMEMORY</code>	Not enough memory or frame buffer resources available
<code>EC_E_INVALIDPARM</code>	Invalid or inconsistent parameters
<code>EC_E_TIMEOUT</code>	Timeout error
<code>EC_E_SLAVE_ERROR</code>	A SubDevice error was detected

See also:

- `emNotify` - `EC_NOTIFY_STATUS_SLAVE_ERROR`
- `emNotify` - `EC_NOTIFY_SLAVE_ERROR_STATUS_INFO`

<code>EC_E_INVALID_SLAVE_STATE</code>	The SubDevice is not in the requested state to execute the operation (e.g. when initiating a mailbox transfer the SubDevice must be at least in PREOP state)
<code>EC_E_SLAVE_NOT_ADDRESSABLE</code>	The SubDevice does not respond to its station address (e.g. when requesting its <code>AL_STATUS</code> value). The SubDevice may be removed from the bus or powered-off.
<code>EC_E_LINK_DISCONNECTED</code>	Link cable not connected
<code>EC_E_MASTERCORE_INACCESSIBLE</code>	MainDevice core inaccessible. This result code usually means a remote connected server / EtherCAT® MainDevice 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 MainDevice is requested to enter the OPERATIONAL state the next request from the API will return this status value unless the OPERATIONAL state is entered.

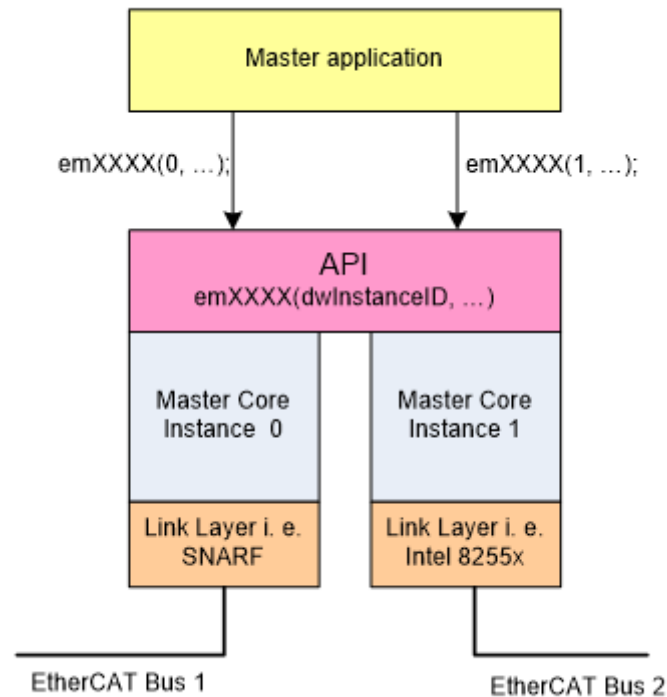
6.2 Multiple EtherCAT® Bus Support

6.2.1 Licensing

Multiple EtherCAT® Bus support is included within the Class B and Class A MainDevice 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® MainDevice allows controlling more than one EtherCAT® bus within one application process. For this use case the MainDevice 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 MainDevice 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 MainDevice instances with the following configuration (`e19800.xml`):

- MainDevice instance 0: One Beckhoff EtherCAT® Evaluation Board EL9800
- MainDevice instance 1: One Beckhoff EtherCAT® Evaluation Board EL9800

Parameters for this application:

```
$ -ndis 192.168.1.32 1 -f e19800.xml @ -ndis 192.168.2.32 1 -f e19800.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 EtherCAT stack 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**

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 dwBusCycleTimeNsec

[in] Bus cycle time [ns]

EC_T_DWORD dwMaxBusSlaves

[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_BYTE byReserved

[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_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 ms count

struct **EC_T_LOG_PARMS**

Public Members

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

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

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

EC_LOG_LEVEL... The following log levels are defined:

`EC_LOG_LEVEL_SILENT`

`EC_LOG_LEVEL_ANY`

`EC_LOG_LEVEL_CRITICAL`

`EC_LOG_LEVEL_ERROR`

`EC_LOG_LEVEL_WARNING`

`EC_LOG_LEVEL_INFO`

`EC_LOG_LEVEL_INFO_API`

`EC_LOG_LEVEL_VERBOSE`

`EC_LOG_LEVEL_VERBOSE_ACYC`

`EC_LOG_LEVEL_VERBOSE_CYC`

`EC_LOG_LEVEL_UNDEFINED`

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

Param pContext

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

Param dwLogMsgSeverity

[in] Log message severity, `EC_LOG_LEVEL_...`

Param szFormat

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

Return

`EC_E_NOERROR` or error code

Log messages are passed from the EC-Master to the callback given at `EC_T_LOG_PARAMS::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_PARAMS::dwLogLevel`. E.g. messages of severity `EC_LOG_LEVEL_WARNING` are not passed to the application if `EC_T_LOG_PARAMS::dwLogLevel` is set to `EC_LOG_LEVEL_ERROR`.

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

```
struct EC_T_PERF_MEAS_INTERNAL_PARMS
```

Public Members

`EC_T_BOOL bEnabled`

[in] Enable/disable internal performance counters

`EC_T_PERF_MEAS_COUNTER_PARMS CounterParms`

[in] Timer function settings. When not provided `OsMeasGetCounterTicks` is used.

`EC_T_PERF_MEAS_HISTOGRAM_PARMS HistogramParms`

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

```
struct EC_T_PERF_MEAS_COUNTER_PARMS
```

Public Members

`EC_PF_PERF_MEAS_GETCOUNTERTICKS pfGetCounterTicks`

[in] Function returning the current counter ticks

`EC_T_VOID *pvGetCounterTicksContext`

[in] Context passed into `GetCounterTicks`

`EC_T_UINT64 qwFrequency`

[in] Frequency in Hz used by the timer in `GetCounterTicks`

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

Param `pvContext`

[in] Arbitrarily application-defined parameter passed to callback

```
struct EC_T_PERF_MEAS_HISTOGRAM_PARMS
```

Public Members

`EC_T_DWORD dwBinCount`

[in] Amount of bins to use for the histogram

`EC_T_UINT64 qwMinTicks`

[in] Results below `qwMinTicks` are stored in the first bin

`EC_T_UINT64 qwMaxTicks`

[in] Results above `qwMaxTicks` are stored in the last bin

6.3.2 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).

Parameters

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

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 the 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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* 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)
- **pParms** – [in] New Master parameters

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack 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* pParms = (EC_T_INIT_MASTER_PARMS*) abyBuffer;
dwRes = emGetMasterParms(dwInstanceId, pParms, sizeof(abyBuffer));
pParms->wReserved = 1;
/* change Master initialization parameters */
dwRes = emSetMasterParms(dwInstanceId, pParms);

```

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 EtherCAT stack 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 do not match 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_ADS_IS_RUNNING* if the 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)

Recovers the bus from permanent frame loss situations.

Scans all connected slaves. Closes and opens ports on the network to rule out slaves which permanently discard frames. The Master notifies every slave port which permanently discards 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 EtherCAT stack isn't initialized or eCnfType is eCnfType_GenPreopENI or eCnfType_GenOpENI and link is disconnected
- *EC_E_INVALIDPARAM* if dwInstanceID is out of range or pParms is EC_NULL contains some values out of range
- *EC_E_LINK_DISCONNECTED* if link is disconnected
- *EC_E_FEATURE_DISABLED* if a configured feature is not included in the license key
- *EC_E_NOTSUPPORTED* if a configured feature is not supported (e.g not compiled in the library)
- *EC_E_CFGFILENOTFOUND* if the ENI file cannot be found
- *EC_E_WRONG_FORMAT* if format errors have been detected in the ENI or EEPROM in case of eCnfType_GenPreopENI or eCnfType_GenOpENI
- *EC_E_OEM_SIGNATURE_MISMATCH* if the OEM signature in the ENI file doesn't match the used OEM key
- *EC_E_ENI_ENCRYPTION_WRONG_VERSION* if the ENI encryption version is not supported (e.g. the library is too old)
- *EC_E_ENI_ENCRYPTED* if the ENI is encrypted and no OEM key has been set
- *EC_E_XML_CYCCMDS_MISSING* if the ENI doesn't contain cyclic commands
- *EC_E_XML_ALSTATUS_READ_MISSING* if the ENI doesn't contain any read AL status command
- *EC_E_XML_CYCCMDS_SIZEMISMATCH* if the size of the cyclic commands in the ENI mismatch
- *EC_E_XML_INVALID_INP_OFF* if some input offsets in the ENI are invalid
- *EC_E_XML_INVALID_OUT_OFF* if some output offsets in the ENI are invalid
- *EC_E_XML_INVALID_CMD_WITH_RED* if the ENI contains LRW commands and cable redundancy is configured
- *EC_E_XML_PREV_PORT_MISSING* if some previous port information are missing in the ENI
- *EC_E_XML_DC_CYCCMDS_MISSING* if the DC related cyclic commands are missing in the ENI
- *EC_E_XML_AOE_NETID_INVALID* if the ENI contains some invalid NetID

enum **EC_T_CNF_TYPE**

Values:

enumerator **eCnfType_Unknown**

- enumerator **eCnfType_Filename**
pbyCnfData: ENI filename to read
- enumerator **eCnfType_Data**
pbyCnfData: ENI data
- enumerator **eCnfType_Datadiag**
pbyCnfData: ENI data for diagnosis
- enumerator **eCnfType_GenPreopENI**
Generate ENI based on bus-scan result to get into PREOP state
- enumerator **eCnfType_GenPreopENIWithCRC**
Same as eCnfType_GenPreopENI with CRC protection
- enumerator **eCnfType_GenOpENI**
Generate ENI based on bus-scan result to get into OP state. The default PDO mapping read from the slaves is activated. See ETG2010 “SII Specification”, Table 14 “Structure Category TXPDO and RXPDO for each PDO”.
- enumerator **eCnfType_None**
Reset configuration
- enumerator **eCnfType_ConfigData**
pbyCnfData: Binary structured configuration
- enumerator **eCnfType_GenOpENINoStrings**
Generate ENI based on bus-scan result to get into OP state, does not read strings from EEPROM
- enumerator **eCnfType_FileByApp**
File access provided by user application, See [EC_T_CNF_FILEBYAPP_DESC](#)
- enumerator **eCnfType_GenEBI**
Generate EBI based on bus-scan result

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)

Param pvContext

[in] Arbitrarily application-defined parameter passed to callback

Return

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)

Param pvContext

[in] Arbitrarily application-defined parameter passed to callback

Return

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)

Param pvContext

[in] Arbitrarily application-defined parameter passed to callback

Param pbyReadData

[out] Pointer to the data read

Param dwReadLen

[in] Amount of bytes to be read by the EtherCAT stack (next part of the configuration file)

Param pdwNumOutData

[out] Amount of bytes written to pbyReadData by callback

Return

EC_E_NOERROR or error code

```
typedef EC_T_DWORD (*EC_PF_CNF_ERROR)(EC_T_VOID *pvContext)
```

Called by the EtherCAT stack instead of OsCfgFileError() within emConfigureNetwork(eCnfType_FileByApp)

Param pvContext

[in] Arbitrarily application-defined parameter passed to callback

Return

EC_E_NOERROR or error code

```
typedef 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)

Param pvContext

[in] Arbitrarily application-defined parameter passed to callback

Param bEof

[out] Indicates whether the end of the file has been reached (EC_TRUE if EOF, EC_FALSE otherwise).

Return*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)

```

(continues on next page)

(continued from previous page)

```

{
    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, &pbbyConfigData, &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 have 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` will 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
)
```

Registers a client on the EC-Master.

It must be called after configuration, otherwise the registration handle is lost. This function may not be called from within the JobTask's context.

Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pfnNotify** – [in] Notification callback function. This function will be called every time a state change occurs, an error occurs or a mailbox transfer terminates.
- **pCallerData** – [in] Pointer to a caller data area which will be passed to the client on every notification callback
- **pRegResults** – [out] Registration results, a pointer to a structure of type *EC_T_REGISTERRESULTS*

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARAM* if dwInstanceId is out of range or the output pointer is EC_NULL
- *EC_E_NOMEMORY* if some memory cannot be allocated

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

Param dwCode

[in] Notification code, see EC_NOTIFY_...

Param pParms

[in] Notification code depending data

```
struct EC_T_REGISTERRESULTS
```

Public Members

EC_T_DWORD dwClntId
[out] Client ID

EC_T_BYTE *pbyPDIn
[out] Pointer to process data input memory

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

EC_T_BYTE *pbyPDOut
[out] Pointer to process data output memory

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

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)
Unregister a client from the EtherCAT master.

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

Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwClntId** – [in] Client ID determined when registering with the master

Returns

EC_E_NOERROR or error code

6.3.12 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*).

By default the Master 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 the 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 the state of previously failed slaves. Transition to a lower state: The master changes to a 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 EtherCAT stack isn't initialized
- `EC_E_INVALIDPARAM` if `dwInstanceID` is out of range
- `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
- `EC_E_ADS_IS_RUNNING` if the ADS 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:

emIoCtl - 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 EtherCAT stack 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
- *EC_E_ADS_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:

emIoCtl - 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 EtherCAT stack 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 emExecJob

```

static EC_T_DWORD ecatExecJob (
    EC_T_USER_JOB eUserJob,
    EC_T_USER_JOB_PARAMS *pUserJobParams
)

```

```

EC_T_DWORD emExecJob (
    EC_T_DWORD dwInstanceId,
    EC_T_USER_JOB eUserJob,
    EC_T_USER_JOB_PARAMS *pUserJobParams
)

```

Execute or initiate the requested job.

To achieve maximum speed, this function is implemented non re-entrant. It is highly recommended that only one single task is calling all required jobs to run the stack. If multiple tasks are calling this function, the calls have to be synchronized externally. Calling it in a context that doesn't support operating system calls can lead to unpredictable behavior.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eUserJob** – [in] User requested job
- **pUserJobParams** – [in] Optional user job parameters

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARAM* 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 the ADS server is running

Brief job overview:

enum **EC_T_USER_JOB**

Values:

enumerator **eUsrJob_Undefined**

enumerator **eUsrJob_ProcessAllRxFrames**

enumerator **eUsrJob_SendAllCycFrames**

enumerator **eUsrJob_MasterTimer**

enumerator **eUsrJob_SendAcycFrames**

enumerator **eUsrJob_SendCycFramesByTaskId**

enumerator **eUsrJob_MasterTimerMinimal**

enumerator **eUsrJob_ProcessRxFramesByTaskId**

enumerator **eUsrJob_ProcessAcycRxFrames**

enumerator **eUsrJob_SwitchEoeFrames**

enumerator **eUsrJob_StartTask**

enumerator **eUsrJob_StopTask**

enumerator **eUsrJob_StampSendAllCycFrames**

enumerator **eUsrJob_StampSendCycFramesByTaskId**

enumerator **eUsrJob_SimulatorTimer**

enumerator **eUsrJob_MonitorTimer**

union **EC_T_USER_JOB_PARMS**

Public Members

EC_T_BOOL **bAllCycFramesProcessed**

EC_T_DWORD **dwNumFramesSent**

EC_T_DWORD **dwTaskIdToSend**

struct *EC_T_USER_JOB_PARMS::_SEND_CYCFRAME_BY_TASKID* **SendCycFramesByTaskId**

struct *EC_T_USER_JOB_PARMS::_PROCESS_RXFRAME_BY_TASKID* **ProcessRxFramesByTaskId**

struct *EC_T_USER_JOB_PARMS::_SWITCH_EOE_FRAMES* **SwitchEoeFrames**

struct *EC_T_USER_JOB_PARMS::_START_TASK* **StartTask**

struct *EC_T_USER_JOB_PARMS::_STOP_TASK* **StopTask**

struct **_PROCESS_RXFRAME_BY_TASKID**

Public Members

EC_T_BOOL **bCycFramesProcessed**

EC_T_DWORD **dwTaskId**

struct **_SEND_CYCFRAME_BY_TASKID**

Public Members

EC_T_DWORD **dwTaskId**

struct **_START_TASK**

Public Members

EC_T_DWORD **dwTaskId**

struct **_STOP_TASK**

Public Members

EC_T_DWORD **dwTaskId**

struct **_SWITCH_EOE_FRAMES**

Public Members

EC_T_DWORD **dwMaxPortsToProcess**

EC_T_DWORD **dwNumFramesProcessed**

Detailed job description:

1. ***eUsrJob_ProcessAllRxFrames***

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

If this flag is set to a value of EC_TRUE it indicates that all previously initiated cyclic frames (*eUsrJob_SendAllCycFrames*) are received and processed within this call. Not used if pUserJobParms is 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® SubDevice'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 sent within this call. Not used if pUserJobParms is 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 sent within this call. Not used if pUserJobParms is set to EC_NULL.

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

4. *eUsrJob_MasterTimer*

To trigger the MainDevice and SubDevice state machines as well as the mailbox handling this call has to be executed cyclically. The MainDevice 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 *emIoCtl* - *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);
```

6.3.18 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 EtherCAT stack 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);

```

6.3.19 emSetLicenseKey

static EC_T_DWORD **ecatSetLicenseKey** (const EC_T_CHAR *szLicenseKey)

```

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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* if dwInstanceId is out of range
- *EC_E_INVALIDSIZE* if the format of the license key is wrong. The correct length is 26, 53 or 56 characters.
- *EC_E_LICENSE_MISSING* if the license key doesn't match the MAC Address

emSetLicenseKey() Example

```

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

```

See also:

- *emInitMaster()*
- *emConfigureNetwork()*

6.3.20 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()*

6.3.21 emIoCtl

```
static EC_T_DWORD ecatIoCtl (
    EC_T_DWORD dwCode,
    const EC_T_VOID *const pbyInBuf,
    EC_T_DWORD dwInBufSize,
    EC_T_VOID *const pbyOutBuf,
    EC_T_DWORD dwOutBufSize,
    EC_T_DWORD *const pdwNumOutData
)
```

```
EC_T_DWORD emIoCtl (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwCode,
    const EC_T_VOID *const pbyInBuf,
    EC_T_DWORD dwInBufSize,
    EC_T_VOID *const pbyOutBuf,
    EC_T_DWORD dwOutBufSize,
    EC_T_DWORD *const pdwNumOutData
)
```

A generic control interface between the application, the EtherCAT stack and its Link Layers.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwCode** – [in] IOCTL code (EC_IOCTL...)
- **pbyInBuf** – [in] IOCTL input parameters
- **dwInBufSize** – [in] Size of IOCTL input parameters in bytes
- **pbyOutBuf** – [out] Buffer for IOCTL output
- **dwOutBufSize** – [in] Size of buffer at pbyOutBuf in bytes

- **pdwNumOutData** – [out] Amount of bytes written to pbyOutBuf by IOCTL. EC_NULL: amount not set by IOCTL.

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARAM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer
- *EC_E_NOMEMORY* if memory cannot be allocated
- *EC_E_ADS_IS_RUNNING* if the ADS server is running

6.3.22 emIoctl - EC_IOCTL_GET_PDMEMORYSIZE

EC_IOCTL_GET_PDMEMORYSIZE

Get the process data image size. This information may be used to provide process data image storage from outside the core. This IOCTL is to be called after network configuration.

Parameters

- **pbyInBuf** – [in] Should be set to EC_NULL
- **dwInBufSize** – [in] Should be set to 0
- **pbyOutBuf** – [out] Pointer to memory where the memory size information will be stored (type: *EC_T_MEMREQ_DESC*)
- **dwOutBufSize** – [in] Size of the output buffer in bytes
- **pdwNumOutData** – [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Returns

EC_E_NOERROR or error code

struct **EC_T_MEMREQ_DESC**

Public Members

EC_T_DWORD **dwPDOutSize**
Size of the output process data image

EC_T_DWORD **dwPDInSize**
Size of the input process data image

See also:

emIoctl()

6.3.23 emIoctl - EC_IOCTL_REGISTER_PDMEMORYPROVIDER

EC_IOCTL_REGISTER_PDMEMORYPROVIDER

This function call registers an external memory provider to the stack, this memory will be used to store process data. If no memory provider is registered the stack will internally allocate the necessary amount of memory. The function *EC_IOCTL_GET_PDMEMORYSIZE* should be executed to determine the amount of memory the stack needs to store process data values. An external memory provider may additionally supply some hooks to give the stack a possibility to synchronize memory access with the application. Also the memory provider has to be registered after configuring the network but prior to registering any client. Every client that registers with the stack will get back the memory pointers to PDOOut/PDIn data registered within this call.

Parameters

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

Returns

EC_E_NOERROR or error code

```
struct EC_T_MEMPROV_DESC
```

Public Members

EC_T_PVOID pvContext

Context pointer. This pointer is used every time 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 the function *EC_T_MEMPROV_DESC.pfPDOOutDataReadRequest* is called.

EC_T_DWORD dwPDOOutDataLength

Length of the output process data buffer

EC_T_PBYTE pbyPDInData

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 the function *EC_T_MEMPROV_DESC.pfPDInDataWriteRequest* is called.

EC_T_DWORD dwPDInDataLength

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 reading 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* pfPDOutDataReadRelease**

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* pfPDOutDataWriteRequest**

This function will be called cyclically within the process data transfer cycle prior to writing new data into the output process data buffer. If `EC_NULL` is set, the fixed buffer `EC_T_MEMPROV_DESC.pbyPDOutData` is used.

***EC_T_PFMEMREL* pfPDOutDataWriteRelease**

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 writing 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* pbyMasterRedPDOutData**

Pointer to the MasterRed output process data buffer (ACTIVE to INACTIVE)

***EC_T_DWORD* dwMasterRedPDOutDataLength**

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* pfMasterRedPDOutReadRequest**

This function will be called within the process data transfer cycle prior to reading data

***EC_T_PFMEMREL* pfMasterRedPDOutReadRelease**

This function will be called after all data has been read from the output process data buffer

***EC_T_PFMEMREQ* pfMasterRedPDOutWriteRequest**

This function will be called within the process data transfer cycle prior to reading data

***EC_T_PFMEMREL* pfMasterRedPDOutWriteRelease**

This function will be called after all data were read from the output process data buffer

***EC_T_PFMEMREQ* pfMasterRedPDInWriteRequest**

This function will be called within the process data transfer cycle prior to writing data

***EC_T_PFMEMREL* pfMasterRedPDInWriteRelease**

This function will be called after all data were written to the input process data buffer

***EC_T_PFMEMREQ* pfMasterRedPDInReadRequest**

This function will be called within the process data transfer cycle prior to writing data

EC_T_PFMEMREL **pfMasterRedPDInReadRelease**

This function will be called after all data were written to the input process data buffer

```
typedef EC_T_VOID (*EC_T_PFMEMREQ)(EC_T_PVOID pvContext, EC_T_DWORD dwTaskId,
EC_T_PBYTE *ppbyPDData)
```

Param pvContext

[in] Arbitrarily application-defined parameter passed to callback

Param 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.

Param 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)
```

Param pvContext

[in] Arbitrarily application-defined parameter passed to callback

Param dwTaskId

[in] Task ID of cyclic data transfer (ENI: Cyclic/TaskId)

See also:

- *emIoCtl - EC_IOCTL_GET_PDMEMORYSIZE*
- *emConfigureNetwork ()*
- *emRegisterClient ()*
- Feature Pack “MainDevice Redundancy”
- *emIoCtl ()*

6.3.24 emIoCtl - EC_IOCTL_REGISTER_CYCFRAME_RX_CB

EC_IOCTL_REGISTER_CYCFRAME_RX_CB

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

Parameters

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

Returns

EC_E_NOERROR or error code

struct **EC_T_CYCFRAME_RX_CBDESC**

Public Members

EC_T_VOID *pCallbackContext

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

EC_PF_CYCFRAME_RECV pfnCallback

[in] This function will be called after the cyclic frame is received, if there is more than one cyclic frame after the last frame. The application has to assure that these functions will not block.

typedef EC_T_VOID (***EC_PF_CYCFRAME_RECV**)(EC_T_DWORD dwTaskId, EC_T_VOID *pvContext)

Param dwTaskId

[in] Task id of the received cyclic frame

Param pvContext

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

See also:

emIoCtl()

6.3.25 emIoCtl - EC_IOCTL_ISLINK_CONNECTED**EC_IOCTL_ISLINK_CONNECTED**

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

Parameters

- **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.

Returns

EC_E_NOERROR or error code

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

See also:

emIoCtl()

6.3.26 emIoctl - EC_IOCTL_GET_LINKLAYER_MODE

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.

Parameters

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

Returns

EC_E_NOERROR or error code

struct **EC_T_LINKLAYER_MODE_DESC**

Public Members

EC_T_LINKMODE **eLinkMode**

[out] Operation mode of main interface

EC_T_LINKMODE **eLinkModeRed**

[out] Operation mode of redundancy interface

See also:

emIoctl()

6.3.27 emIoctl - EC_IOCTL_GET_CYCLIC_CONFIG_INFO

EC_IOCTL_GET_CYCLIC_CONFIG_INFO

Determine cyclic configuration details from ENI configuration file. It can be called only after configuring the network.

Parameters

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

Returns

EC_E_NOERROR or error code

struct **EC_T_CYC_CONFIG_DESC**

Public Members

EC_T_DWORD dwNumCycEntries
[out] Total number of cyclic entries

EC_T_DWORD dwTaskId
[out] Task ID of selected cyclic entry (ENI: Cyclic/TaskId)

EC_T_DWORD dwPriority
[out] Priority of selected cyclic entry

EC_T_DWORD dwCycleTime
[out] Cycle time of selected cyclic entry

See also:

emIoCtl()

6.3.28 emIoCtl - EC_IOCTL_IS_SLAVETOSLAVE_COMM_CONFIGURED

EC_IOCTL_IS_SLAVETOSLAVE_COMM_CONFIGURED

Determine if any slave to slave communication is configured.

Parameters

- **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.

Returns

EC_E_NOERROR or error code

See also:

emIoCtl()

6.3.29 emIoctl - 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);
```

See also:

emIoctl()

6.3.30 emIoctl - EC_LINKIOCTL_GET_ETHERNET_ADDRESS

Provides MAC addresses of main or red line.

emIoctl - 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

See also:

emIoctl()

6.3.31 emIoctl - EC_LINKIOCTL_GET_SPEED

emIoctl - 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

See also:

emIoctl()

6.3.32 emIoCtl - EC_LINKIOCTL_GET_PCI_INFO

Get current network adapter's PCI information

emIoCtl - 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_IDENTIFICATION **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_IDENTIFICATION**

Public Members

EC_T_DWORD **dwVendorId**
Vendor ID

EC_T_DWORD **dwDeviceId**
Device ID

EC_T_DWORD **dwReserved**
Reserved for future use

struct **EC_T_PCI_INFO_IOBAR**

Public Members

EC_T_UINT64 **qwBaseAddress**
I/O bar base address

EC_T_DWORD **dwLen**
Bar length

EC_T_DWORD **dwReserved**
Reserved for future use

struct **EC_T_PCI_INFO_MEMBAR**

Public Members

EC_T_UINT64 **qwBaseAddress**
Memory bar base address

EC_T_DWORD **dwLen**
Bar length

EC_T_DWORD **dwReserved**
Reserved for future use

struct **EC_T_PCI_INFO_INTERRUPT**

Public Members

EC_T_DWORD **dwIrq**
IRQ number

EC_T_CPUSET **CpuAffinity**
Reserved for future use

EC_T_DWORD **adwReserved[2]**
Reserved for future use

See also:

emIoCtl()

6.3.33 emIoCtl - EC_IOCTL_SET_CYCFRAME_LAYOUT

EC_IOCTL_SET_CYCFRAME_LAYOUT

Set the cyclic frames layout.

Parameters

- **pbyInBuf** – [in] Pointer to an *EC_T_CYCFRAME_LAYOUT* value containing the cyclic frame layout
- **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.

Returns

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 few 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

See also:

emIoCtl()

6.3.34 emIoCtl - EC_IOCTL_SET_MASTER_DEFAULT_TIMEOUTS

EC_IOCTL_SET_MASTER_DEFAULT_TIMEOUTS

Set master default timeouts.

Parameters

- **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

Returns

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

EC_T_DWORD **dwInitCmdRetry**

Timeout [ms] between retry sending an init-command

EC_T_DWORD **dwMbxCmd**

Timeout [ms] between retry sending an mailbox command

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: 30000 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()*
- *emIoCtl()*

6.3.35 emIoCtl - EC_IOCTL_SET_COPYINFO_IN_SENDCYCFRAMES**EC_IOCTL_SET_COPYINFO_IN_SENDCYCFRAMES**

Set copy info processed in either *eUsrJob_SendAllCycFrames* or in *eUsrJob_ProcessAllRxFrames*.

Parameters

- **pbyInBuf** – [in] Pointer to EC_T_BOOL. EC_TRUE: SendCycFrames, EC_FALSE: ProcessAllRxFrames.
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns

EC_E_NOERROR or error code

Default: Set by ProcessAllRxFrames.

See also:

emIoCtl()

6.3.36 emIoCtl - EC_IOCTL_SET_BUS_CYCLE_TIME**EC_IOCTL_SET_BUS_CYCLE_TIME**

Set bus cycle time [us] master parameter without calling *emInitMaster()* again.

Parameters

- **pbyInBuf** – [in] Pointer to value of EC_T_DWORD. Value may not be 0!
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns*EC_E_NOERROR* or error code

Implicitly recalculates Order Timeout and *EC_T_INIT_MASTER_PARMS::dwEcatCmdTimeout*.

See also:*emIoctl()*

6.3.37 emIoctl - EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES

EC_IOCTL_ADDITIONAL_VARIABLES_FOR_SPECIFIC_DATA_TYPES

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

Parameters

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

Returns*EC_E_NOERROR* or error code

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

See also:*emIoctl()*

6.3.38 emIoctl - EC_IOCTL_SLV_ALIAS_ENABLE

EC_IOCTL_SLV_ALIAS_ENABLE

Enables slave alias addressing for all slaves.

Parameters

- **pbyInBuf** – [in] Should be set to EC_NULL
- **dwInBufSize** – [in] Should be set to 0
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns*EC_E_NOERROR* or error code

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

See also:

`emIoctl()`

6.3.39 emIoctl - EC_IOCTL_SET_IGNORE_INPUTS_ON_WKC_ERROR

EC_IOCTL_SET_IGNORE_INPUTS_ON_WKC_ERROR

Set ignore inputs on WKC error.

Parameters

- **pbyInBuf** – [in] Pointer to value of EC_T_BOOL. EC_TRUE: Ignore inputs 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

Returns

EC_E_NOERROR or error code

Calling this IOCTL with EC_TRUE as parameter will ignore input data of cyclic commands on WKC error. By default input data are updated if WKC is non zero. 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.

See also:

`emIoctl()`

6.3.40 emIoctl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ERROR

EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ERROR

Set inputs to zero on WKC error.

Parameters

- **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

Returns

EC_E_NOERROR or error code

Calling this IOCTL with EC_TRUE as parameter will set inputs to zero on WKC error. By default input data are updated if WKC is non zero. 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.

See also:

`emIoctl()`

6.3.41 emIoctl - EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ZERO

EC_IOCTL_SET_ZERO_INPUTS_ON_WKC_ZERO

Set inputs to zero if WKC is zero.

Parameters

- **pbyInBuf** – [in] Pointer to value of EC_T_BOOL. EC_TRUE: Set inputs to zero if 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

Returns

`EC_E_NOERROR` or error code

Calling this IOCTL with EC_TRUE as parameter will set inputs to zero on WKC is zero. By default input data are ignored on WKC is zero and remain unchanged. 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 emIoctl - EC_IOCTL_SET_ZERO_INPUTS_ON_FRAME_LOSS

EC_IOCTL_SET_ZERO_INPUTS_ON_FRAME_LOSS

Set inputs to zero on frame loss.

Parameters

- **pbyInBuf** – [in] Pointer to value of EC_T_BOOL. EC_TRUE: Set inputs to zero on frame loss
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns

`EC_E_NOERROR` or error code

Calling this IOCTL with EC_TRUE as parameter will set inputs to zero on frame loss. By default input data are ignored on frame loss and remain unchanged.

See also:

`emIoctl()`

6.3.43 emIoctl - EC_IOCTL_SET_GENENI_ASSIGN_EEPROM_BACK_TO_EC

EC_IOCTL_SET_GENENI_ASSIGN_EEPROM_BACK_TO_EC

Enable or disable creation of “assign EEPROM back to ECAT” InitCmd if ENI generated based on bus-scan result.

Parameters

- **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

Returns

EC_E_NOERROR or error code

The InitCmd “assign EEPROM back to ECAT” in the ENI depends on the attribute “AssignToPdi” of the EEPROM tag in the SubDevice’s description within the ESI file. Because this attribute is not reflected in the SII in the SubDevice’s EEPROM, the MainDevice cannot know its value and inserts the InitCmd by default for legacy reasons.

See also:

emIoctl()

6.3.44 emIoctl - EC_IOCTL_SET_EOE_DEFERRED_SWITCHING_ENABLED

EC_IOCTL_SET_EOE_DEFERRED_SWITCHING_ENABLED

Enable or disable deferred EoE switching.

Parameters

- **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

Returns

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 SubDevices and EoE end point(s).

See also:

emIoctl()

6.3.45 emIoctl - EC_IOCTL_SET_MAILBOX_POLLING_CYCLES

EC_IOCTL_SET_MAILBOX_POLLING_CYCLES

This call changes the mailbox polling cycles.

Parameters

- **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`

Returns

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]

See also:

emIoctl()

6.3.46 emIoctl - EC_IOCTL_SET_MASTER_MAX_STATE

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.

Parameters

- **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`

Returns

EC_E_NOERROR or error code

See also:

- enum *EC_T_STATE*

- `emIoCtl()`

6.3.47 `emIoCtl - EC_IOCTL_SET_AUTO_ADJUST_MBX_STATE_COUNT_ENABLED`

`EC_IOCTL_SET_AUTO_ADJUST_MBX_STATE_COUNT_ENABLED`

This call specifies if the mailbox states count must be automatically adjusted according the mailbox state addresses in the ENI. This is needed if the ENI is inconsistent regarding the mailbox states, leading to Error 0x98130033 ENI: Inconsistent content. Default: Use ENI specified mailbox state count (`EC_FALSE`).

Parameters

- `pbyInBuf` – [in] Pointer to `EC_T_BOOL`
- `dwInBufSize` – [in] Size of the input buffer in bytes, e.g. `sizeof(EC_T_BOOL)`
- `pbyOutBuf` – [out] Should be set to `EC_NULL`
- `dwOutBufSize` – [in] Should be set to 0
- `pdwNumOutData` – [out] Should be set to `EC_NULL`

Returns

`EC_E_NOERROR` or error code

See also:

`emIoCtl()`

6.3.48 `emIoCtl - EC_IOCTL_ACTIVATE_VOE_RECV_FIFO`

`EC_IOCTL_ACTIVATE_VOE_RECV_FIFO`

Activates and set the size of the VoE receive FIFO.

Parameters

- `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`

Returns

`EC_E_NOERROR` or error code

See also:

`emIoCtl()`

6.3.49 emIoCtl - EC_IOCTL_SET_GEN_ENI_PARM

Identifier	Description
EC_GEN_ENI_PARM_ID_SLAVE_PREFIX	Prefix of SubDevices and their variables' names. By default the prefix is 'Slave'.

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 *eCnfType_GenPreopENI* or *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.

Parameters

- **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`

Returns

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 **GenEniParam**

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

EC_T_WORD wSlaveAddress

See also:

emIoCtl()

6.3.50 emIoctl - EC_IOCTL_REALLOC_MBX_QUEUE

EC_IOCTL_REALLOC_MBX_QUEUE

This call reallocates the number of queues of the different mailbox protocols.

Parameters

- **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`

Returns

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` **wMbxProtocols**

Combination of Mbx protocols `EC_MBX_PROTOCOL_COE...`

`EC_T_DWORD` **dwMaxMbxTferQueued**

Maximal number of transfers queued for the specified mailbox protocol of the specified slave

See also:

emIoctl()

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
)

```

Blocking function to retrieve consistent process data from outside the JobTask context.

This function requests a copy of the process data (stored in RAM). The actual memcpy operation is executed by the JobTask to ensure data consistency. While waiting for the copy to complete, the calling context blocks and repeatedly calls sleep with an interval of at least the cycle time or one millisecond, whichever is greater. The function returns either with the requested process data once the copy is finished, or when the specified timeout has expired.

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 is 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: The function is blocking and should be used carefully in time-sensitive contexts and may not be called from within the JobTask context.

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

Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC_TRUE: read output data, EC_FALSE: read input data
- **dwOffset** – [in] Byte offset in Process data to read from
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length [bytes]
- **dwTimeout** – [in] Timeout [ms]

Returns

EC_E_NOERROR or error code

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 [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_BYTE *pbyData,
    EC_T_DWORD dwDataBitLen,
    EC_T_DWORD dwTimeout
)

```

Writes a specific number of bits from a given buffer to the process image with a bit offset (synchronized).

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

Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bOutputData** – [in] EC_TRUE: write output data, EC_FALSE: write input data
- **dwBitOffsetPd** – [in] Bit offset in Process data image
- **pbyData** – [in] Buffer containing transferred data
- **dwDataBitLen** – [in] Buffer length [bit]
- **dwTimeout** – [in] Timeout [ms]. The timeout value must not be set to EC_NOWAIT.

Returns

EC_E_NOERROR or error code

See also:

emSetProcessData()

emSetProcessDataBits() Example

```

/* write bits to Process Data Image (synchronized with JobTask) */
EC_T_BYTE 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_BYTE *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_BYTE *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()*
- *emIoCtl - EC_IOCTL_GET_PDMEMORYSIZE*
- *emIoCtl - 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()*
- *emIoCtl - EC_IOCTL_GET_PDMEMORYSIZE*
- *emIoCtl - 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 emGetProcessVarInfoNumOf, emGetProcessVarInfoEx

```
static EC_T_DWORD ecatGetProcessVarInfoNumOf (
    EC_T_VAR_DIRECTION eVarDirection,
    EC_T_VAR_SOURCE eVarSource,
    EC_T_BOOL bFixedAddress,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD *pdwProcessVarInfoNumOf
)
```

```

EC_T_DWORD emGetProcessVarInfoNumOf (
    EC_T_DWORD dwInstanceID,
    EC_T_VAR_DIRECTION eVarDirection,
    EC_T_VAR_SOURCE eVarSource,
    EC_T_BOOL bFixedAddress,
    EC_T_WORD wSlaveAddress,
    EC_T_DWORD *pdwProcessVarInfoNumOf
)

```

Get process variables information.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eVarDirection** – [in] INPUTs, OUTPUTs, See *EC_T_VAR_DIRECTION* .
- **eVarSource** – [in] Slave, Master, See *EC_T_VAR_SOURCE* .
- **bFixedAddress** – [in] Use station address if EC_TRUE. Otherwise use AutoInc address.
- **wSlaveAddress** – [in] Slave address according to bFixedAddress
- **pdwProcessVarInfoNumOf** – [out] Process variables count

Returns

EC_E_NOERROR or error code

```

static EC_T_DWORD ecatGetProcessVarInfoEx (
    EC_T_VAR_DIRECTION eVarDirection,
    EC_T_VAR_SOURCE eVarSource,
    EC_T_BOOL bFixedAddress,
    EC_T_WORD wSlaveAddress,
    EC_T_PROCESS_VAR_INFO_EX *aoVarInfo,
    EC_T_DWORD dwMaxProcessVarInfoNumOf,
    EC_T_DWORD *pdwProcessVarInfoNumOf
)

```

```

EC_T_DWORD emGetProcessVarInfoEx (
    EC_T_DWORD dwInstanceID,
    EC_T_VAR_DIRECTION eVarDirection,
    EC_T_VAR_SOURCE eVarSource,
    EC_T_BOOL bFixedAddress,
    EC_T_WORD wSlaveAddress,
    EC_T_PROCESS_VAR_INFO_EX *aoVarInfoEx,
    EC_T_DWORD dwMaxVarInfoCnt,
    EC_T_DWORD *pdwVarInfoCnt
)

```

Get process variables information.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **eVarDirection** – [in] INPUTs, OUTPUTs, See *EC_T_VAR_DIRECTION* .
- **eVarSource** – [in] Slave, Master, See *EC_T_VAR_SOURCE* .
- **bFixedAddress** – [in] Use station address if EC_TRUE. Otherwise use AutoInc address.
- **wSlaveAddress** – [in] Slave address according to bFixedAddress
- **aoVarInfoEx** – [out] The read process variable extended information entries

- **dwMaxVarInfoCnt** – [in] Maximum number of variables that can be stored at aoVarInfoEx
- **pdwVarInfoCnt** – [out] Number process variable entries that have been stored in aoVarInfoEx

Returns

EC_E_NOERROR or error code

The following example demonstrates how to get all process data variables:

emGetProcessVarInfoEx() Example

```

EC_T_DWORD dwProcessVarInfoNumOf = 0;
dwRes = emGetProcessVarInfoNumOf(dwInstanceId, eVarDirection_All, eVarSource_All, &dwProcessVarInfoNumOf);
/* ... */
EC_T_PROCESS_VAR_INFO_EX* aoVarInfoEx = (EC_T_PROCESS_VAR_INFO_EX*)OsMalloc(dwProcessVarInfoNumOf * sizeof(EC_T_PROCESS_VAR_INFO_EX));
dwRes = emGetProcessVarInfoEx(dwInstanceId, eVarDirection_All, eVarSource_All, EC_FALSE, 0, aoVarInfoEx, dwProcessVarInfoNumOf, EC_NULL);
/* ... */
OsSafeFree(aoVarInfoEx);

```

EC_T_VAR_DIRECTION

enum **EC_T_VAR_DIRECTION**

Values:

enumerator **eVarDirection_Undefined**
Undefined Direction

enumerator **eVarDirection_INPUT**
INPUTs

enumerator **eVarDirection_OUTPUT**
OUTPUTs

enumerator **eVarDirection_All**
INPUTs and OUTPUTs

enumerator **eVarDirection_BCcppDummy**

EC_T_VAR_SOURCE

enum **EC_T_VAR_SOURCE**

Values:

enumerator **eVarSource_Undefined**
Undefined Source

enumerator **eVarSource_All**
Slaves and Master/Monitor/Simulator

enumerator **eVarSource_AllSlaves**
All Slaves

```

enumerator eVarSource_Slave
    Slave

enumerator eVarSource_Master
    Master

enumerator eVarSource_Monitor
    Monitor

enumerator eVarSource_Simulator
    Simulator

enumerator eVarSource_BCppDummy

```

6.4.13 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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found

See also:

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

emGetSlaveInpVarInfoNumOf() Example

```

/* get number of input variables of specific slave */
EC_T_WORD numOfVars = 0;
dwRes = emGetSlaveInpVarInfoNumOf(dwInstanceId, EC_TRUE, 1004, &numOfVars);

```

6.4.14 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 a specific slave.

Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wNumOfVarsToRead** – [in] Number process variable entries that have been stored in pSlaveProcVarInfoEntries
- **pSlaveProcVarInfoEntries** – [out] The read process variable information entries
- **pwReadEntries** – [out] The number of read process variable information entries

Returns

EC_E_NOERROR or error code

```
struct EC_T_PROCESS_VAR_INFO
```

Public Members

```
EC_T_CHAR szName[MAX_PROCESS_VAR_NAME_LEN]
    [out] Name of the found process variable
```

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

EC_T_WORD wFixedAddr

[out] Station address of the slave that is owner of this variable

EC_T_INT nBitSize

[out] Size in bits of the found process variable

EC_T_INT nBitOffs

[out] Bit offset in the process data image

EC_T_BOOL bIsInputData

[out] Determines whether the found process variable is an input variable or an output variable

MAX_PROCESS_VAR_NAME_LEN

Maximum length of a process variable name: 71 characters

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);

```

6.4.15 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 EtherCAT stack 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 EcType.h, DEFTYPE_BOOLEAN.

```
EC_T_WORD wFixedAddr
```

[out] Station address of the slave that is owner of this variable

```
EC_T_INT nBitSize
```

[out] Size in bits of the found process variable

```
EC_T_INT nBitOffs
```

[out] Bit offset in the process data image

```
EC_T_BOOL bIsInputData
```

[out] Determines whether the found process variable is an input variable or an output variable

```
EC_T_WORD wIndex
```

[out] Object index

```
EC_T_WORD wSubIndex
```

[out] Object sub index

```
EC_T_WORD wPdoIndex
```

[out] Index of PDO (process data object)

```
EC_T_WORD wWkcStateDiagOffs
```

[out] Bit offset in the diagnostic image (API GetDiagnosisImagePtr)

```
EC_T_WORD wMasterSyncUnit
```

[out] Master Sync Unit ID (ENI: Slave/ProcessData/RxPdo[1..4]@Su, Slave/ProcessData/TxPdo[1..4]@Su, comment at Cyclic/Frame/Command)

EC_T_DWORD dwTaskId
[out] ID of task where process variable is located

EC_T_CYC_COPY_INFO CopyInfo
[out] Copy Info if applied to the variable

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

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);

```

6.4.16 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, 1002, &numOfVars);
```

See also:

- [emGetSlaveOutpVarInfo\(\)](#)
- [emGetSlaveOutpVarInfoEx\(\)](#)

6.4.17 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

6.4.18 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

6.4.19 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 sub index
- **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

6.4.20 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 sub index
- **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

6.4.21 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

6.4.22 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

6.4.23 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

6.4.24 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

6.4.25 emLinkInputVarByName

```
static EC_T_DWORD ecatLinkInputVarByName (
    const EC_T_CHAR *szVarName,
    EC_T_LINK_VAR_INFO *pLinkVarInfo
)
EC_T_DWORD emLinkInputVarByName (
    EC_T_DWORD dwInstanceID,
    const EC_T_CHAR *szVarName,
    EC_T_LINK_VAR_INFO *pLinkVarInfo
)
    Link application variable to input process image. Input process variable is given by its name.
```

Parameters

- **dwInstanceID** – [in] Master instance ID
- **szVarName** – [in] Name of variable
- **pLinkVarInfo** – [in] Information about linked variable

Returns

EC_E_NOERROR or an error code

```
typedef EC_T_VOID (*EC_PF_LINK_VAR_CONVERTER)(const EC_T_LINK_VAR_INFO *pApplVarInfo,
EC_T_BYTE *pbyProcessImage)
```

Param pApplVarInfo

[in] Information about application variable

Param pbyProcessImage

[in] Pointer to input resp. output process image

```
struct EC_T_LINK_VAR_INFO
```

Public Members

EC_T_VOID ***pvContext**

[in] Arbitrarily application-defined parameter

EC_T_BYTE ***pbyApplVar**

[in] Pointer to application variable where process variable should be linked to

EC_T_WORD **wApplVarType**

[in] Type of application variable

EC_T_WORD **wApplVarBitSize**

[in] Size of variable (in bits)

EC_T_DWORD **dwApplVarBitOffs**

[in] Bit offset of variable

EC_PF_LINK_VAR_CONVERTER **pfnConverter**

[in] Callback function to manipulate the input when reading from process image resp. the output when writing to process image (could be EC_NULL)

EC_T_PROCESS_VAR_INFO_EX ***pProcVarInfoEx**

[out] Extended information about process variable

6.4.26 emLinkInputVarByObject

```
static EC_T_DWORD ecatLinkInputVarByObject (
```

```
    EC_T_BOOL bFixedAddressing,
```

```
    EC_T_WORD wSlaveAddress,
```

```
    EC_T_WORD wIndex,
```

```
    EC_T_WORD wSubIndex,
```

```
    EC_T_LINK_VAR_INFO *pLinkVarInfo
```

```
)
```

```
EC_T_DWORD emLinkInputVarByObject (
```

```
    EC_T_DWORD dwInstanceID,
```

```
    EC_T_BOOL bFixedAddressing,
```

```
    EC_T_WORD wSlaveAddress,
```

```
    EC_T_WORD wIndex,
```

```
    EC_T_WORD wSubIndex,
```

```
    EC_T_LINK_VAR_INFO *pLinkVarInfo
```

```
)
```

Link application variable to input process image. Input process variable is given by its object index and subindex

of a specific slave.

Parameters

- **dwInstanceID** – [in] Master instance ID
- **bFixedAddressing** – [in] Use station address if EC_TRUE. Otherwise use AutoInc address.
- **wSlaveAddress** – [in] Slave address according to bFixedAddressing
- **wIndex** – [in] Object index
- **wSubIndex** – [in] Object subindex
- **pLinkVarInfo** – [in] Information about linked variable

Returns

EC_E_NOERROR or an error code

See also:

EC_T_LINK_VAR_INFO

6.4.27 emLinkOutputVarByName

```
static EC_T_DWORD ecatLinkOutputVarByName (
    const EC_T_CHAR *szVarName,
    EC_T_LINK_VAR_INFO *pLinkVarInfo
)
```

```
EC_T_DWORD emLinkOutputVarByName (
    EC_T_DWORD dwInstanceID,
    const EC_T_CHAR *szVarName,
    EC_T_LINK_VAR_INFO *pLinkVarInfo
)
```

Link application variable to output process image. Output process variable is given by its name.

Parameters

- **dwInstanceID** – [in] Master instance ID
- **szVarName** – [in] Name of variable
- **pLinkVarInfo** – [in] Information about linked variable

Returns

EC_E_NOERROR or an error code

See also:

EC_T_LINK_VAR_INFO

6.4.28 emLinkOutputVarByObject

```
static EC_T_DWORD ecatLinkOutputVarByObject (
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wIndex,
    EC_T_WORD wSubIndex,
    EC_T_LINK_VAR_INFO *pLinkVarInfo
)
```

```
EC_T_DWORD emLinkOutputVarByObject (
    EC_T_DWORD dwInstanceID,
    EC_T_BOOL bFixedAddressing,
    EC_T_WORD wSlaveAddress,
    EC_T_WORD wIndex,
    EC_T_WORD wSubIndex,
    EC_T_LINK_VAR_INFO *pLinkVarInfo
)
```

Link application variable to output process image. Output process variable is given by its object index and subindex of a specific slave.

Parameters

- **dwInstanceID** – [in] Master instance ID
- **bFixedAddressing** – [in] Use station address if EC_TRUE. Otherwise use AutoInc address.
- **wSlaveAddress** – [in] Slave address according to bFixedAddressing
- **wIndex** – [in] Object index
- **wSubIndex** – [in] Object subindex
- **pLinkVarInfo** – [in] Information about linked variable

Returns

EC_E_NOERROR or an error code

See also:

EC_T_LINK_VAR_INFO

6.4.29 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_SIZEMISMATCH*.

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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARAM* if dwInstanceID is out of range
- *EC_E_NOTSUPPORTED* if eCycFrameLayout_FIXED is configured

6.4.30 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 located 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);

```

6.4.31 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 */

```

6.4.32 EC_GET_FRM_DWORD

EC_GET_FRM_DWORD (ptr)

Reads a value of type EC_T_DWORD (32 bit) at given pointer. The value is swapped on big endian systems.

Parameters

- **ptr** – [in] Source buffer

Returns

EC_T_DWORD value (32 bit) from buffer

```

EC_T_BYTE byFrame[] = {0x01, 0xF4, 0xDD, 0x85, 0x03, 0x00, 0x60, 0xC1, 0x00};
EC_T_DWORD dwResult = 0;

dwResult = EC_GET_FRM_DWORD(byFrame);
/* dwResult is 0x85DDF401 on little endian systems */

dwResult = EC_GET_FRM_DWORD(byFrame + 5);
/* dwResult is 0x00C16000 on little endian systems */

dwResult = EC_GET_FRM_DWORD(byFrame + 2);
/* dwResult is 0x000385DD on little endian systems */

```

6.4.33 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 */
```

6.4.34 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 ... */
```

6.4.35 EC_SET_FRM_DWORD

EC_SET_FRM_DWORD (ptr, dw)

Writes a value of type EC_T_DWORD (32 bit) at given pointer. The value is swapped on big endian systems.

Parameters

- **ptr** – [in] Destination buffer
- **dw** – [in] 32 bit value

```
EC_T_BYTE byFrame[32];

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

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

6.4.36 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 ... */
```

6.4.37 EC_COPYBITS

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

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

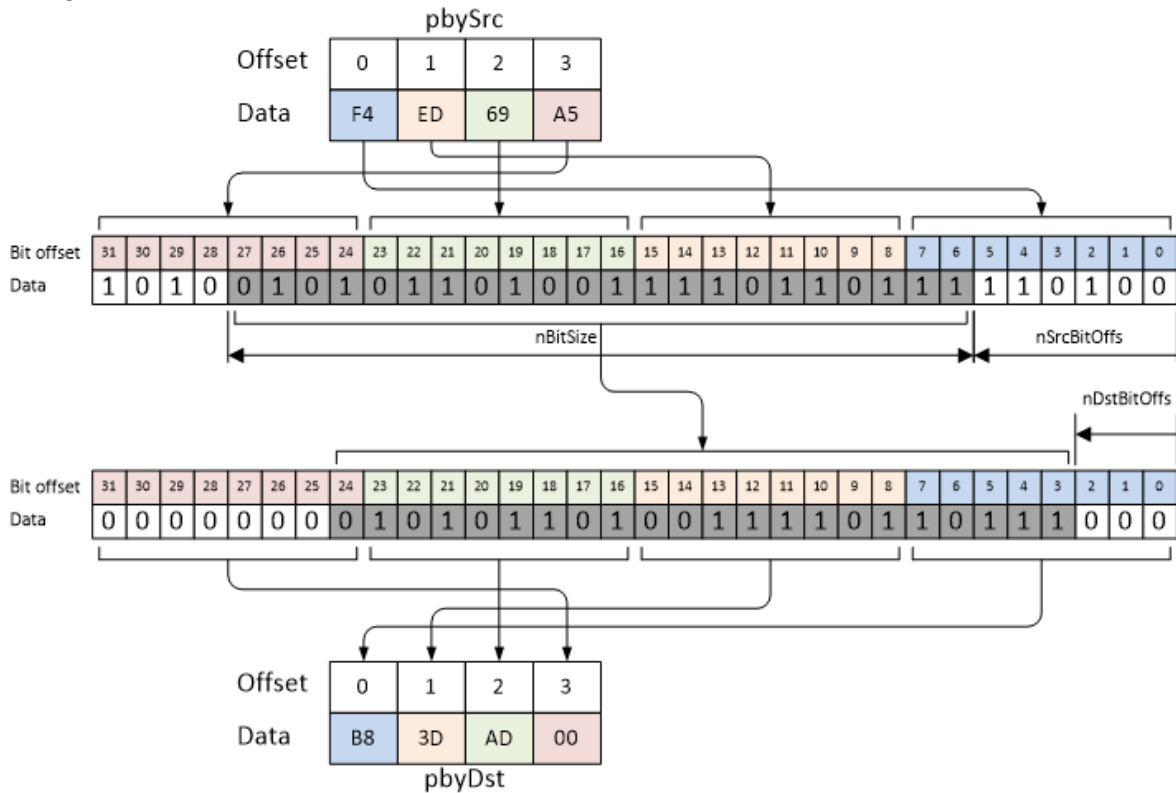
Note: The memory buffers must be allocated before. The buffers must be big enough to hold the block starting at the given offsets! The buffers are not checked for overrun.

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 */
```

6.4.38 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));
```

6.4.39 EC_GETBITS

EC_GETBITS (pbySrcBuf, pbyDstData, nSrcBitOffs, nBitSize)

Reads a given number of bits from source buffer starting at given bit offset to destination buffer.

Note: This function should only be used to get bit-aligned data. For byte-aligned data the corresponding functions should be used.

Parameters

- **pbySrcBuf** – [in] Source buffer to be copied
- **pbyDstData** – [out] Destination buffer where data is copied to
- **nSrcBitOffs** – [in] Source bit offset where data is copied from
- **nBitSize** – [in] Bit count to be copied

See also:

- *EC_GET_FRM_WORD*
- *EC_GET_FRM_DWORD*
- *EC_GET_FRM_QWORD*

6.4.40 EC_SETBITS

EC_SETBITS (pbyDstBuf, pbySrcData, nDstBitOffs, nBitSize)

Writes a given number of bits from source data starting at first bit to destination buffer at given bit offset.

Note: This function should only be used to set bit-aligned data. For byte-aligned data the corresponding functions should be used.

Parameters

- **pbyDstBuf** – [out] Destination buffer where data is copied to
- **pbySrcData** – [in] Source buffer to be copied, starting with first bit
- **nDstBitOffs** – [in] Destination bit offset where data is copied to
- **nBitSize** – [in] Bit count to be copied

See also:

- `EC_SET_FRM_WORD`
- `EC_SET_FRM_DWORD`
- `EC_SET_FRM_QWORD`

6.4.41 EC_COPYBIT

EC_COPYBIT (pbyBuf, nBitOffs, bVal)

Copy a boolean bit value into the buffer.

Parameters

- **pbyBuf** – Destination buffer.
- **nBitOffs** – Bit offset to write.
- **bVal** – Boolean bit value.

6.4.42 EC_TESTBIT

EC_TESTBIT (pbyBuf, nBitOffs)

Test whether a bit in the buffer is set.

Parameters

- **pbyBuf** – Source buffer.
- **nBitOffs** – Bit offset to test.

Returns

EC_TRUE if bit is set, otherwise EC_FALSE.

6.4.43 EC_SETBIT

EC_SETBIT (pbyBuf, nBitOffs)

Set a single bit in the buffer.

Parameters

- **pbyBuf** – Destination buffer.
- **nBitOffs** – Bit offset to set.

6.4.44 EC_CLRBIT

EC_CLRBIT (pbyBuf, nBitOffs)

Clear a single bit in the buffer.

Parameters

- **pbyBuf** – Destination buffer.
- **nBitOffs** – Bit offset to clear.

6.5 Generic notification interface

One of the parameters the client has to set when registering with the EtherCAT® MainDevice is a generic notification callback function (`emNotify()`). The MainDevice calls this function every time an 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 MainDevice, sending raw commands). In such cases the behavior is undefined.

This callback function is usually called in the context of the EtherCAT® MainDevice 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® MainDevice 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® MainDevice the client has to determine a generic notification callback function. The MainDevice 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 sending EtherCAT® commands (e.g. initiation of mailbox transfers, starting/stopping the MainDevice, 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® MainDevice 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_NOTIFYPARMS *pParms)
```

```
struct EC_T_NOTIFYPARMS
```

Data structure filled with detailed information about the according notification.

Public Members

```
EC_T_VOID *pCallerData
```

[in] Client depending caller data parameter. This pointer is one of the parameters when the client registers.

```
EC_T_BYTE *pbyInBuf
```

[in] Notification input parameters

```
EC_T_DWORD dwInBufSize
```

[in] Size of notification input parameters in bytes

```
EC_T_BYTE *pbyOutBuf
```

[out] Buffer for notification output (result)

```
EC_T_DWORD dwOutBufSize
```

[in] Size of buffer at pbyOutBuf in bytes

EC_T_DWORD *pdwNumOutData

[out] Amount of bytes written to pbyOutBuf by notification. EC_NULL: amount not set by notification.

6.5.2 emNotify - EC_NOTIFY_STATECHANGED

Notification about a change in the MainDevice'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 MainDevice operational state
- dwInBufSize: [in] Size of the input buffer provided at pbyInBuf in bytes
- pbyOutBuf: [out] Should be set to EC_NULL
- dwOutBufSize: [in] Should be set to 0
- pdwNumOutData: [out] Should be set to EC_NULL

struct **EC_T_STATECHANGE**

Public Members

EC_T_STATE **oldState**

Old operational state

EC_T_STATE **newState**

New operational state

See also:

emIoctl - 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 MainDevice Redundancy Notifications

See also:

Feature Pack “MainDevice 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 emIoctl - 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_REFCLOCK_PRESENCE* (default Off)
- *EC_NOTIFY_DCX_SYNC* (default On)
- *EC_NOTIFY_HC_DETECTADDDGROUPS* (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)

EC_IOCTL_SET_NOTIFICATION_ENABLED

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

Parameters

- **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*

Returns

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 emIoctl - EC_IOCTL_GET_NOTIFICATION_ENABLED

EC_IOCTL_GET_NOTIFICATION_ENABLED

The enabled state of notifications can be retrieved using *EC_IOCTL_GET_NOTIFICATION_ENABLED*.

Parameters

- **pbyInBuf** – [in] Pointer to *EC_T_GET_NOTIFICATION_ENABLED_PARMS*
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Pointer to EC_T_BOOL to carry out current enable set
- **dwOutBufSize** – [in] Size of the output buffer provided at pbyOutBuf in bytes
- **pdwNumOutData** – [out] Pointer to EC_T_DWORD. Amount of bytes written to the output buffer.

Returns

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:

emIoctl - EC_IOCTL_SET_NOTIFICATION_ENABLED

6.6 SubDevice 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 the 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 number 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 the 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 wAutoIncAddress = 0xFFFB;
EC_T_DWORD dwSlavePos = emGetSlaveIdAtPosition(dwInstanceId, wAutoIncAddress);
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 *Slave device state's*
- **dwTimeout** – [in] Timeout [ms]. This function will block until the requested state is reached or the timeout elapsed. If the timeout value is set to EC_NOWAIT the function will return immediately.

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized or denies the requested state, see comments below
- *EC_E_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_TIMEOUT* if dwTimeout elapsed during the API call
- *EC_E_BUSY* if the EtherCAT stack cannot execute the request at this time, the function has to be called at a later time
- *EC_E_NOTREADY* if the working counter was not set when requesting the slave's state (slave may not be connected or did not respond)
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

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 *Slave device state's*
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized or denies the requested state, see comments below
- *EC_E_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 EtherCAT stack 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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceId is out of range or the output pointers are EC_NULL
- *EC_E_SLAVE_NOT_PRESENT* if the slave is not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found

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

```

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 the slave is currently connected to the bus, EC_FALSE if not

Returns

- `EC_E_NOERROR` if successful
- `EC_E_INVALIDSTATE` if EtherCAT stack isn't initialized
- `EC_E_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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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 emIoctl - EC_IOCTL_SET_MBX_RETRYACCESS_PERIOD

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.

Parameters

- **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

Returns

`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 SubDevice changed its EtherCAT® state. This notification is disabled by default.

emNotify - EC_NOTIFY_SLAVE_STATECHANGED

Parameter

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

```
struct EC_T_SLAVE_STATECHANGED_NOTIFY_DESC
```

Public Members

EC_T_SLAVE_PROP **SlaveProp**
Slave properties

EC_T_STATE **newState**
New slave state

See also:

emIoCtl - *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:

emIoCtl - *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_NOTIFY_DESC*
- *dwInBufSize*: [in] Size of the input buffer provided at *pbyInBuf* in bytes
- *pbyOutBuf*: [out] Should be set to *EC_NULL*
- *dwOutBufSize*: [in] Should be set to 0
- *pdwNumOutData*: [out] Should be set to *EC_NULL*

struct **EC_T_SLAVES_STATECHANGED_NOTIFY_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 EtherCAT stack 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* if another transfer request is already pending or the master or the corresponding slave is currently changing its operational state
- *EC_E_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC_E_INVALIDSIZE* if the size of the complete command does not fit into a single Ethernet frame. The maximum amount of data to transfer must not exceed 1486 bytes.

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_NOTIFY_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 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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* if dwInstanceId is out of range or the command is not supported or the timeout value is set to EC_NOWAIT
- *EC_E_SLAVE_NOT_PRESENT* if the slave is not present
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC_E_TIMEOUT* if dwTimeout elapsed during the API call
- *EC_E_BUSY* if another transfer request is already pending or the master or the corresponding slave is currently changing its operational state
- *EC_E_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC_E_INVALIDSIZE* if the size of the complete command does not fit into a single Ethernet frame. The maximum amount of data to transfer must not exceed 1486 bytes.

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_NTIFY_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 transfered data
- **wLen** – [in] Number of bytes to receive

Returns

- `EC_E_NOERROR` if successful
- `EC_E_INVALIDSTATE` if EtherCAT stack 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_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 SubDevice register transfer is completed.

emNotify - EC_NOTIFY_SLAVE_REGISTER_TRANSFER

Parameter

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

struct `EC_T_SLAVEREGISTER_TRANSFER_NOTIFY_DESC`

Public Members

`EC_T_DWORD dwTferId`

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

`EC_T_DWORD dwResult`

Result of Slave register transfer

`EC_T_BOOL bRead`

`EC_TRUE`: Read register, `EC_FALSE`: Write register transfer

`EC_T_WORD wFixedAddr`

Station address of slave

`EC_T_WORD wRegisterOffset`

Register offset

`EC_T_WORD wLen`

Length of slave register transfer

`EC_T_BYTE *pbyData`

Pointer to the data read

`EC_T_WORD wWkc`

Received working counter

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 a slave.

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

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wEEPROMStartOffset** – [in] Word address to start EEPROM read from
- **pwReadData** – [in] Pointer to EC_T_WORD array to carry the read data
- **dwReadLen** – [in] Size of the EC_T_WORD array provided at pwReadData (in EC_T_WORDS)
- **pdwNumOutData** – [out] Pointer to EC_T_DWORD carrying actually read data (in EC_T_WORDS) after completion
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Returns

EC_E_NOERROR or error code

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,
    7 /* 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 an EEPROM data read operation from a slave and returns immediately.

An `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

emReadSlaveEEPROMReq() 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 data to write
- **dwWriteLen** – [in] Size of Write Data WORD array (in WORDS)
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Returns

EC_E_NOERROR or error code

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 an EEPROM data write operation from a 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. An 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_NOTIFY_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 returns 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_NOTIFY_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.

An 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. An `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_NOTIFY_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 SubDevice EEPROM operation is completed.

emNotify - EC_NOTIFY_EEPROM_OPERATION

Parameter

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

```
struct EC_T_EEPROM_OPERATION_NOTIFY_DESC
```

Public Members

`EC_T_DWORD dwTferId`

Transfer ID. For every new EEPROM operation a unique ID has to be assigned. This ID can be used after completion to identify the transfer.

`EC_T_EEPROM_OPERATION_TYPE eType`

Type of EEPROM operation

`EC_T_DWORD dwResult`

Result of EEPROM operation

`EC_T_SLAVE_PROP SlaveProp`

Slave properties

```
union _EC_T_EEPROM_OPERATION_NOTIFY_DESC_RESULT
```

```
struct _EC_T_EEPROM_OPERATION_NOTIFY_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_NOTIFY_DESC_RESULT_READ
```

Public Members

EC_T_WORD **wEEPROMStartOffset**

Start address of EEPROM operation. Given by API.

EC_T_WORD ***pwData**

Pointer to WORD array containing the data. Given by API.

EC_T_DWORD **dwReadLen**

Number of Words to be read. Given by API.

EC_T_DWORD **dwNumOutData**

Number of Words actually read from EEPROM

struct **_EC_T_EEPROM_OPERATION_NOTIFY_DESC_RESULT_WRITE**

Public Members

EC_T_WORD **wEEPROMStartOffset**

Start address of EEPROM operation. Given by API.

EC_T_WORD ***pwData**

Pointer to WORD array containing the data. Given by API.

EC_T_DWORD **dwWriteLen**

Number of Words to be written. Given by API.

enum **EC_T_EEPROM_OPERATION_TYPE**

Values:

enumerator **eEEPROMOp_Unknown**

Unknown EEPROM operation, only for internal use

enumerator **eEEPROMOp_Assign**

Assign slave EEPROM operation, used by emAssignSlaveEEPROMReq

enumerator **eEEPROMOp_Active**

Active slave EEPROM operation, used by emActiveSlaveEEPROMReq

enumerator **eEEPROMOp_Read**

Read slave EEPROM operation, used by emReadSlaveEEPROMReq

enumerator **eEEPROMOp_Write**

Write slave EEPROM operation, used by emWriteSlaveEEPROMReq

enumerator **eEEPROMOp_Reload**

Reload slave EEPROM operation, used by emReloadSlaveEEPROMReq

enumerator **eEEPROMOp_Reset**

Reset slave EEPROM operation, used by emResetSlaveController

6.6.31 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 the slave is 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 emIoctl - EC_IOCTL_ALL_SLAVES_MUST_REACH_MASTER_STATE

EC_IOCTL_ALL_SLAVES_MUST_REACH_MASTER_STATE

Specifies if all the slaves must reach the requested master state.

Parameters

- **pbyInBuf** – [in] Pointer to EC_T_BOOL variable. If set to EC_TRUE all slaves must reach the master requested state, if set to EC_FALSE the master can reach the requested state even if some slaves are missing or cannot reach the requested state.
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns

EC_E_NOERROR or error code

Missing mandatory SubDevices will be signaled by *emNotify* - *EC_NOTIFY_SLAVE_PRESENCE*. SubDevices that cannot reach the requested MainDevice state will be signaled 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 dwPdOffsIn3**
[out] 3rd sync unit process input data bit offset (from ENI file)
- EC_T_DWORD dwPdSizeIn3**
[out] 3rd sync unit process input data bit size (from ENI file)
- EC_T_DWORD dwPdOffsOut3**
[out] 3rd sync unit process output data bit offset (from ENI file)
- EC_T_DWORD dwPdSizeOut3**
[out] 3rd sync unit process output data bit size (from ENI file)
- EC_T_DWORD dwPdOffsIn4**
[out] 4th sync unit process input data bit offset (from ENI file)
- EC_T_DWORD dwPdSizeIn4**
[out] 4th sync unit process input data bit size (from ENI file)
- EC_T_DWORD dwPdOffsOut4**
[out] 4th sync unit process output data bit offset (from ENI file)
- EC_T_DWORD dwPdSizeOut4**
[out] 4th sync unit process output data bit size (from ENI file)
- EC_T_DWORD dwMbxSupportedProtocols**
[out] Mailbox protocols supported by the slave (from ENI file). Combination of *Supported mailbox protocols* flags.
- EC_T_DWORD dwMbxOutSize**
[out] Mailbox output byte size (from ENI file)
- EC_T_DWORD dwMbxInSize**
[out] Mailbox input byte size (from ENI file)
- EC_T_DWORD dwMbxOutSize2**
[out] Bootstrap mailbox output byte size (from ENI file)
- EC_T_DWORD dwMbxInSize2**
[out] Bootstrap mailbox input byte size (from ENI file)
- EC_T_BOOL bDcSupport**
[out] Slave supports DC (from ENI file)
- EC_T_WORD wNumProcessVarsInp**
[out] Number of input process data variables (from ENI file)
- EC_T_WORD wNumProcessVarsOutp**
[out] Number of output process data variables (from ENI file)

EC_T_WORD wPrevStationAddress

[out] Station address of the previous slave (from ENI file)

EC_T_WORD wPrevPort

[out] Connected port of the previous slave (from ENI file)

EC_T_WORD wIdentifyAdo

[out] ADO used for identification command (from ENI file)

EC_T_WORD wIdentifyData

[out] Identification value to be validated (from ENI file)

EC_T_BYTE byPortDescriptor

[out] Port descriptor (ESC register 0x0007) (from ENI file)

EC_T_WORD wWkcStateDiagOffsIn[EC_CFG_SLAVE_PD_SECTIONS]

[out] Offset of WkcState bit in diagnosis image (ENI: ProcessData/Recv[1..4]/BitStart): 0xFFFFFFFF = offset not available. WkcState bit values: 0 = Data valid, 1 = Data invalid.

EC_T_WORD wWkcStateDiagOffsOut[EC_CFG_SLAVE_PD_SECTIONS]

[out] Offset of WkcState bit in diagnosis image (ENI: ProcessData/Send[1..4]/BitStart): 0xFFFFFFFF = offset not available. WkcState bit values: 0 = Data valid, 1 = Data invalid.

EC_T_WORD awMasterSyncUnitIn[EC_CFG_SLAVE_PD_SECTIONS]

[out] Sync Unit (ENI: ProcessData/TxPdo[1..4]@Su)

EC_T_WORD awMasterSyncUnitOut[EC_CFG_SLAVE_PD_SECTIONS]

[out] Sync Unit (ENI: ProcessData/RxPdo[1..4]@Su)

EC_T_BOOL bDisabled

[out] Slave disabled by API SetSlaveDisabled / SetSlavesDisabled.

EC_T_BOOL bDisconnected

[out] Slave disconnected by API SetSlaveDisconnected / SetSlavesDisconnected.

EC_T_BOOL bExtended

[out] Slave generated by API ConfigExtend

EC_T_BOOL bDcReferenceClock

[out] Slave is reference clock (from ENI file)

EC_T_BOOL bDcPotentialRefClock

[out] Slave can be used as a reference clock (from ENI file)

EC_T_DWORD dwDcCycleTime0

[out] Cycle time of Sync0 event [ns] (from ENI file)

EC_T_DWORD dwDcCycleTime1

[out] Calculated value dwDcCycleTime1 [ns] = Cycle time of Sync1 event - Cycle time of Sync1 event + Shift time of Sync0 event (from ENI file)

EC_T_INT nDcShiftTime

[out] Shift time of Sync0 event [ns] (from ENI file)

Supported mailbox protocols flags

EC_MBX_PROTOCOL_AOE

EC_MBX_PROTOCOL_EOE

EC_MBX_PROTOCOL_COE

EC_MBX_PROTOCOL_FOE

EC_MBX_PROTOCOL_SOE

EC_MBX_PROTOCOL_VOE

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 EtherCAT stack isn't initialized

- *EC_E_INVALIDPARAM* if dwInstanceID is out of range
- *EC_E_NOTFOUND* if no slave matching bFixedAddressing / wSlaveAddress can be found
- *EC_E_NO_MBX_SUPPORT* if the slave does not support mailbox communication
- *EC_E_NO_EOE_SUPPORT* if the slave supports mailbox communication, but not EoE

struct **EC_T_CFG_SLAVE_EOE_INFO**

Public Members

EC_T_DWORD dwSlaveId
[out] Slave ID

EC_T_BOOL bMacAddr
[out] Indicates whether the MAC address could be read and is valid

EC_T_BYTE abyMacAddr[6]
[out] MAC address

EC_T_BOOL bIpAddr
[out] Indicates whether the IP address could be read and is valid

EC_T_IPADDR oIpAddr
[out] IP address

EC_T_BOOL bSubnetMask
[out] Indicates whether the subnet mask could be read and is valid

EC_T_IPADDR oSubnetMask
[out] Subnet mask

EC_T_BOOL bDefaultGateway
[out] Indicates whether the default gateway could be read and is valid

EC_T_IPADDR oDefaultGateway
[out] Default gateway

EC_T_BOOL bDnsServer
[out] Indicates whether the DNS server could be read and is valid

EC_T_IPADDR oDnsServer
[out] DNS server

EC_T_BOOL bDnsName
[out] Indicates whether the DNS name could be read and is valid

EC_T_CHAR szDnsName[32]
[out] DNS name

EC_T_BOOL bDisableEoe
[out] Indicates whether the EoE is Disabled or not

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 SyncManager information of a configured slave from the ENI file.

Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **pSlaveSmInfo** – [out] Information about the slave.

Returns

EC_E_NOERROR or error code

```
struct EC_T_CFG_SLAVE_SM_ENTRY
```

Public Members

EC_T_WORD **wPhysAddr**
[out] ESC (0x800 + y * 8)

EC_T_WORD **wLength**
[out] ESC (0x802 + y * 8)

EC_T_BYTE **byOpMode**
[out] Bits 0..1 ESC (0x804 + y * 8)

EC_T_BYTE **byDirection**
[out] Bits 2..3 ESC (0x804 + y * 8)

EC_T_DWORD **dwPdBitOffs**
[out] Process input data bit offset (from ENI file)

EC_T_DWORD **dwPdBitSize**
[out] Process input data bit size (from ENI file)

EC_T_WORD **wWkcStateDiagBitOffs**
[out] Offset of WkcState bit in diagnosis image

EC_T_WORD **wMasterSyncUnit**
[out] Sync Unit (ENI: ProcessData/TxPdo[1..4]@Su)

struct **EC_T_CFG_SLAVE_SM_INFO**

Public Members

EC_T_DWORD **dwSlaveId**
[out] Slave ID

EC_T_DWORD **dwSmInfoNumOf**
[out] Number of available sync managers

EC_T_CFG_SLAVE_SM_ENTRY **aoSmInfos**[ECREG_SYNCMANAGER_MAX_NUMOF]
[out] Sync managers info

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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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: *www xxxx yyyy zzzz* (each nibble : port 3210)

www : Signal detected 1=yes, 0=no

xxxx : Loop closed 1=yes, 0=no

yyyy : Link established 1=yes, 0=no

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

EC_T_WORD **wAutoIncAddress**

[out] The slave's auto increment address

EC_T_BOOL **bDcSupport**

[out] Slave supports DC (Bus Topology Scan)

EC_T_BOOL **bDc64Support**

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

EC_T_DWORD **dwVendorId**

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

EC_T_DWORD **dwProductCode**

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

EC_T_DWORD **dwRevisionNumber**

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

EC_T_DWORD **dwSerialNumber**

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

EC_T_BYTE byESCType

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

EC_T_BYTE byESCRevision

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

EC_T_WORD wESCBuild

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

EC_T_BYTE byPortDescriptor

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

EC_T_WORD wFeaturesSupported

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

EC_T_WORD wStationAddress

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

EC_T_WORD wAliasAddress

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

EC_T_WORD wALStatus

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

EC_T_WORD wALStatusCode

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

EC_T_DWORD dwSystemTimeDifference

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

EC_T_WORD wMbxSupportedProtocols

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

EC_T_WORD wDLStatus

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

EC_T_WORD wPrevPort

[out] Connected port of the previous slave

EC_T_WORD wIdentifyData

[out] Last read identification value see [EC_T_CFG_SLAVE_INFO.wIdentifyAdo](#)

EC_T_BOOL bLineCrossed

[out] Line crossed was detected at this slave

EC_T_DWORD dwSlaveDelay

[out] Delay behind slave [ns]. This value is only valid if a DC configuration is used.

EC_T_DWORD dwPropagDelay

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

EC_T_BOOL bIsRefClock

[out] Slave is reference clock

EC_T_BOOL bIsDeviceEmulation

[out] Slave without Firmware. ESC register 0x0141, enabled by EEPROM offset 0x0000.8.

EC_T_WORD wLineCrossedFlags

[out] Combination of *Line crossed flags*

EC_T_DWORD dwCyclicWkcErrorCnt

[out] Counter for Cyclic WC Error

EC_T_DWORD dwSlaveAbsentCnt

[out] Counter for Absent/Not Present Slaves

EC_T_DWORD dwUnexpectedStateCnt

[out] Counter for Abnormal State Change

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 a slave.

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

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **bFixedAddressing** – [in] EC_TRUE: use station address, EC_FALSE: use AutoInc address
- **wSlaveAddress** – [in] Slave address according bFixedAddressing
- **wAdo** – [in] ADO used for identification command
- **pwValue** – [out] Pointer to Word value containing the Identification value
- **dwTimeout** – [in] Timeout [ms]

Returns

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

- *EC_E_BUSY* if another transfer request is already pending or the master or the corresponding slave is currently changing its operational state
- *EC_E_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC_E_ADO_NOT_SUPPORTED* if the slave does not support requesting ID mechanism
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

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 EtherCAT stack 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_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 SubDevice identification request is completed.

emNotify - EC_NOTIFY_SLAVE_IDENTIFICATION

Parameter

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

struct **EC_T_SLAVE_IDENTIFICATION_NOTIFY_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 emIoctl - EC_IOCTL_SET_AUTO_ACK_AL_STATUS_ERROR_ENABLED

EC_IOCTL_SET_AUTO_ACK_AL_STATUS_ERROR_ENABLED

Specifies if slave errors must be automatically acknowledged.

Parameters

- **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

Returns

EC_E_NOERROR or error code

The pending SubDevice error will be acknowledged during the next *emSetSlaveState()* call.

6.6.41 emIoctl - EC_IOCTL_SET_AUTO_ADJUST_CYCCMD_WKC_ENABLED

EC_IOCTL_SET_AUTO_ADJUST_CYCCMD_WKC_ENABLED

Specifies if the cyclic commands expected WKC must be automatically adjusted according to the state and the presence of the slaves.

Parameters

- **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 stays 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

Returns

EC_E_NOERROR or error code

If TRUE, the notification *emNotify - EC_NOTIFY_CYCCMD_WKC_ERROR* is only generated if a SubDevice doesn't increment the WKC although it should. *AUTO_ADJUST_CYCCMD_WKC* is disabled by default.

See also:

- *Cyclic cmd WKC validation and Frame Loss*
- *Working Counter (WKC) State in Diagnosis Image*

6.6.42 emSetSlaveDisabled

Before using this function, please check if the following patents have 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 cannot be set higher than PREOP. If the state is higher than PREOP at the time this function is called the state will be automatically changed 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 emIoctl - EC_IOCTL_SET_SLAVE_MAX_STATE

Specifies maximum state for specific slave.

EC_IOCTL_SET_SLAVE_MAX_STATE

This call specifies maximum state for the slave.

Parameters

- **pbyInBuf** – [in] Pointer to struct *EC_T_SLAVE_MAX_STATE_DESC*
- **dwInBufSize** – [in] Size of the input buffer in bytes, e.g. `sizeof(EC_T_SLAVE_MAX_STATE_DESC)`
- **pbyOutBuf** – [out] Should be set to `EC_NULL`
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to `EC_NULL`

Returns

EC_E_NOERROR or error code

struct **EC_T_SLAVE_MAX_STATE_DESC**

Public Members

EC_T_DWORD dwSlaveId
[in] Slave ID

EC_T_STATE **eState**
[in] Requested Slave State

6.6.44 emSetSlaveDisconnected

Before using this function, please check if the following patents have 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 PROCESSING 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 cannot be set higher than INIT. If the state is higher than INIT at the time this function is called, the state will be automatically changed 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 have 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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARAM* 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: wwwwww xxxx yyyy zzzz (each nibble : port 3210)

wwwwww : Signal detected 1=yes, 0=no (ESC Register 0x110 Bit 9, 11, 13, 15)

xxxx : Loop closed 1=yes, 0=no (ESC Register 0x110 Bit 8, 10, 12, 14)

yyyy : Link established 1=yes, 0=no (ESC Register 0x110 Bit 4, 5, 6, 7)

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

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* if dwInstanceID is out of range or the output pointer is EC_NULL
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found

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 can also 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 EtherCAT stack 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
- *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.

An *EC_T_PORT_OPERATION_NOTIFY_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_NOTIFY_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 EtherCAT stack 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

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

```
struct EC_T_PORT_OPERATION_NOTIFY_DESC
```

Public Members

EC_T_DWORD **dwTransferId**

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 emIoctl - EC_IOCTL_SET_NEW_BUSSLAVES_TO_INIT

EC_IOCTL_SET_NEW_BUSSLAVES_TO_INIT

Force state change to INIT for all new slaves in the network after detection.

Parameters

- **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

Returns

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 SubDevices the EtherCAT® MainDevice stack will notify the application about such an event. The MainDevice automatically detects unexpected SubDevice states by evaluating the AL Status event interrupt. If the interrupt is set, the MainDevice reads the state of each SubDevice and compares it to the expected (required) state. In case of a state mismatch the MainDevice 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. SubDevice unrelated errors
2. SubDevice related errors

A SubDevice related error notification will also contain the information about which SubDevice has generated an error. If for example a SubDevice could not be set into the requested state the application will get the *emNotify - EC_NOTIFY_SLAVE_INITCMD_RESPONSE_ERROR* error notification including SubDevice related information. A SubDevice unrelated error does not contain this information even if one specific SubDevice caused the error. For example if one or multiple SubDevices 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

SubDevice is powered off or disconnected while bus is operational

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

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

MainDevice 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. SubDevice is disconnected or powered off:

- MainDevice detects a DL status event interrupt and performs a bus scan.
- MainDevice calls *emNotify - EC_NOTIFY_SLAVE_PRESENCE*.
- Application gets informed and could set the MainDevice including all SubDevices into a lower state, e.g. *eEcatState_INIT*.

2. SubDevice state is not OPERATIONAL anymore

- MainDevice calls *emNotify - EC_NOTIFY_SLAVE_UNEXPECTED_STATE*.
- Application gets informed and could either set the MainDevice including all SubDevices into a lower state (e.g. *eEcatState_PREOP*) or call *emSetSlaveState ()* to repair the failed SubDevice.

3. SubDevice is re-connected or powered on:

- MainDevice detects a DL status event interrupt and performs a bus scan.
- MainDevice calls *emNotify - EC_NOTIFY_SLAVE_PRESENCE*.
- Application could wait until all SubDevices are re-connected by calling the functions *emGetNumConnectedSlaves ()* and *emGetNumConfiguredSlaves ()*.
- After all SubDevices are re-connected the application could either set the MainDevice including all SubDevices to *eEcatState_INIT* and afterwards to *eEcatState_OP*, or the application uses *emSetSlaveState ()* to repair only the failed SubDevices.

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

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pLogParms** – [in] New Log parameters

6.7.2 emEthDbgMsg

```
static EC_T_DWORD ecatEthDbgMsg (  
    EC_T_BYTE byEthTypeByte0,  
    EC_T_BYTE byEthTypeByte1,  
    EC_T_CHAR *szMsg  
)
```

```
EC_T_DWORD emEthDbgMsg (  
    EC_T_DWORD dwInstanceID,  
    EC_T_BYTE byEthTypeByte0,  
    EC_T_BYTE 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_BYTE byEthTypeByte0 = 6;  
EC_T_BYTE 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 emloCtl - EC_IOCTL_GET_SLVSTATISTICS

EC_IOCTL_GET_SLVSTATISTICS

Get Slave's statistics counter. Counters are collected on a regular basis (default: off) and show errors on Ethernet layer.

Parameters

- **pbyInBuf** – [in] Pointer to an 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.

Returns

EC_E_NOERROR or error code

struct **EC_T_SLVSTATISTICS_DESC**

Public Members

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

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

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

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

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

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

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

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

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

See also:

- *emloCtl - EC_IOCTL_SET_SLVSTAT_PERIOD*

- *emIoCtl - 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:

- *emIoCtl - EC_IOCTL_GET_SLVSTATISTICS*
- *emGetSlaveId()*

6.7.5 emIoCtl - EC_IOCTL_CLR_SLVSTATISTICS

EC_IOCTL_CLR_SLVSTATISTICS

Clear all error registers in all slaves.

Parameters

- **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

Returns*EC_E_NOERROR* or error code**6.7.6 emClearSlaveStatistics**

```
static EC_T_DWORD emClearSlaveStatistics (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 emIoctl - EC_IOCTL_GET_SLVSTAT_PERIOD****EC_IOCTL_GET_SLVSTAT_PERIOD**

Get Slave Statistics collection period. Period of 0: automatic slave statistics collection disabled.

Parameters

- **pbyInBuf** – [in] Should be set to EC_NULL
- **dwInBufSize** – [in] Should be set to 0
- **pbyOutBuf** – [out] Pointer to an 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.

Returns*EC_E_NOERROR* or error code

6.7.8 emloCtl - EC_IOCTL_SET_SLVSTAT_PERIOD

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 sets the time between the read sequences.

Parameters

- **pbyInBuf** – [in] Pointer to an 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

Returns

EC_E_NOERROR or error code

6.7.9 emloCtl - EC_IOCTL_FORCE_SLVSTAT_COLLECTION

EC_IOCTL_FORCE_SLVSTAT_COLLECTION

Sends datagrams to collect slave statistics counters.

Parameters

- **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

Returns

EC_E_NOERROR or error code

6.7.10 emloCtl - EC_IOCTL_CLEAR_MASTER_INFO_COUNTERS

EC_IOCTL_CLEAR_MASTER_INFO_COUNTERS

Reset Master Info Counters according to given bit masks.

Parameters

- **pbyInBuf** – [in] Pointer to a 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

Returns

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 emIoctl - EC_IOCTL_SET_FRAME_RESPONSE_ERROR_NOTIFY_MASK

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 can then decide to suppress those error messages. By default all errors except *EC_FRAME_RESPONSE_ERROR_NOTIFY_MASK_NON_ECAT_FRAME* are enabled (the notification mask is set to *EC_FRAME_RESPONSE_ERROR_NOTIFY_MASK_DEFAULT*).

Parameters

- **pbyInBuf** – [in] Pointer to an 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

Returns

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 emIoctl - EC_IOCTL_SET_FRAME_LOSS_SIMULATION

Important: Do not activate this on shipped releases. Frameloss has significant influence on performance and reliability of the application!

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.

Parameters

- **pbyInBuf** – [in] Array of four `EC_T_DWORDS`s (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`

Returns

`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 emIoctl - EC_IOCTL_SET_RXFRAME_LOSS_SIMULATION

EC_IOCTL_SET_RXFRAME_LOSS_SIMULATION

Same as `EC_IOCTL_SET_FRAME_LOSS_SIMULATION` but only enables receive direction frame losses.

Parameters

- **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

Returns

EC_E_NOERROR or error code

6.7.14 emIoctl - EC_IOCTL_SET_TXFRAME_LOSS_SIMULATION

EC_IOCTL_SET_TXFRAME_LOSS_SIMULATION

Same as *EC_IOCTL_SET_FRAME_LOSS_SIMULATION* but only enables transmit direction frame losses.

Parameters

- **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

Returns

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_NOTIFY_DESC **FrameLossAfterSlaveDesc**

Frame loss after Slave descriptor

EC_T_S2SMBX_ERROR_DESC **S2SMBxErrorDesc**

S2S Mailbox Error descriptor

EC_T_BAD_CONNECTION_NOTIFY_DESC **BadConnectionDesc**

Bad connection descriptor

EC_T_COMMUNICATION_TIMEOUT_NOTIFY_DESC **CommunicationTimeoutDesc**

Communication timeout descriptor

EC_T_TAP_LINK_STATUS_NOTIFY_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 SubDevice) is stored in the appropriate structure of a union. These error information structures are described in the following sections.

The EtherCAT® MainDevice 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 a SubDevice). Using the control interface *emIoCtl - 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 MainDevice. These commands will address one or multiple SubDevices. These EtherCAT® commands contain a working counter which has to be incremented by each SubDevice that is addressed. The working counter will be checked after the EtherCAT® command is received by the MainDevice. If the expected working counter does not match 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 MainDevice is determined by the EtherCAT® configuration (XML) file for each cyclic EtherCAT® command (section Config/Cyclic/Frame/Command/Cnt). Detailed error information is stored in the 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:

- *emIoCtl - EC_IOCTL_SET_NOTIFICATION_ENABLED* for how to control the deactivation.
- *Cyclic cmd WKC validation and Frame Loss*
- *Working Counter (WKC) State in Diagnosis Image*
- *emIoCtl - 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 MainDevice init commands. The working counter value expected by the MainDevice is determined by the EtherCAT® configuration (XML) file for each MainDevice 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 SubDevices which have to process this init command.

Detailed error information is stored in the 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 SubDevice init commands. The working counter value expected by the MainDevice is determined by the EtherCAT® configuration (XML) file for each SubDevice 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 the structure `EC_T_WKCERR_DESC` of `EC_T_ERROR_NOTIFICATION_DESC`. The structure member SlaveProp contains information about the corresponding SubDevice.

6.7.19 emNotify - EC_NOTIFY_FOE_MBSLAVE_ERROR

This error will be indicated in case a SubDevice notifies an error over FoE.

6.7.20 emNotify - EC_NOTIFY_EOE_MBXSNB_WKC_ERROR

This error will be indicated in case the working counter of an EoE mailbox write command was not set to the expected value of 1.

Detailed error information is stored in the structure `EC_T_WKCERR_DESC` of `EC_T_ERROR_NOTIFICATION_DESC`. The structure member `SlaveProp` contains information about the corresponding SubDevice.

6.7.21 emNotify - EC_NOTIFY_COE_MBXSNB_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 the structure `EC_T_WKCERR_DESC` of `EC_T_ERROR_NOTIFICATION_DESC`. The structure member `SlaveProp` contains information about the corresponding SubDevice.

6.7.22 emNotify - EC_NOTIFY_FOE_MBXSNB_WKC_ERROR

This error will be indicated in case the working counter of an FoE mailbox write command was not set to the expected value of 1.

6.7.23 emNotify - EC_NOTIFY_VOE_MBXSNB_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 the structure `EC_T_WKCERR_DESC` of `EC_T_ERROR_NOTIFICATION_DESC`. The structure member `SlaveProp` contains information about the corresponding SubDevice.

6.7.24 emNotify - EC_NOTIFY_S2SMBX_ERROR

This error will be indicated in case a SubDevice-To-SubDevice 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 received Ethernet frame does not match the frame expected or if an expected frame was not received.

This notification is enabled by default.

See also:

`emIoCtl - 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 MainDevice and sent to the SubDevices at a later time (usually after sending cyclic frames). The MainDevice 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 MainDevice is initialized.

See also:

emInitMaster()

The MainDevice will retry sending the frame if the MainDevice 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 MainDevice due to an 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 MainDevice configuration parameter *dwMaxAcycCmdsPerCycle* -> shorter MainDevice timer cycle, i.e. shorter period between two calls to

```
emExecJob (eUsrJob_MasterTimer)
```

-> higher timeout value (MainDevice configuration parameter *dwEcatCmdTimeout*)

If the frame was sent too late by the MainDevice (due to improper configuration values) it will also be received too late and the MainDevice then signals an *eRspErr_WRONG_IDX* or *eRspErr_UNEXPECTED* error (as the MainDevice 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 MainDevice 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 MainDevice due to an improper configuration, to avoid these error notifications the configuration may be changed as follows:**

- lower value for MainDevice 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*) has to be properly scheduled with sending cyclic frames by calling *emExecJob()* (*eUsrJob_SendAllCycFrames*).

Using the control interface *emIoCtl - 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 the 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_BYTE byEcCmdHeaderIdxSet

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

EC_T_BYTE byEcCmdHeaderIdxAct

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

EC_T_WORD wCycFrameNum

Number of the lost cyclic frame from the ENI

EC_T_DWORD dwTaskId

Cyclic Task ID (ENI: Cyclic/TaskId). Only valid if bIsCyclicFrame is set

enum **EC_T_FRAME_RSPERR_TYPE**

Values:

enumerator **eRspErr_UNDEFINED**

Undefined

enumerator **eRspErr_NO_RESPONSE**

No Ethernet frame received (timeout, frame loss)

enumerator **eRspErr_WRONG_IDX**

Wrong IDX value in acyclic frame

enumerator **eRspErr_UNEXPECTED**

Unexpected frame was received

enumerator **eRspErr_FRAME_RETRY**

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

enumerator **eRspErr_RETRY_FAIL**

All retry mechanism fails to re-sent acyclic frames

enumerator **eRspErr_FOREIGN_SRC_MAC**

Frame with MAC from other Master received

enumerator **eRspErr_NON_ECAT_FRAME**

Non EtherCAT frame received

enumerator **eRspErr_CRC**

Ethernet frame with CRC error received

6.7.26 emNotify - EC_NOTIFY_SLAVE_INITCMD_RESPONSE_ERROR

This error code will be indicated if a SubDevice does not respond appropriately while sending SubDevice init commands. The SubDevice 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 the 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:

emIoCtl - 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 SubDevice 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 ms will be used by the EtherCAT® MainDevice. The timeout value used for EoE mailbox SubDevices will be set to a fixed value of 5000 ms.

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 MainDevice init commands. The MainDevice 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 the 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® MainDevice checks whether all SubDevices are still in OPERATIONAL state. If at least one SubDevice is not OPERATIONAL this error will be indicated.

6.7.30 emNotify - EC_NOTIFY_ALL_DEVICES_OPERATIONAL

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

6.7.31 emNotify - EC_NOTIFY_STATUS_SLAVE_ERROR

When processing cyclic frames the EtherCAT® MainDevice checks if at least one SubDevice has the ERROR bit in the AL-STATUS register set. In that case this error will be indicated. The MainDevice will then automatically determine detailed error information of the SubDevice(s) indicating an error and acknowledge the error status. The application will get an *emNotify - EC_NOTIFY_SLAVE_ERROR_STATUS_INFO* notification for each such SubDevice. Usually those SubDevices 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:

emIoCtl - EC_IOCTL_SET_NOTIFICATION_ENABLED for how to control the deactivation

6.7.32 emNotify - EC_NOTIFY_SLAVE_ERROR_STATUS_INFO

Every time the MainDevice detects a SubDevice error, the Error bit on the specific SubDevice is cleared and this error code will be signalled to the application. Detailed error information is stored in the 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:

emIoCtl - *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 MainDevice 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_NTIFY_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:

emIoCtl - `EC_IOCTL_SET_NOTIFICATION_ENABLED` for how to control the activation

6.7.34 emNotify - EC_NOTIFY_SLAVE_UNEXPECTED_STATE

This error is signaled every time a SubDevice changes into an unexpected state. Detailed error information is stored in the 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:

emIoCtl - `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 MainDevice 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_NOTIFY_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:

emIoctl - *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 SubDevices can be found although the SubDevices are connected. This does not affect link connection detection, therefore this notification will not be 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 MainDevice, 1 = restarting_
↳MainDevice */
```

6.7.39 emNotify - EC_NOTIFY_EEPROM_CHECKSUM_ERROR

This error is signaled every time an EEPROM checksum error is detected.

Detailed error information is stored in the 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 signaled when invalid mailbox data have been received from SubDevice. 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 signaled every time a PDI watchdog error is detected. Detailed error information is stored in the 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_PFLOGFRAME_CB pvLogFrameCallBack,
    EC_T_VOID *pvContext
)
EC_T_DWORD emLogFrameEnable (
    EC_T_DWORD dwInstanceID,
    EC_T_PFLOGFRAME_CB pvLogFrameCallBack,
    EC_T_VOID *pvContext
)
```

Setup a callback function to log the EtherCAT network traffic.

The callback function is called by the cyclic task. Therefore the code inside the callback has to be fast and non-blocking. The callback parameter `dwLogFlags` can be used as a filter to log just specific frames. The master discards the frame if the callback function modifies the Ethernet frame type at byte offset 12.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pvLogFrameCallBack** – [in] Pointer to frame logging callback function
- **pvContext** – [in] Pointer to function specific context

Returns

`EC_E_NOERROR` or error code

```
typedef EC_T_VOID (*EC_T_PFLOGFRAME_CB)(EC_T_VOID *pvContext, EC_T_DWORD dwLogFlags,
EC_T_DWORD dwFrameSize, EC_T_BYTE *pbyFrame)
```

Note: The master discards the frame if the callback function modifies the Ethernet frame type at byte offset 12.

Param pvContext

[in] Arbitrarily application-defined parameter passed to callback

Param dwLogFlags

[in] Frame logging flags, `EC_LOG_FRAME_FLAG_...`

Param dwFrameSize

[in] Size of frame in bytes

Param 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 to much CPU time!!!
 */
EC_T_VOID LogFrameHandler(EC_T_VOID* pvContext, EC_T_DWORD dwLogFlags, EC_T_DWORD_
↳dwFrameSize, EC_T_BYTE* pbyFrame)
{
    EC_T_STATE          eMasterState;

    /* get MainDevice state */
    eMasterState = (EC_T_STATE)(dwLogFlags & EC_LOG_FRAME_FLAG_MASTERSTATE_MASK);

    /* skip tx frame */
    if ((S_dwLogFrameLevel == 3) && !(dwLogFlags & EC_LOG_FRAME_FLAG_RX_FRAME))
        return;

    /* skip cyclic frame */
    if ((S_dwLogFrameLevel == 2) && !(dwLogFlags & EC_LOG_FRAME_FLAG_ACYC_FRAME))
        return;

    /* skip red frame */
    if (dwLogFlags & EC_LOG_FRAME_FLAG_RED_FRAME)
        return;

    /* do something with pbyFrame ... */
}

```

6.7.44 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 EtherCAT stack 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. See *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 the 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 a 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 MainDevice 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 signaled every time a bad connection is detected within the call of `emBadConnectionsDetect()` or `emSelfTestScan()`. It contains the exact location of the bad connection between two SubDevices. This notification is enabled by default.

```
struct EC_T_BAD_CONNECTION_NTIFY_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:

- *emIoCtl - EC_IOCTL_SET_NOTIFICATION_ENABLED* for how to control the deactivation
- *ETG.1600 Guideline for Planning, Assembling and Commissioning of EtherCAT Networks*

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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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 sets the default value.

EC_T_DWORD **dwFrameSizeMin**

[in] Min frame size [bytes]. Default value is 60. A value of 0 sets the default value.

EC_T_DWORD **dwFrameSizeMax**

[in] Max frame size [bytes]. Default value is 1514. A value of 0 sets the default value.

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 sets the default value.

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 MainDevice init parms.

```
/* enable performance measurements */
oInitParms.PerfMeasInternalParms.bEnabled = EC_TRUE;

/* initialize the MainDevice */
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) [us]: 12.5/ 15.9/ 25.6
PerfMsmt 'JOB_SendAllCycFrames' (min/avg/max) [us]: 3.6/ 5.7/ 14.8
PerfMsmt 'JOB_MasterTimer' (min/avg/max) [us]: 2.1/ 3.7/ 8.2
PerfMsmt 'JOB_SendAcycFrames' (min/avg/max) [us]: 0.3/ 0.6/ 2.6
PerfMsmt 'Cycle Time' (min/avg/max) [us]: 918.4/ 999.6/1067.9
PerfMsmt 'myAppWorkPd' (min/avg/max) [us]: 0.1/ 0.4/ 0.8
PerfMsmt 'JOB_Total' (min/avg/max) [us]: 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)

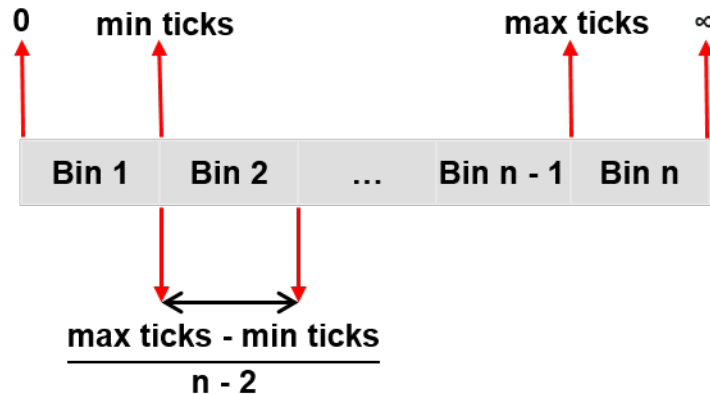
```

/* 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 MainDevice */
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

Distance benchmarks are used to measure the time between the cycle start and the benchmark start. 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=qwMaxTicks=0` from 0 - cycle time to $0.5 * \text{cycle time} - 1.5 * \text{cycle time}$

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;
OsMmemset (&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 a 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 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 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 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 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 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 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 error code

struct **EC_T_PERF_MEAS_INFO**

Public Members

EC_T_CHAR **szName**[MAX_STD_STRLEN]
Name of the benchmark

EC_T_UINT64 **qwFrequency**
Frequency in Hz used by the timer

EC_T_USER_JOB **eUserJob**
UserJob associated with the benchmark

EC_T_DWORD **dwBinCountHistogram**
length of Histogram Bins

EC_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 error code

struct **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 $-\infty < x < 500\text{us}$

aBins[1]: [500us, 505us) corresponds to $500\text{us} \leq x < 505\text{us}$

aBins[2]: [505us, 510us) corresponds to $505\text{us} \leq x < 510\text{us}$

...

aBins[199]: [1490us, 1495us) corresponds to $1490\text{us} \leq x < 1495\text{us}$

aBins[200]: [1495us, 1500us) corresponds to $1495\text{us} \leq x < 1500\text{us}$

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] Instance ID (Multiple EtherCAT Network Support)
- **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 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] Instance ID (Multiple EtherCAT Network Support)
- **dwTaskId** – [in] Task Job ID
- **pdwNumOf** – [out] Number of performance measurements

Returns*EC_E_NOERROR* or 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] Instance ID (Multiple EtherCAT Network Support)
- **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 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] Instance ID (Multiple EtherCAT Network Support)
- **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 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 an 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 SubDevice into the transfer object, determine the transfer ID, store the client ID in the object and initiate the transfer (e.g. an 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 EC-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. The 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 EC-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) 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_BYTE ***pbyMbxTferDescData**

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

struct **EC_T_MBXTFER**

Public Members

EC_T_DWORD **dwClntId**

[] Client ID

EC_T_MBXTFER_DESC **MbxTferDesc**

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

EC_T_MBXTFER_TYPE **eMbxTferType**

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

EC_T_DWORD dwDataLen

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

EC_T_BYTE *pbyMbxTferData

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

EC_T_MBXTFER_STATUS eTferStatus

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

EC_T_DWORD dwErrorCode

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

EC_T_DWORD dwTferId

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

EC_T_MBX_DATA MbxData

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

enum **EC_T_MBXTFER_TYPE**

Values:

enumerator **eMbxTferType_COE_SDO_DOWNLOAD**
CoE SDO download

enumerator **eMbxTferType_COE_SDO_UPLOAD**
CoE SDO upload

enumerator **eMbxTferType_COE_GETODLIST**
CoE Get object dictionary list

enumerator **eMbxTferType_COE_GETOBDESC**
CoE Get object description

enumerator **eMbxTferType_COE_GETENTRYDESC**
CoE Get object entry description

enumerator **eMbxTferType_COE_EMERGENCY**
CoE emergency request

enumerator **eMbxTferType_COE_RX_PDO**
CoE RxPDO

enumerator **eMbxTferType_FOE_FILE_UPLOAD**
FoE upload

enumerator **eMbxTferType_FOE_FILE_DOWNLOAD**
FoE download

enumerator **eMbxTferType_SOE_READREQUEST**
SoE read request

enumerator **eMbxTferType_SOE_READRESPONSE**
SoE read response

enumerator **eMbxTferType_SOE_WRITEREQUEST**
SoE write request

enumerator **eMbxTferType_SOE_WRITERESPONSE**
SoE write response

enumerator **eMbxTferType_SOE_NOTIFICATION**
SoE notification

enumerator **eMbxTferType_SOE_EMERGENCY**
SoE emergency

enumerator **eMbxTferType_VOE_MBX_READ**
VoE read

enumerator **eMbxTferType_VOE_MBX_WRITE**
VoE write

enumerator **eMbxTferType_AOE_READ**
AoE read

enumerator **eMbxTferType_AOE_WRITE**
AoE write

enumerator **eMbxTferType_AOE_READWRITE**
AoE read/write

enumerator **eMbxTferType_AOE_WRITECONTROL**
AoE write control

enumerator **eMbxTferType_RAWMBX**
Raw mbx

enumerator **eMbxTferType_FOE_SEG_DOWNLOAD**
FoE segmented download

enumerator **eMbxTferType_FOE_SEG_UPLOAD**
FoE segmented upload

enumerator **eMbxTferType_S2SMBX**
S2S mbx

enumerator **eMbxTferType_FOE_UPLOAD_REQ**
FoE upload request

enumerator **eMbxTferType_FOE_DOWNLOAD_REQ**
FoE download request

enumerator **eMbxTferType_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*, containing 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 SubDevices of underlying field-buses, e.g. for a CAN application protocol SubDevice connected to an 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 EtherCAT stack 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 an AoE mailbox read request to an EtherCAT slave device.

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

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID
- **wTargetPort** – [in] Target port
- **dwIndexGroup** – [in] AoE read command index group
- **dwIndexOffset** – [in] AoE read command index offset
- **dwDataLen** – [in] Buffer length [bytes]
- **pbyData** – [out] Buffer receiving transferred data
- **pdwDataOutLen** – [out] Number of bytes read from the target device

- **pdwErrorCode** – [out] AoE response error code
- **pdwCmdResult** – [out] AoE read command result code
- **dwTimeout** – [in] Timeout [ms]. The function will block at most for this time.

Returns

- *EC_E_NOERROR* on success
- *EC_E_AOE_VENDOR_SPECIFIC* if the AoE device has responded with a user defined error code
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

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 function 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* on success
- *EC_E_AOE_VENDOR_SPECIFIC* if the AoE device has responded with a user defined error code
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

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 an AoE mailbox write request to an EtherCAT slave device.

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

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID
- **wTargetPort** – [in] Target port
- **dwIndexGroup** – [in] AoE write command index group
- **dwIndexOffset** – [in] AoE write command index offset
- **dwDataLen** – [in] Buffer length [bytes]
- **pbyData** – [in] Buffer containing 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* on success
- *EC_E_AOE_VENDOR_SPECIFIC* if the AoE device has responded with a user defined error code
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

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 function 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` on success
- `EC_E_AOE_VENDOR_SPECIFIC` if the AoE device has responded with a user defined error code
- `EC_E_MASTER_RED_STATE_INACTIVE` if Master Redundancy is configured and master is inactive

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 an AoE mailbox read/write request to an EtherCAT slave device.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **poTargetNetId** – [in] Target NetID
- **wTargetPort** – [in] Target port
- **dwIndexGroup** – [in] AoE read/write command index group
- **dwIndexOffset** – [in] AoE read/write command index offset
- **dwReadDataLen** – [in] Number of bytes to read from the target device
- **dwWriteDataLen** – [in] Number of bytes to write to the target device
- **pbyData** – [in, out] Buffer containing and receiving 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* on success
- *EC_E_AOE_VENDOR_SPECIFIC* if the AoE device has responded with a user defined error code
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

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 an 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 [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* on success
- *EC_E_AOE_VENDOR_SPECIFIC* if the AoE device has responded with a user defined error code
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

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_BYTE byObSubIndex,
    EC_T_BYTE *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_BYTE byObSubIndex,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout,
    EC_T_DWORD dwFlags
)
```

Execute a CoE SDO download to an EtherCAT slave device.

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

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **byObSubIndex** – [in] Object sub index. If Complete Access only 0 or 1 allowed.
- **pbyData** – [in] Buffer containing transferred data
- **dwDataLen** – [in] Buffer length [bytes]
- **dwTimeout** – [in] Timeout [ms]
- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceId is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *CoE SDO error code*

emCoeSdoDownload Example

The following code demonstrates how to download a CoE object to a SubDevice 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_BYTE     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 sub index. 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 EtherCAT stack 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 the slave is not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if the slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if the 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 the ADS server is running
- *CoE SDO error code*

emCoeSdoDownloadReq Example

The following code demonstrates how to download a CoE object to a SubDevice 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_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 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*, containing 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 requested 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_BYTE **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 sub index. If Complete Access only 0 or 1 allowed.
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length [bytes]
- **pdwOutDataLen** – [out] Length of received data [byte]
- **dwTimeout** – [in] Timeout [ms]

- **dwFlags** – [in] Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceId is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *CoE SDO error code*

emCoeSdoUpload Example

The following code demonstrates how to upload a CoE object from a SubDevice 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_BYTE 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.

A unique transfer ID must be written into *EC_T_MBXTFER.dwTferId*. EC_NOTIFY_MBOXRCV is given on completion.

Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object created with emMbxTferCreate
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **byObSubIndex** – [in] Object sub index. 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 EtherCAT stack 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 the ADS server is running
- *CoE SDO error code*

emCoeSdoUploadReq Example

The following code demonstrates how to upload a CoE object from a SubDevice 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(...)
     - pAppContext->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*, containing 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 requested 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 emCoeGetODList

```
static EC_T_DWORD ecatCoeGetODList (
    EC_T_DWORD dwSlaveId,
    EC_T_COE_ODLIST_TYPE eListType,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_COE_ODLIST *poOdList,
    EC_T_DWORD dwTimeout
)
```

```
EC_T_DWORD emCoeGetODList (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_COE_ODLIST_TYPE eListType,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_COE_ODLIST *poOdList,
    EC_T_DWORD dwTimeout
)
```

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

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

Note: The data buffer will receive the slave response containing the list type followed by the list itself. Therefore the buffer must be 2 bytes bigger than the expected list size.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **eListType** – [in] Which object types shall be transferred
- **pbyData** – [out] Buffer receiving transferred data
- **dwDataLen** – [in] Buffer length in bytes
- **poOdList** – [out] Received OD list object
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if master isn't initialized
- *EC_E_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_StoredFRepl**
Only stored for a device replacement objects

enumerator **eODListType_StartupParm**
Only startup parameter objects

emCoeGetODList Example

The following code demonstrates how to get the list of objects that are available in a SubDevice using the API `emCoeGetODList()`.

```

EC_T_BYTE*      pbyOdList = EC_NULL;
EC_T_COE_ODLIST oOdList;
OsMemset(&oOdList, 0, sizeof(EC_T_COE_ODLIST));

/* allocate payload memory */
pbyOdList = (EC_T_BYTE*)OsMalloc(CROD_ODLTFER_SIZE);
if (EC_NULL == pbyOdList)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}

dwRes = emCoeGetODList(dwInstanceId, dwSlaveId, eODListType_ALL, pbyOdList,
↪CROD_ODLTFER_SIZE, &oOdList, 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));
    goto Exit;
}

dwRetVal = dwRes;
Exit:
OsSafeFree(pbyOdList);

```

See also:

- `emGetSlaveId()`

- See `CoeReadObjectDictionary()` in `EcSdoServices.cpp` as an example for `emCoeGetODList()`.

6.11.8 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
)
```

Initiates retrieval of a list of object IDs that are available in a slave and returns immediately.

A unique transfer ID must be written into `EC_T_MBXTFER.dwTferId`. `EC_NOTIFY_MBOXRCV` is given on completion.

Note: The mailbox transfer object will receive the slave response containing the list type followed by the list itself. Therefore the buffer must be 2 bytes bigger than the expected list size.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer
- **dwSlaveId** – [in] Slave ID
- **eListType** – [in] Which object types shall be transferred
- **dwTimeout** – [in] Timeout [ms]

Returns

- `EC_E_NOERROR` if successful
- `EC_E_INVALIDSTATE` if EtherCAT stack isn't initialized
- `EC_E_INVALIDPARG` 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*

emCoeGetODListReq Example

The following code demonstrates how to get the list of objects that are available in a SubDevice 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,
↪"emCoeGetODListReq: %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)
    {
        emMbxTferAbort(dwInstanceId, pMbxTfer);
    }
    emMbxTferDelete(dwInstanceId, pMbxTfer);
}

OsSafeFree(oMbxTferDesc.pbyMbxTferDescData);

```

See also:

- `emNotify - eMbxTferType_COE_GETODLIST`

- `emMbxTferCreate()`
- `emGetSlaveId()`

6.11.9 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`, containing 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

6.11.10 emCoeGetObjectDesc

```
static EC_T_DWORD ecatCoeGetObjectDesc (
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_CHAR *pchObName,
    EC_T_WORD wObNameLen,
    EC_T_COE_OBDESC *poObDesc,
    EC_T_DWORD dwTimeout
)
```

```
EC_T_DWORD emCoeGetObjectDesc (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_CHAR *pchObName,
    EC_T_WORD wObNameLen,
    EC_T_COE_OBDESC *poObDesc,
    EC_T_DWORD dwTimeout
)
```

Determines the description of a specific object.

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

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **pchObName** – [out] Buffer receiving object name
- **wObNameLen** – [in] Buffer length
- **poObDesc** – [out] Received object description object
- **dwTimeout** – [in] Timeout [ms]

Returns

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

emCoeGetObjectDesc Example

The following code demonstrates how to determine the description of a specific object using the API `emCoeGetObjectDesc()`.

```

EC_T_CHAR*      pchObName = EC_NULL;
EC_T_COE_OBDESC oObDesc;
OsMemset(&oObDesc, 0, sizeof(EC_T_COE_OBDESC));

/* allocate payload memory */
pchObName = (EC_T_CHAR*)OsMalloc(CROD_OBDESC_SIZE);
if (EC_NULL == pchObName)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}

dwRes = emCoeGetObjectDesc(dwInstanceId, dwSlaveId, 0x6411, pchObName, CROD_
↪OBDESC_SIZE, &oObDesc, 5000);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪"emCoeGetObjectDesc: %s (0x%x)\n", ecatGetText(dwRes), dwRes));
    goto Exit;
}

dwRetVal = dwRes;
Exit:
OsSafeFree(pchObName);

```

See also:

- `emGetSlaveId()`
- See `CoeReadObjectDictionary()` in `EcSdoServices.cpp` as an example for `emCoeGetObjectDesc()`.

6.11.11 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
)

```

Initiates retrieval of the description of a specific object and returns immediately.

A unique transfer ID must be written into `EC_T_MBXTFER.dwTferId`. `EC_NOTIFY_MBOXRCV` is given on completion.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **dwTimeout** – [in] Timeout [ms]

Returns

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

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;
}

```

(continues on next page)

(continued from previous page)

```

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)
    {
        emMbxTferAbort(dwInstanceId, pMbxTfer);
    }
    emMbxTferDelete(dwInstanceId, pMbxTfer);
}

OsSafeFree(oMbxTferDesc.pbyMbxTferDescData);

```

See also:

- *emNotify - eMbxTferType_COE_GETOBDESC*
- *emMbxTferCreate()*
- *emGetSlaveId()*

6.11.12 emNotify - eMbxTferType_COE_GETOBDESC

Completion of an SDO information service transfer to get an object description.

emNotify - eMbxTferType_COE_GETOBDESC**Parameter**

- *pbyInBuf*: [in] Pointer to a structure of type *EC_T_MBXTFER*, containing 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_ObjDesc* 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_BYTE **byObjCode**

Object code, see Table 62, ETG.1000 section 6

EC_T_BYTE **byObjCategory**

Object category

EC_T_BYTE **byMaxNumSubIndex**

Maximum sub index number

EC_T_WORD **wObjNameLen**

Length of the object name

EC_T_WORD **wStationAddress**

Station address of the slave

EC_T_CHAR ***pchObjName**

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.

6.11.13 emCoeGetEntryDesc

```
static EC_T_DWORD ecatCoeGetEntryDesc (
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTE byObjSubIndex,
    EC_T_BYTE byValueInfo,
    EC_T_BYTE *pbyData,
    EC_T_WORD wDataLen,
    EC_T_COE_ENTRYDESC *poEntryDesc,
    EC_T_DWORD dwTimeout
)
```

```

EC_T_DWORD emCoeGetEntryDesc (
    EC_T_DWORD dwInstanceId,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTE byObSubIndex,
    EC_T_BYTE byValueInfo,
    EC_T_BYTE *pbyData,
    EC_T_WORD wDataLen,
    EC_T_COE_ENTRYDESC *poEntryDesc,
    EC_T_DWORD dwTimeout
)

```

Determines the description of a specific object entry.

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

Parameters

- **dwInstanceId** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **byObSubIndex** – [in] Object sub-index
- **byValueInfo** – [in] The value info bit mask includes which elements shall be in the response. See *Value info flags* for available values.
- **pbyData** – [out] Buffer receiving additional entry desc data
- **wDataLen** – [in] Buffer length in bytes
- **poEntryDesc** – [out] Received entry desc object
- **dwTimeout** – [in] Timeout [ms]

Returns

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

emCoeGetEntryDesc Example

The following code demonstrates how to get the description of a specific object entry using the API `emCoeGetEntryDesc()`.

```

EC_T_BYTE*          pbyData = EC_NULL;
EC_T_COE_ENTRYDESC oEntryDesc;
OsMemset(&oEntryDesc, 0, sizeof(EC_T_COE_ENTRYDESC));

/* allocate payload memory */
pbyData = (EC_T_BYTE*)OsMalloc(CROD_ENTRYDESC_SIZE);
if (EC_NULL == pbyData)
{
    dwRetVal = EC_E_NOMEMORY;
    goto Exit;
}

dwRes = emCoeGetEntryDesc(dwInstanceId, dwSlaveId, 0x6411, 1, 0, pbyData, CROD_
↪ENTRYDESC_SIZE, &oEntryDesc, 5000);
if (dwRes != EC_E_NOERROR)
{
    dwRetVal = dwRes;
    goto Exit;
}

dwRetVal = dwRes;
Exit:
OsSafeFree(pbyData);

```

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:

- `emGetSlaveId()`
- See `CoeReadObjectDictionary()` in `EcSdoServices.cpp` as an example for `emCoeGetEntryDesc()`.

6.11.14 emCoeGetEntryDescReq

```
static EC_T_DWORD ecatCoeGetEntryDescReq (
    EC_T_MBXTFER *pMbxTfer,
    EC_T_DWORD dwSlaveId,
    EC_T_WORD wObIndex,
    EC_T_BYTE byObSubIndex,
    EC_T_BYTE byValueInfo,
    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_BYTE byObSubIndex,
    EC_T_BYTE byValueInfo,
    EC_T_DWORD dwTimeout
)
```

Initiates retrieval of the description of a specific object entry and returns immediately.

A unique transfer ID must be written into *EC_T_MBXTFER.dwTferId*. EC_NOTIFY_MBOXRCV is given on completion.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pMbxTfer** – [in] Mailbox transfer object
- **dwSlaveId** – [in] Slave ID
- **wObIndex** – [in] Object index
- **byObSubIndex** – [in] Object sub-index
- **byValueInfo** – [in] The value info bit mask includes which elements shall be in the response. See *Value info flags* for available values.
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_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*

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;

/* allocate payload memory */
oMbxTferDesc.pbyMbxTferDescData = (EC_T_BYTE*)OsMalloc (oMbxTferDesc.
↪dwMaxDataLen);
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)
{
    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);
}

OsSafeFree (oMbxTferDesc.pbyMbxTferDescData);

```

See also:

- `emNotify - eMbxTferType_COE_GETENTRYDESC`
- `emMbxTferCreate()`

- `emGetSlaveId()`

6.11.15 emNotify - eMbxTferType_COE_GETENTRYDESC

Completion of an SDO information service transfer to get an object entry description.

emNotify - eMbxTferType_COE_GETENTRYDESC

Parameter

- `pbyInBuf`: [in] Pointer to a structure of type `EC_T_MBXTFER`, containing 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 entry description stored in element `EC_T_MBX_DATA::CoE_EntryDesc` of type `EC_T_COE_ENTRYDESC` is part of `EC_T_MBXTFER::MbxData` and may have to be buffered by the client. Access to the memory area `EC_T_MBXTFER::MbxData` outside of the notification caller context is illegal and the results are undefined.

```
struct EC_T_COE_ENTRYDESC
```

Public Members

`EC_T_WORD wObIndex`

Index in the object dictionary

`EC_T_BYTE byObSubIndex`

Sub index in the object dictionary

`EC_T_BYTE byValueInfo`

Bit mask which information is included in `pbyData`. See *Value info flags*.

`EC_T_WORD wDataType`

Object data type according to ETG.1000

`EC_T_WORD wBitLen`

Object size (number of bits)

`EC_T_BYTE byObAccess`

Access rights. See *Object access flags*.

`EC_T_BOOL bRxPdoMapping`

Object is mappable in a RxPDO

`EC_T_BOOL bTxPdoMapping`

Object is mappable in a TxPDO

EC_T_BOOL bObjCanBeUsedForBackup

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.

6.11.16 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
- **dwChannel** – [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);

```

6.11.17 emNotify - EC_NOTIFY_COE_INIT_CMD

Indicates a COE mailbox transfer completion during SubDevice 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`, containing 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_BYTE byCcs`
Client command specifier (read or write access)

`EC_T_BOOL bCompleteAccess`
Complete access

`EC_T_WORD wIndex`
Object Index

EC_T_BYTE bySubIndex
Object SubIndex

EC_T_DWORD dwDataLen
InitCmd data length

EC_T_BYTE *pbyData
InitCmd data

See also:

emIoCtl - EC_IOCTL_SET_NOTIFICATION_ENABLED

6.11.18 CoE Emergency (emNotify - eMbxTferType_COE_EMERGENCY)

Indication of a CoE emergency request. An *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*, containing 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 EC-Master. The content in *EC_T_MBXTFER::dwTferId* is undefined as it is not needed by the client and the MainDevice. The transfer result is stored in *EC_T_MBXTFER::dwErrorCode*.

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

struct **EC_T_COE_EMERGENCY**

Public Members

EC_T_WORD wErrorCode
Error code according to EtherCAT specification

EC_T_BYTE byErrorRegister
Error register

EC_T_BYTE abyData[EC_COE_EMERGENCY_DATASIZE]
Error data

EC_T_WORD wStationAddress
Slave node address of the faulty slave

CoE Emergency Example

```

/* send CoE Emergency */
EC_T_BYTE abyEmergencyData[6] = { 0x01, 0x02, 0x03, 0x04, 0x05, 0x06 };
dwRes = esSendSlaveCoeEmergency(dwSimulatorId, wSlaveAddress, 0x1234 /* code */
↪, abyEmergencyData, 6 /* data length */);
↪ if (dwRes != EC_E_NOERROR)
    {
        EcLogMsg(EC_LOG_LEVEL_ERROR, (pEcLogContext, EC_LOG_LEVEL_ERROR,
↪ "esSendSlaveCoeEmergency failed: %s (0x%lx)\n", esGetText(dwSimulatorId, dwRes),
↪ dwRes));
        goto Exit;
    }

```

See also:

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

6.11.19 CoE Abort (emNotify - EC_NOTIFY_MBSLAVE_COE_SDO_ABORT)

The application can abort asynchronous CoE Uploads and Downloads. The SubDevice 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, containing 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 SdoAbort-Desc.

```
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()

6.11.20 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 EC-Master error codes to CoE error codes 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.

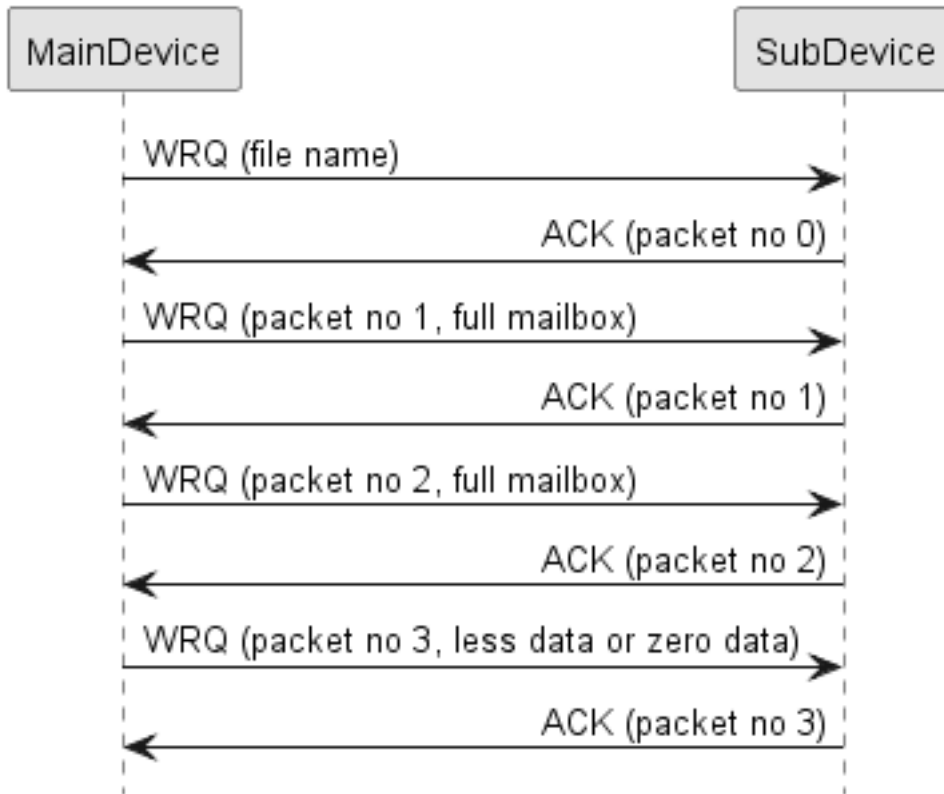
Reference:

- ETG.1000.5, ETG.1000.6 and ETG.5003.2

6.12.1 Specification

FoE file download

Regular download



Segmented download

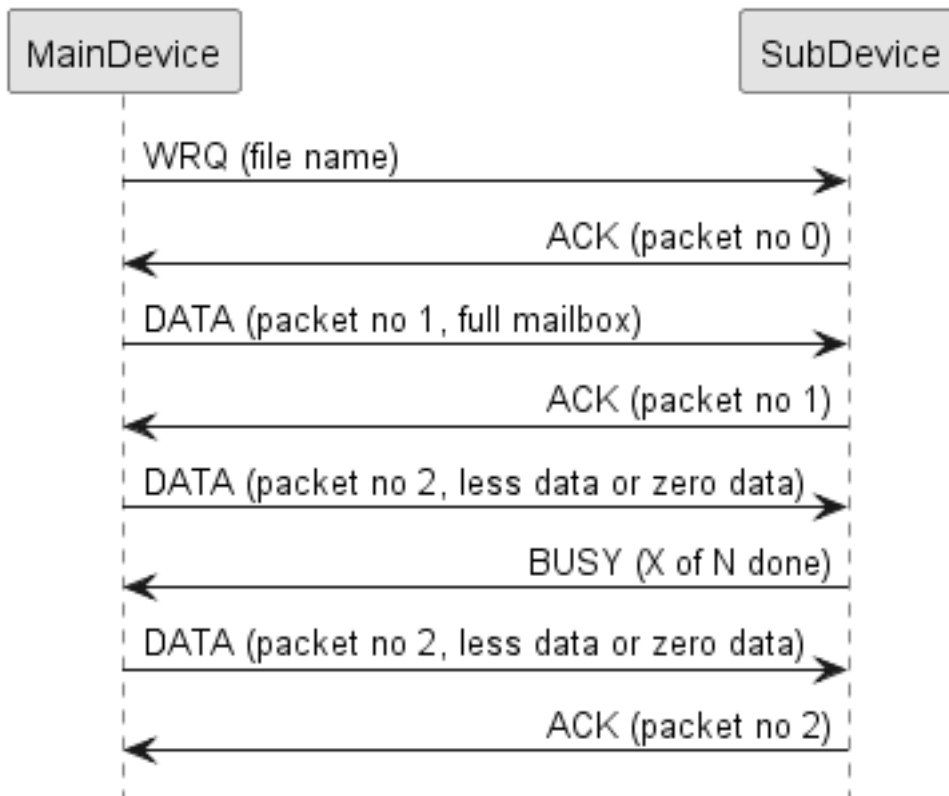
In case of segmented download the EC-Master raises `emNotify - EC_NOTIFY_MBOXRCV` in the [EC-Master Class B](#) documentation to request more data from the application after each ACK from the SubDevice. The notification handler may not block the EC-Master, e.g. due to reading from or writing to the file system, therefore applications typically do not handle `EC_NOTIFY_MBOXRCV` within the `JobTask` context. The segments are transferred using the SubDevice'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)

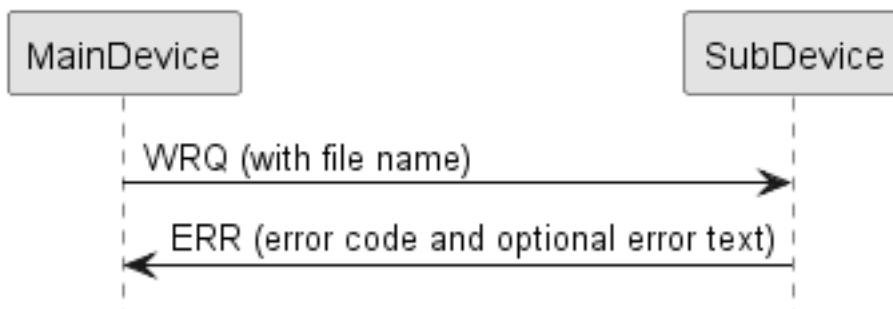
See also:

- `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

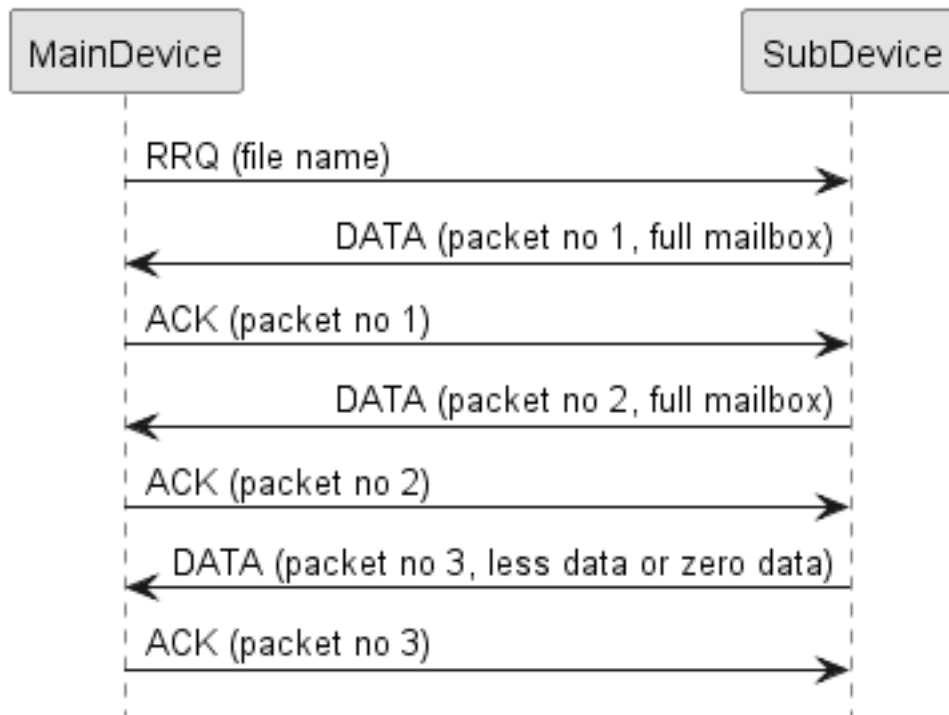


FoE file upload

The names of available files and their size are SubDevice 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 SubDevice’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 SubDevice 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 an 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 [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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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 EC-Master to a SubDevice, 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_BYTE* 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);
    }
    OsSafeFree(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_BYTE *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 an 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 [bytes]
- **pdwOutDataLen** – [out] Length of received data [byte]
- **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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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 EtherCAT stack 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()

The following example demonstrates how to perform a FoE Download.

emFoeDownloadReq() Example

This example code shows how to initiate file download to a SubDevice using emFoeDownloadReq(), 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_DWORD dwBufferSize = 0;

/* setup mbx transfers */
CEcMbxTfer MbxTfer1;
MbxTfer1.Create(FOE_BUFFER_SIZE);

/* read file to buffer */
pfLocalFile = OsFopen(MY_FOE_TRANSFER_LOCAL_FILENAME, "rb");
if (EC_NULL == pfLocalFile)
{
    dwRetVal = EC_E_OPENFAILED;
    goto Exit;
}
dwBufferSize = OsGetFileSize(pfLocalFile);

dwRes = (EC_T_DWORD)OsFread(MbxTfer1.GetMbxTfer()->pbyMbxTferData, 1,
↪dwBufferSize, pfLocalFile);
if (dwRes != dwBufferSize)
{
    dwRetVal = EC_E_ERROR;
    goto Exit;
}

/* download file to slave */
dwRes = ecatFoeDownloadReq(MbxTfer1.GetMbxTfer(), dwSlaveId, MY_FOE_TRANSFER_
↪SLAVE_FILENAME,
    (EC_T_DWORD)OsStrlen(MY_FOE_TRANSFER_SLAVE_FILENAME),
    0, 600000 /* 10 minutes */);
/* wait for transfer finished */
while (eMbxTferStatus_Pend == MbxTfer1.GetMbxTfer()->eTferStatus)
{
    OsSleep(10);
}

/* check transfer result */
dwRes = MbxTfer1.GetMbxTfer()->dwErrorCode;
if (EC_E_NOERROR != dwRes)
{
    dwRetVal = dwRes;
    goto Exit;
}

dwRetVal = EC_E_NOERROR;
Exit:
/* free resources */
if (EC_NULL != pfLocalFile)

```

(continues on next page)

(continued from previous page)

```

{
    OsFclose (pfLocalFile);
}

```

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] Overall timeout [ms] of the FoE transfer. Only evaluated when initiating the request.

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack 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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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] Overall timeout [ms] of the FoE transfer. Only evaluated when initiating the request.

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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 SubDevice with address *MY_FOE_SLAVE_ADDRESS* and store it in a file. The data uploaded from *MY_FOE_TRANSFER_SLAVE_FILENAME* of the SubDevice 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;

```

(continues on next page)

(continued from previous page)

```

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));
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))

```

(continues on next page)

(continued from previous page)

```

{
    /* 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;

        /* 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);
}
OsSafeFree (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_MBXSNW_WKC_ERROR

This error will be indicated in case the working counter of an FoE mailbox write command was not set to the expected value of 1. Detailed error information is stored in the structure *EC_T_WKCERR_DESC* of the union element WkcErrDesc.

struct *EC_T_WKCERR_DESC*

EC_T_SLAVE_PROP SlaveProp

EC_T_BYTE *byCmd*

EC_T_DWORD *dwAddr*

EC_T_WORD *wWkcSet*

EC_T_WORD *wWkcAct*

EC_T_DWORD *dwTaskId*

EC_T_WORD *wMsuld*

6.12.10 emNotify - EC_NOTIFY_FOE_MBSLAVE_ERROR

This error will be indicated in case an FoE mailbox SubDevice sends an error message. Detailed error information is stored in the 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 **achErrorString**[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 the 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 Profile 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 information about SoE see the IEC IEC61800-7-300 document 22G185eFDIS.

The 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 invoke fixed functional processes. When processing is finished, a notification will be received. To abort a running command execution *emSoeAbortProcCmd()* has to be called.
- **Notification:**
In case of a notification all registered clients will get this notification. A notification will be received when processing 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 SubDevice 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 indicate which elements of an IDN are read or written. The ElementFlags indicate 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 contains 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

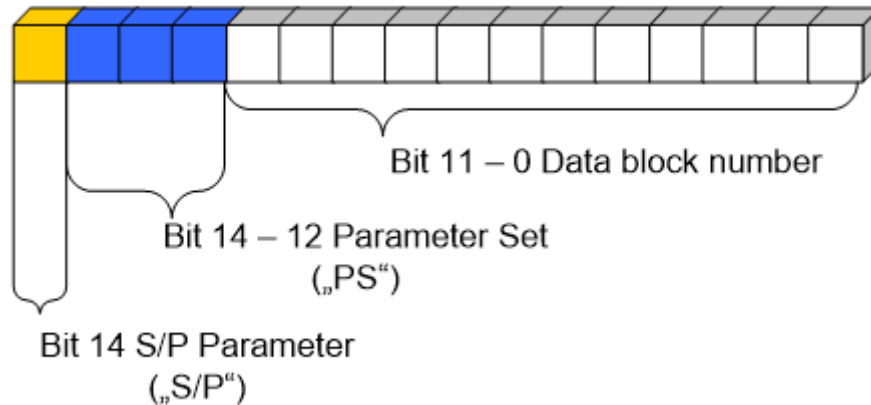
Operation data

SOE_BM_ELEMENTFLAG_DEFAULT

The IDN default value

6.13.2 SoE IDN coding

The parameter addressing area consists 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 (TN_{cyc}) which is a 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_BYTE byDriveNo,
    EC_T_BYTE *pbyElementFlags,
    EC_T_WORD wIDN,
    EC_T_BYTE *pbyData,
    EC_T_DWORD dwDataLen,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emSoeWrite (
    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 dwTimeout
)
```

Execute an SoE SSC Write service which downloads 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 an 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 [byte]
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* if dwInstanceID is out of range, the input pointer is EC_NULL or contains EC_NULL pointer, or dwTimeout is EC_NOWAIT
- *EC_E_NOMEMORY* if the mailbox protocol queue of the slave is full
- *EC_E_SLAVE_NOT_PRESENT* if slave not present
- *EC_E_NOTFOUND* if no slave matching dwSlaveId can be found
- *EC_E_NO_MBX_SUPPORT* if slave has no mailbox support
- *EC_E_INVALID_SLAVE_STATE* if slave is in an invalid state for mailbox transfer
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive
- *EC_E_ADS_IS_RUNNING* if ADS server is running
- *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 an SoE SCC Read service which uploads 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 [byte]
- **pdwOutDataLen** – [out] Length of received data [byte]
- **dwTimeout** – [in] Timeout [ms]

Returns

- *EC_E_NOERROR* if successful
- *EC_E_INVALIDSTATE* if EtherCAT stack 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
- *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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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
- *SoE error code*

See also:

- `emSoeRead()`
- `emGetSlaveId()`

6.13.7 emSoeAbortProcCmd

```
static EC_T_DWORD ecatSoeAbortProcCmd (
    EC_T_DWORD dwSlaveId,
    EC_T_BYTE byDriveNo,
    EC_T_BYTE *pbyElementFlags,
    EC_T_WORD wIDN,
    EC_T_DWORD dwTimeout
)
EC_T_DWORD emSoeAbortProcCmd (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwSlaveId,
    EC_T_BYTE byDriveNo,
    EC_T_BYTE *pbyElementFlags,
    EC_T_WORD wIdn,
    EC_T_DWORD dwTimeout
)
```

Abort SSC Procedure Command sequence.

A Procedure Command takes 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_MBXSNW_WKC_ERROR

This error will be indicated in case the working counter of an SoE mailbox write command was not set to the expected value of 1. Detailed error information is stored in the 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 an SoE mailbox write command responded with an error. Detailed error information is stored in the 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 [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], EC_NOWAIT returns the function immediately

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 [byte]
- **pdwOutDataLen** – [out] Length of received data [byte]
- **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 EtherCAT stack isn't initialized

- *EC_E_INVALIDPARG* if dwInstanceID is out of range
- *EC_E_TIMEOUT* if dwTimeout elapsed during the API call
- *EC_E_VOE_NO_MBX_RECEIVED* if no VoE data was 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_BYTE **byChnPri**

Channel, Priority

EC_T_BYTE **byTypCntRsvd**

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 SubDevice 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, containing 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 SubDevice. 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, containing 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 to 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. See EC_CMD_TYPE
- **dwMemoryAddress** – [in] Slave memory address, depending on the command to be sent this is either a physical or a 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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARG* 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* if another transfer request is already pending or the master or the corresponding slave is currently changing its operational state
- *EC_E_NOTREADY* if the working counter was not set when sending the command (slave may not be connected or did not respond)
- *EC_E_INVALIDSIZE* if the size of the complete command does not fit into a single Ethernet frame. The maximum amount of data to transfer must not exceed 1486 bytes.
- *EC_E_MASTER_RED_STATE_INACTIVE* if Master Redundancy is configured and master is inactive

The following EtherCAT® commands are supported:

- eRawCmd_APRD Auto Increment Physical Read (avoid to use, see below)
- eRawCmd_APWR Auto Increment Physical Write (avoid to use, see below)
- eRawCmd_APRW Auto Increment Physical Read/Write (avoid to use, see below)
- eRawCmd_FPRD Fixed addressed Physical Read
- eRawCmd_FPWR Fixed addressed Physical Write
- eRawCmd_FPRW Fixed addressed Physical Read/Write
- eRawCmd_BRD Broadcast (wire or'ed) Read
- eRawCmd_BWR Broadcast Write
- eRawCmd_BRW Broadcast Read/Write
- eRawCmd_LRD Logical Read
- eRawCmd_LWR Logical Write
- eRawCmd_LRW Logical Read/Write
- eRawCmd_ARMW Auto Increment Physical Read, multiple Write

6.15.2 emCIntSendRawMbx

```
static EC_T_DWORD ecatCIntSendRawMbx (
    EC_T_DWORD dwCIntId,
    EC_T_BYTE *pbyMbxCmd,
    EC_T_DWORD dwMbxCmdLen,
    EC_T_DWORD dwTimeout
)
```

```
EC_T_DWORD emCIntSendRawMbx (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwCIntId,
    EC_T_BYTE *pbyMbxCmd,
    EC_T_DWORD dwMbxCmdLen,
    EC_T_DWORD dwTimeout
)
```

Send a raw mailbox command.

Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **dwCIntId** – [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 emCIntQueueRawCmd

```
static EC_T_DWORD ecatCIntQueueRawCmd (
    EC_T_DWORD dwCIntId,
    EC_T_WORD wInvokeId,
    EC_T_BYTE byCmd,
    EC_T_DWORD dwMemoryAddress,
    EC_T_BYTE *pbyData,
    EC_T_WORD wLen
)
```

```
EC_T_DWORD emCIntQueueRawCmd (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD dwCIntId,
    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_PARAMS.dwEcatCmdTimeout` and `EC_T_INIT_MASTER_PARAMS.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 a 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 EtherCAT stack isn't initialized
- `EC_E_INVALIDPARG` 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_APRW**

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 a 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:

emClntQueueRawCmd()

6.15.5 emNotify - EC_NOTIFY_RAWCMD_DONE

This notification is given when the response to an *emTferSingleRawCmd()* or *emClntQueueRawCmd()* is received.

emNotify - EC_NOTIFY_RAWCMD_DONE**Parameter**

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

struct **EC_T_RAWCMDRESPONSE_NOTIFY_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_BYTE *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® MainDevice stack supports scanning the EtherCAT® Bus to determine which SubDevices are available. If a configuration was provided the connected SubDevices are validated against the given ENI.

See also:

emScanBus ()

6.16.1 emIoCtl - EC_IOCTL_SB_ENABLE

EC_IOCTL_SB_ENABLE

Enables Busscan support.

Parameters

- **pbyInBuf** – [in] Pointer to Timeout Parameter Value [ms] (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

Returns

EC_E_NOERROR or error code

The bus scan support is enabled by default.

6.16.2 emIoCtl - EC_IOCTL_SB_RESTART

EC_IOCTL_SB_RESTART

This call will restart the bus scanning cycle. On completion the Notification EC_NOTIFY_SB_STATUS is given.

Parameters

- **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

Returns

EC_E_NOERROR or error code

The timeout value given by *emIoCtl - 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 MainDevice configuration. This feature may be used to dynamically create or adjust the XML configuration file. When issuing this IoCtl, the application has to take care *emExecJob ()* is called cyclically to trigger MainDevice state machines, timers, send acyclic and receive frames accordingly.

6.16.3 emloCtl - EC_IOCTL_SB_STATUS_GET

EC_IOCTL_SB_STATUS_GET

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

Parameters

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

Returns

EC_E_NOERROR or error code

See also:

emNotify - EC_NOTIFY_SB_STATUS

6.16.4 emloCtl - EC_IOCTL_SB_SET_TOPOLOGY_CHANGED_DELAY

EC_IOCTL_SB_SET_TOPOLOGY_CHANGED_DELAY

This call will set the topology changed delay value. The master will wait this duration in ms to react on appearing links in topology. The default value is 1000 ms.

Parameters

- **pbyInBuf** – [in] Pointer to EC_T_DWORD containing the delay information in ms
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns

EC_E_NOERROR or error code

6.16.5 emloCtl - EC_IOCTL_SB_SET_TOPOLOGY_CHANGED_DELAYS

EC_IOCTL_SB_SET_TOPOLOGY_CHANGED_DELAYS

This call will set the topology changed delay values individually. The master will wait individual durations in ms (0 ms: disabled) for slave ports, main link and red link to react on appearing links in topology. The default value is 1000 ms.

Parameters

- **pbyInBuf** – [in] Pointer to *EC_T_TOPOLOGY_CHANGED_DELAYS* containing the delay information in ms
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL

- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns

EC_E_NOERROR or error code

struct **EC_T_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 emIoCtl - EC_IOCTL_SB_SET_ERROR_ON_CROSSED_LINES

EC_IOCTL_SB_SET_ERROR_ON_CROSSED_LINES

This call will enable or disable bus mismatch if IN and OUT connectors are swapped. If enabled the swapped IN and OUT connectors will lead to bus mismatch. By default swapped IN and OUT connectors will lead to bus mismatch.

Parameters

- **pbyInBuf** – [in] Pointer to EC_T_BOOL variable. If set to EC_TRUE swapped IN and OUT connectors will lead to bus mismatch, if set to EC_FALSE swapped IN and OUT connectors are tolerated.
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns

EC_E_NOERROR or error code

6.16.7 emIoctl - EC_IOCTL_SB_SET_ERROR_ON_LINE_BREAK

EC_IOCTL_SB_SET_ERROR_ON_LINE_BREAK

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.

Note: *EC_E_REDLINEBREAK*: Line break in a cable redundant network.

Note: *EC_E_JUNCTION_RED_LINE_BREAK*: Line break in a junction redundant network.

Note: See EC_NOTIFY_SB_MISMATCH, EC_NOTIFY_SB_STATUS.

Parameters

- **pbyInBuf** – [in] Pointer to value of EC_T_BOOL: EC_TRUE: Enable, EC_FALSE: Disable. Default: Enabled.
- **dwInBufSize** – [in] Should be set to sizeof(EC_T_BOOL)
- **pbyOutBuf** – [out] Should be set to EC_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC_NULL

Returns

EC_E_NOERROR or error code

6.16.8 emIoctl - EC_IOCTL_SB_SET_TOPOLOGY_CHANGE_AUTO_MODE

Warning: This documentation is preliminary and is subject to change

EC_IOCTL_SB_SET_TOPOLOGY_CHANGE_AUTO_MODE

This call will enable or disable the automatic topology change mode. By default the automatic mode is enabled.

Parameters

- **pbyInBuf** – [in] Pointer to EC_T_BOOL variable. If set to EC_TRUE the automatic 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

Returns

EC_E_NOERROR or error code

In automatic mode, new SubDevices will be discovered automatically. In manual mode, after new SubDevices have been connected, an *emNotify - EC_NOTIFY_HC_TOPOCHGDONE* notification will be given without opening the

ports of the SubDevices on the bus. When the application is able to handle the new SubDevices, it should call *emIoctl* - *EC_IOCTL_SB_ACCEPT_TOPOLOGY_CHANGE*.

6.16.9 emIoctl - EC_IOCTL_SB_ACCEPT_TOPOLOGY_CHANGE

Warning: This documentation is preliminary and is subject to change

EC_IOCTL_SB_ACCEPT_TOPOLOGY_CHANGE

This call will trigger a scan bus. On completion the Notification *EC_NOTIFY_SB_STATUS* is given.

Parameters

- **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*

Returns

EC_E_NOERROR or error code

This function may be called after an *emNotify* - *EC_NOTIFY_HC_TOPOCHGDONE* notification was given if the automatic topology change mode was previously disabled using *emIoctl* - *EC_IOCTL_SB_SET_TOPOLOGY_CHANGE_AUTO_MODE*. During this bus scan the ports of the SubDevices will (re)open and new SubDevices can be detected. The timeout value given by *emIoctl* - *EC_IOCTL_SB_ENABLE* will be used. When issuing this Ioctl, the application has to take care *emExecJob()* is called cyclically to trigger MainDevice state machines, timers, send acyc and receive frames accordingly.

6.16.10 emNotify - EC_NOTIFY_SB_STATUS

Bus scan status notification.

This notification is enabled by default.

emNotify - EC_NOTIFY_SB_STATUS

Parameter

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

struct **EC_T_SB_STATUS_NOTIFY_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:

emIoCtl - 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 a bus scan detects a mismatch of connected SubDevices and configuration, due to unexpected SubDevices or missing mandatory SubDevices.

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 SubDevices can be found although the SubDevices 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:

emIoCtl - *EC_IOCTL_SET_NOTIFICATION_ENABLED* for how to control the deactivation

6.16.12 emNotify - EC_NOTIFY_SB_DUPLICATE_HC_NODE

A bus mismatch was detected during the scan due to duplicated SubDevice(s). The application receives this notification if there are two SubDevices 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 SubDevice (config) station Address
- `EC_T_SB_MISMATCH_DESC::wCfgAIncAddress` Duplicated SubDevice (config) Auto-Increment Address.
- `EC_T_SB_MISMATCH_DESC::dwCfgVendorId` Duplicated SubDevice (config) Vendor ID
- `EC_T_SB_MISMATCH_DESC::dwCfgProdCode` Duplicated SubDevice (config) Product code
- `EC_T_SB_MISMATCH_DESC::dwCfgRevisionNo` Duplicated SubDevice (config) Revision Number
- `EC_T_SB_MISMATCH_DESC::dwCfgSerialNo` Duplicated SubDevice (config) Serial Number

6.16.13 emNotify - EC_NOTIFY_SLAVE_PRESENCE

This notification is given when a SubDevice appears or disappears from the network.

This notification is enabled by default.

emNotify - EC_NOTIFY_SLAVE_PRESENCE

Parameter

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

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

struct `EC_T_SLAVE_PRESENCE_NOTIFY_DESC`

Public Members

`EC_T_WORD wStationAddress`
Slave station address

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

See also:

emIoCtl - `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 the MainDevice status is changed, whichever comes first. Disconnecting SubDevices 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_NOTIFY_DESC`
- `dwInBufSize`: [in] Size of the input buffer provided at `pbyInBuf` in bytes
- `pbyOutBuf`: [out] Should be set to `EC_NULL`
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to `EC_NULL`

```
struct EC_T_SLAVES_PRESENCE_NOTIFY_DESC
```

Public Members

`EC_T_WORD wCount`
Number of slave presence notifications

`EC_T_SLAVE_PRESENCE_NOTIFY_DESC`
SlavePresence[MAX_SLAVES_PRESENCE_NOTIFY_ENTRIES]
Slave presence descriptions

See also:

`emIoCtl - EC_IOCTL_SET_NOTIFICATION_ENABLED`

6.16.15 emNotify - EC_NOTIFY_LINE_CROSSED

Cable swapping detected. All SubDevice's port 0 must lead to MainDevice.

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 (EC-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 a port would lead to frame loss. This notification should only print a log message.

emNotify - EC_NOTIFY_FRAMELOSS_AFTER_SLAVE

Parameter

- *pbyInBuf*: [in] Pointer to *EC_T_ERROR_NOTIFICATION_DESC* containing *EC_T_FRAMELOSS_AFTER_SLAVE_NOTIFY_DESC*
- *dwInBufSize*: [in] Size of the input buffer provided at *pbyInBuf* in bytes
- *pbyOutBuf*: [out] Should be set to *EC_NULL*
- *dwOutBufSize*: [in] Should be set to 0
- *pdwNumOutData*: [out] Should be set to *EC_NULL*

```
struct EC_T_FRAMELOSS_AFTER_SLAVE_NOTIFY_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_DETECTADGROUPS`, `EC_NOTIFY_HC_PROBEALLGROUPS` belong to the Feature Pack Hot Connect.

6.16.19 emIoctl - EC_IOCTL_SB_NOTIFY_UNEXPECTED_BUS_SLAVES

EC_IOCTL_SB_NOTIFY_UNEXPECTED_BUS_SLAVES

Specifies if unexpected bus slaves must be notified as bus mismatch.

Parameters

- **pbyInBuf** – [in] Pointer to `EC_T_BOOL` variable. If set to `EC_TRUE` unexpected bus slaves on the network will be notified by `EC_NOTIFY_SB_MISMATCH`.
- **dwInBufSize** – [in] Size of the input buffer provided at `pbyInBuf` in bytes
- **pbyOutBuf** – [out] Should be set to `EC_NULL`
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to `EC_NULL`

Returns

EC_E_NOERROR or error code

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 EtherCAT stack isn't initialized
- *EC_E_INVALIDPARM* 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 a topology change was 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 SubDevices have been detected and DC initialized.

6.16.22 emIoctl - EC_IOCTL_SB_SET_NO_DC_SLAVES_AFTER_JUNCTION

EC_IOCTL_SB_SET_NO_DC_SLAVES_AFTER_JUNCTION

Declares that no DC slaves are located after junction.

Parameters

- **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

Returns

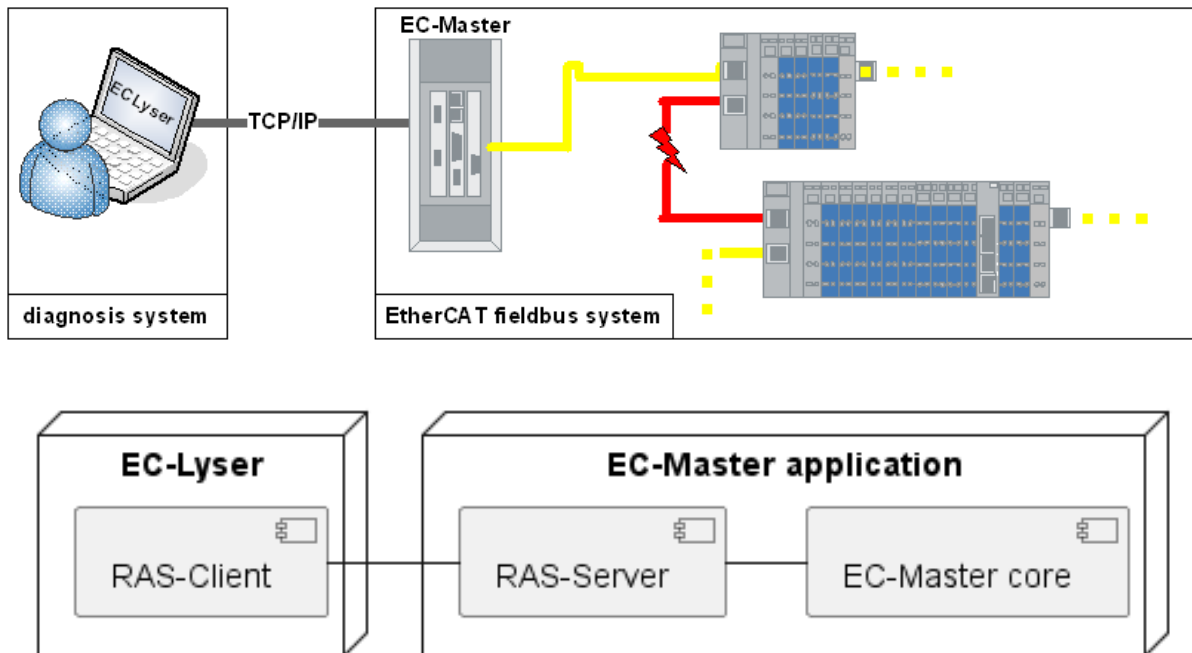
EC_E_NOERROR or error code

Calling this IOCTL if DC SubDevices 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 (
    EC_T_RAS_SERVER_PARMS *pParms,
    EC_T_PVOID *ppHandle
)
```

Initializes and starts a remote API Server Instance.

Parameters

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

Returns

EC_E_NOERROR or error code

struct **EC_T_RAS_SERVER_PARMS**

Public Members

EC_T_DWORD **dwSignature**

[in] Set to EC_RAS_SERVER_SIGNATURE

EC_T_DWORD **dwSize**

[in] Set to sizeof(EC_T_RAS_SERVER_PARMS)

EC_T_LOG_PARMS **LogParms**

[in] Logging parameters

EC_T_IPADDR **oAddr**

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

EC_T_WORD **wPort**

[in] Remote Access Server (RAS) listen port

EC_T_WORD **wMaxClientCnt**

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

EC_T_DWORD **dwCycleTime**

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

EC_T_DWORD **dwCommunicationTimeout**

[in] Timeout [ms] before automatically closing connection

EC_T_CPUSET **oAcceptorThreadCpuAffinityMask**

[in] Acceptor Thread CPU affinity mask

EC_T_DWORD **dwAcceptorThreadPrio**

[in] Acceptor Thread Priority

EC_T_DWORD **dwAcceptorThreadStackSize**

[in] Acceptor Thread Stack Size

EC_T_CPUSET **oClientWorkerThreadCpuAffinityMask**

[in] Client Worker Thread CPU affinity mask

EC_T_DWORD **dwClientWorkerThreadPrio**

[in] Client Worker Thread Priority

EC_T_DWORD **dwClientWorkerThreadStackSize**

[in] Client Worker Thread Stack Size

EC_T_DWORD **dwMaxQueuedNotificationCnt**

[in] Amount of concurrently queue able Notifications

EC_T_DWORD dwMaxParallelMbxTferCnt
 [in] Amount of concurrent active mailbox transfers

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

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

EC_T_DWORD dwCycErrInterval
 [in] Interval which allows cyclic Notifications

EC_T_DWORD dwMaxQueuedNotificationSize
 [in] Size of concurrent active mailbox transfers

EC_PF_CHECK_TOKEN pfCheckToken
 [in] Function pointer called to check token

EC_T_VOID *pvCheckTokenContext
 [in] Check token context

EC_T_BYTE *pbyTlsCert
 [in] TLS certificate filename string

EC_T_DWORD dwTlsCertSize
 [in] Size of TLS certificate

EC_T_TLS_CERT_TYPE eTlsCertType
 [in] TLS certificate type

EC_T_BYTE *pbyTlsPrivKey
 [in] TLS private key filename string

EC_T_DWORD dwTlsPrivKeySize
 [in] Size of TLS private key

EC_T_TLS_PRIVKEY_TYPE eTlsPrivKeyType
 [in] TLS certificate type

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

Public Members

EC_T_INNER_IPADDR sAddr
 IPv4 address (endianness independent)

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

struct **EC_T_INNER_IPADDR**

Public Members

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

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 - EC_RAS_NOTIFY_CONNECTION

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

emRasNotify - EC_T_RAS_CONNOTIFYDESC

Parameter

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

```
struct EC_T_RAS_CONNOTIFYDESC
```

Public Members

EC_T_DWORD **dwCause**

[in] Cause of state connection state change

EC_T_DWORD **dwCookie**

[in] Unique identification cookie of connection instance

7.2.5 emRasNotify - EC_RAS_NOTIFY_REGISTER

Notification that a connected application registered a client to the EC-Master stack.

emRasNotify - EC_RAS_NOTIFY_REGISTER

Parameter

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

struct **EC_T_RAS_REGNOTIFYDESC**

Public Members

EC_T_DWORD **dwCookie**

[in] Unique identification cookie of connection instance

EC_T_DWORD **dwResult**

[in] Result of registration request

EC_T_DWORD **dwInstanceId**

[in] Master Instance client registered to

EC_T_DWORD **dwClientId**

[in] Client ID of registered client

7.2.6 emRasNotify - EC_RAS_NOTIFY_UNREGISTER

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

emRasNotify - EC_RAS_NOTIFY_UNREGISTER

Parameter

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

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

See also:*EC_T_RAS_REGNOTIFYDESC***7.2.7 emRasNotify - EC_RAS_NOTIFY_MARSHALERROR**

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

emRasNotify - EC_RAS_NOTIFY_MARSHALERRORDESC**Parameter**

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

struct **EC_T_RAS_MARSHALERRORDESC**

Public Members

EC_T_DWORD dwCookie
[in] Unique identification cookie of connection instance

EC_T_DWORD dwCause
[in] Cause of the command marshalling error

EC_T_DWORD dwLenStatCmd
[in] Length of the faulty command

EC_T_DWORD dwCommandCode
[in] Command code of the faulty command

7.2.8 emRasNotify - EC_RAS_NOTIFY_ACKERROR

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

emRasNotify - EC_RAS_NOTIFY_ACKERROR**Parameter**

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

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

7.2.9 `emRasNotify - EC_RAS_NOTIFY_NONOTIFYMEMORY`

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

`emRasNotify - EC_RAS_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 - EC_RAS_NOTIFY_STDNOTIFYMEMORYSMALL`

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

`emRasNotify - EC_RAS_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 - EC_RAS_NOTIFY_MBXNOTIFYMEMORYSMALL`

Notification given, when the buffer size for Mailbox notifications available in pre-allocated notification store is 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 become out of sync and therefore no longer usable. Mailbox notifications should be dimensioned correctly, see `emRasSrvStart()`.

`emRasNotify - EC_RAS_NOTIFY_MBXNOTIFYMEMORYSMALL`

Parameter

- `pbyInBuf`: [in] Pointer to `EC_T_DWORD` containing unique identification cookie of connection instance
- `dwInBufSize`: [in] Size of the input buffer in bytes

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

8 Error Codes

8.1 Groups

No.	Group	Abbr.	Description
1	Application Error	APP	Error within application, running the MainDevice E.g. API function call with invalid parameters
2	EtherCAT® network information file problem	ENI	MainDevice configuration XML file mismatches SubDevice configuration on bus E.g. Bus Topology Scan cannot detect all SubDevices configured within network information file
3	MainDevice parameter configuration	CFG	MainDevice configuration parameters erroneous E.g. mailbox command queue not large enough
4	Bus/SubDevice Error	SLV	SubDevice error E.g. Working Counter Error
5	Real-time Ethernet Driver	LLA	Real-time Ethernet Driver error (network interface driver) E.g. Intel Pro/1000 NIC could not be found
6	Remote API	RAS	Remote API error E.g. connection to Remote API server is not possible from client
7	Internal software error	ISW	MainDevice internal error E.g. MainDevice state machine in undefined state
8	DC MainDevice Sync	DCM	DC SubDevice 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. EtherCAT stack should abort current mailbox transfer)

EC_E_INVALIDSIZE

0x98110005: APP: Invalid size

EC_E_INVALIDDATA

0x98110006: ISW: Invalid data (multiple error sources)

EC_E_NOTREADY

0x98110007: ISW: Not ready (multiple error sources)

EC_E_BUSY

0x98110008: APP: Busy (e.g. EtherCAT stack is currently busy and not available to process the API request. The function may be called again later)

EC_E_ACYC_FRM_FREEQ_EMPTY

0x98110009: ISW: Cannot queue acyclic EtherCAT command (Acyclic command queue is full. Possible solution: Increase of configuration value dwMaxQueuedEthFrames)

EC_E_NOMEMORY

0x9811000A: CFG: No memory left (e.g. memory full / fragmented))

EC_E_INVALIDPARM

0x9811000B: APP: Invalid parameter (e.g. API function called with erroneous parameter set)

EC_E_NOTFOUND

0x9811000C: APP: Not found (e.g. Network Information File ENI not found or API called with invalid slave ID)

EC_E_DUPLICATE

0x9811000D: ISW: Duplicated fixed address detected (handled internally)

EC_E_INVALIDSTATE

0x9811000E: ISW: Invalid state (EtherCAT stack not initialized or not configured)

EC_E_TIMER_LIST_FULL

0x9811000F: ISW: Cannot add slave to timer list (slave timer list full)

EC_E_TIMEOUT

0x98110010: Timeout

EC_E_OPENFAILED

0x98110011: ISW: Open failed

EC_E_SENDFAILED

0x98110012: LLA: Frame send failed

EC_E_INSERTMAILBOX

0x98110013: CFG: Insert Mailbox error (internal limit MAX_QUEUED_COE_CMDS: 20)

EC_E_INVALIDCMD

0x98110014: ISW: Invalid Command (Unknown mailbox command code)

EC_E_UNKNOWN_MBX_PROTOCOL

0x98110015: ISW: Unknown Mailbox Protocol Command (Unknown Mailbox protocol or mailbox command with unknown protocol association)

EC_E_ACCESSDENIED

0x98110016: ISW: Access Denied (e.g. master internal software error)

EC_E_IDENTIFICATIONFAILED

0x98110017: ENI: Identification failed (e.g. identification command failed)

EC_E_LOCK_CREATE_FAILED

0x98110018: SYS: Create lock failed (e.g. OsCreateLockTyped failed)

EC_E_VERSION_MISMATCH

0x98110019: APP: Version mismatch for loaded library

EC_E_PRODKEY_INVALID

0x9811001A: CFG: Product Key Invalid (e.g. application using protected version of the stack, which stops operation after the evaluation time limit reached if a license is not provided)

EC_E_WRONG_FORMAT

0x9811001B: ENI: Wrong configuration format (e.g. Network information file empty or malformed), SLV: Malformed EEPROM content

EC_E_FEATURE_DISABLED

0x9811001C: APP: Feature disabled (e.g. Application tried to perform a missing or disabled API function)

EC_E_SHADOW_MEMORY

0x9811001D: Shadow memory requested in wrong mode

EC_E_BUSCONFIG_MISMATCH

0x9811001E: ENI: Bus configuration mismatch (e.g. Network information file and currently connected bus topology does not match)

EC_E_CONFIGDATAREAD

0x9811001F: ENI: Error reading configuration file (e.g. Network information file could not be read)

EC_E_ENI_NO_SAFEOP_OP_SUPPORT

0x98110020: Configuration doesn't support SAFEOP and OP requested state

EC_E_XML_CYCCMDS_MISSING

0x98110021: ENI: Cyclic commands are missing (e.g. Network information file does not contain cyclic commands)

EC_E_XML_ALSTATUS_READ_MISSING

0x98110022: ENI: AL_STATUS register read missing in XML file for at least one state (e.g. Read of AL Status register is missing in cyclic part of given network information file)

EC_E_MCSM_FATAL_ERROR

0x98110023: ISW: Fatal internal McSm (master control state machine is in an undefined state)

EC_E_SLAVE_ERROR

0x98110024: SLV: Slave error (e.g. A slave error was detected. See also EC_NOTIFY_STATUS_SLAVE_ERROR and EC_NOTIFY_SLAVE_ERROR_STATUS_INFO)

EC_E_FRAME_LOST

0x98110025: SLV: Frame lost, IDX mismatch (EtherCAT *frame(s)* lost on bus, means the response was not received. In case this error shows frequently a problem with the wiring could be the cause)

EC_E_CMD_MISSING

0x98110026: SLV: At least one EtherCAT command is missing in the received frame (e.g. received EtherCAT frame incomplete)

EC_E_CYCCMD_WKC_ERROR

0x98110027: Cyclic command WKC error

EC_E_INVALID_DCL_MODE

0x98110028: APP: IOCTL EC_IOCTL_DC_LATCH_REQ_LTIMVALS invalid in DCL auto read mode (this function cannot be used if DC Latching is running in mode “Auto Read”)

EC_E_AI_ADDRESS

0x98110029: SLV: Auto increment address increment mismatch (e.g. Network information file and bus topology doesn't match any more. Error shows only, if an already recognized slave isn't present any more)

EC_E_INVALID_SLAVE_STATE

0x9811002A: APP: Slave in invalid state, e.g. not in OP (API not callable in this state) (mailbox commands are not allowed in current slave state)

EC_E_SLAVE_NOT_ADDRESSABLE

0x9811002B: SLV: Station address lost (or slave missing) - FPRD to AL_STATUS failed (e.g. Slave had a power cycle)

EC_E_CYC_CMDS_OVERFLOW

0x9811002C: ENI: Too many cyclic commands in XML configuration file (e.g. *EC_T_INIT_MASTER_PARAMS.dwMaxAcycFramesQueued* too small)

EC_E_LINK_DISCONNECTED

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

EC_E_MASTERCORE_INACCESSIBLE

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

EC_E_COE_MBXSEND_WKC_ERROR

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

EC_E_COE_MBXRCV_WKC_ERROR

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

EC_E_NO_MBX_SUPPORT

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

EC_E_NO_COE_SUPPORT

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

EC_E_NO_EOE_SUPPORT

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

EC_E_NO_FOE_SUPPORT

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

EC_E_NO_SOE_SUPPORT

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

EC_E_NO_VOE_SUPPORT

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

EC_E_EVAL_VIOLATION

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

EC_E_EVAL_EXPIRED

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

EC_E_LICENSE_MISSING

0x98110039: License key invalid or missing

EC_E_CFGFILENOTFOUND

0x98110070: CFG: Network configuration file not found (e.g. path to configuration file (XML) was wrong or the file is not available)

EC_E_EEPROMREADERROR

0x98110071: SLV: Command error while EEPROM upload (read slave EEPROM)

EC_E_EEPROMWRITEERROR

0x98110072: SLV: Command error while EEPROM download (write slave EEPROM)

EC_E_XML_CYCCMDS_SIZEMISMATCH

0x98110073: ENI: Cyclic command wrong size (too long) (size in network configuration file (XML) does not match size of process data)

EC_E_XML_INVALID_INP_OFF

0x98110074: ENI: Invalid input offset in cyclic command, please check InputOffs

EC_E_XML_INVALID_OUT_OFF

0x98110075: ENI: Invalid output offset in cyclic command, please check OutputOffs

EC_E_PORTCLOSE

0x98110076: Port close failed

EC_E_PORTOPEN

0x98110077: Port open failed

EC_E_SLAVE_NOT_PRESENT

0x9811010E: APP / SLV: command not executed (slave not present on bus) (e.g. slave disappeared or was never present)

EC_E_EEPROMRELOADERROR

0x98110110: Command error while EEPROM reload

EC_E_SLAVECTRLRESETERROR

0x98110111: Command error while Reset Slave Controller

EC_E_SYSDRIVERMISSING

0x98110112: SYS: Cannot open system driver (e.g. system driver was not loaded)

EC_E_BUSCONFIG_TOPOCHANGE

0x9811011E: Bus configuration not detected, Topology changed (e.g. Topology changed while scanning bus)

EC_E_EOE_MBX_WKC_ERROR

0x9811011F: EoE: Mailbox receive: working counter

EC_E_FOE_MBX_WKC_ERROR

0x98110120: FoE: Mailbox receive: working counter

EC_E_SOE_MBX_WKC_ERROR

0x98110121: SoE: mailbox receive: working counter

EC_E_AOE_MBX_WKC_ERROR

0x98110122: AoE: Mailbox receive: working counter

EC_E_VOE_MBX_WKC_ERROR

0x98110123: SLV: VoE mailbox send: working counter (VoE mailbox couldn't be written)

EC_E_EEPROMASSIGNERROR

0x98110124: SLV: EEPROM assignment failed

EC_E_MBX_ERROR_TYPE

0x98110125: SLV: Unknown mailbox error code received in mailbox

EC_E_REDLINEBREAK

0x98110126: SLV: Redundancy line break (e.g. cable break between slaves or between EtherCAT network adapter and first slave)

EC_E_XML_INVALID_CMD_WITH_RED

0x98110127: ENI: Invalid EtherCAT command in cyclic frame with redundancy (e.g. BRW commands are not allowed with redundancy)

EC_E_XML_PREV_PORT_MISSING

0x98110128: ENI: <PreviousPort>-tag is missing (e.g. if the auto increment address is not the first slave on the bus we check if a previous port tag OR a hot connect tag is available)

EC_E_XML_DC_CYCCMDS_MISSING

0x98110129: DC enabled and DC cyclic commands missing (e.g. access to 0x0900)

EC_E_DLSTATUS_IRQ_TOPOCHANGED

0x98110130: SLV: Data link (DL) status interrupt because of changed topology (automatically handled by master)

EC_E_PTS_IS_NOT_RUNNING

0x98110131: PTS: Pass Through Server is not running (Pass-Through-Server was tried to be enabled/disabled or stopped without being started)

EC_E_PTS_IS_RUNNING

0x98110132: PTS: Pass Through Server is running (obsolete, replaced by EC_E_ADS_IS_RUNNING)

EC_E_ADS_IS_RUNNING

0x98110132: PTS: ADS adapter (Pass Through Server) is running (API call conflicts with ADS state (running))

EC_E_PTS_THREAD_CREATE_FAILED

0x98110133: PTS: Could not start the Pass Through Server

EC_E_PTS SOCK_BIND_FAILED

0x98110134: PTS: The Pass Through Server could not bind the IP address with a socket (e.g. Possibly because the IPaddress (and Port) is already in use or the IP-address does not exist)

EC_E_PTS_NOT_ENABLED

0x98110135: PTS: The Pass Through Server is running but not enabled

EC_E_PTS_LL_MODE_NOT_SUPPORTED

0x98110136: PTS: The Link Layer mode is not supported by the Pass Through Server (e.g. The Master is running in interrupt mode but the Pass-Through-Server only supports polling mode)

EC_E_VOE_NO_MBX_RECEIVED

0x98110137: SLV: No VoE mailbox received yet from specific slave

EC_E_DC_REF_CLOCK_SYNC_OUT_UNIT_DISABLED

0x98110138: DC (time loop control) unit of reference clock disabled

EC_E_DC_REF_CLOCK_NOT_FOUND

0x98110139: SLV: Reference clock not found! May happen if reference clock is removed from network.

EC_E_MBX_CMD_WKC_ERROR

0x9811013B: SLV: Mailbox command working counter error (e.g. Mailbox init command Retry Count exceeded)

EC_E_NO_AOE_SUPPORT

0x9811013C: APP / SLV: AoE: Protocol not supported (e.g. Application calls AoE-API although not implemented at slave)

EC_E_AOE_INV_RESPONSE_SIZE

0x9811013D: AoE: Invalid AoE response received

EC_E_AOE_ERROR

0x9811013E: AoE: Common AoE device error

EC_E_AOE_SRVNOTSUPP

0x9811013F: AoE: Service not supported by server

EC_E_AOE_INVALIDGRP

0x98110140: AoE: Invalid index group

EC_E_AOE_INVALIDOFFSET

0x98110141: AoE: Invalid index offset

EC_E_AOE_INVALIDACCESS

0x98110142: AoE: Reading/writing not permitted

EC_E_AOE_INVALIDSIZE

0x98110143: AoE: Parameter size not correct

EC_E_AOE_INVALIDDATA0x98110144: 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_INVALIDPARM0x98110149: 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_NOMOREHDLIS

0x98110154: AoE: No more notification handles

EC_E_AOE_INVALIDWATCHSIZE

0x98110155: AoE: Size for watch to big

EC_E_AOE_NOTINIT

0x98110156: AoE: Device not initialized

EC_E_AOE_TIMEOUT

0x98110157: AoE: Device has a timeout

EC_E_AOE_NOINTERFACE

0x98110158: AoE: Query interface failed

EC_E_AOE_INVALIDINTERFACE

0x98110159: AoE: Wrong interface required

EC_E_AOE_INVALIDCLSID

0x9811015A: AoE: Class ID invalid

EC_E_AOE_INVALIDOBJID

0x9811015B: AoE: Object ID invalid

EC_E_AOE_PENDING

0x9811015C: AoE: Request pending

EC_E_AOE_ABORTED

0x9811015D: AoE: Request aborted

EC_E_AOE_WARNING

0x9811015E: AoE: Signal warning

EC_E_AOE_INVALIDARRAYIDX

0x9811015F: AoE: Invalid array index

EC_E_AOE_SYMBOLNOTACTIVE

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

EC_E_AOE_ACCESSDENIED

0x98110161: AoE: Access denied

EC_E_AOE_INTERNAL

0x98110162: AoE: Internal error

EC_E_AOE_TARGET_PORT_NOT_FOUND

0x98110163: AoE: Target port not found

EC_E_AOE_TARGET_MACHINE_NOT_FOUND

0x98110164: AoE: Target machine not found

EC_E_AOE_UNKNOWN_CMD_ID

0x98110165: AoE: Unknown command ID

EC_E_AOE_PORT_NOT_CONNECTED

0x98110166: AoE: Port not connected

EC_E_AOE_INVALID_AMS_LENGTH

0x98110167: AoE: Invalid AMS length

EC_E_AOE_INVALID_AMS_ID

0x98110168: AoE: invalid AMS Net ID

EC_E_AOE_PORT_DISABLED

0x98110169: AoE: Port disabled

EC_E_AOE_PORT_CONNECTED

0x9811016A: AoE: Port already connected

EC_E_AOE_INVALID_AMS_PORT

0x9811016B: AoE: Invalid AMS port

EC_E_AOE_NO_MEMORY

0x9811016C: AoE: No memory

EC_E_AOE_VENDOR_SPECIFIC

0x9811016D: AoE: Vendor specific AoE device error

EC_E_XML_AOE_NETID_INVALID

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

EC_E_MAX_BUS_SLAVES_EXCEEDED

0x9811016F: CFG: Error: Maximum number of bus slave has been exceeded (The maximum number of preallocated bus slave objects is too small. The maximum number can be adjusted by the master initialization parameter EC_T_INITMASTERPARMS.dwMaxBusSlaves)

EC_E_MBXERR_SYNTAX

0x98110170: SLV: Mailbox error: Syntax of 6 octet Mailbox header is wrong (Slave error mailbox return value: 0x01)

EC_E_MBXERR_UNSUPPORTEDPROTOCOL

0x98110171: SLV: Mailbox error: The Mailbox protocol is not supported (Slave error mailbox return value: 0x02)

EC_E_MBXERR_INVALIDCHANNEL

0x98110172: SLV: Mailbox error: Field contains wrong value (Slave error mailbox return value: 0x03)

EC_E_MBXERR_SERVICENOTSUPPORTED

0x98110173: SLV: Mailbox error: The addressed service in the mailbox protocol is not supported (Slave error mailbox return value: 0x04)

EC_E_MBXERR_INVALIDHEADER

0x98110174: SLV: Mailbox error: The mailbox protocol header of the mailbox protocol is wrong (Slave error mailbox return value: 0x05)

EC_E_MBXERR_SIZETOOSHORT

0x98110175: SLV: Mailbox error: Length of received mailbox data is too short (Slave error mailbox return value: 0x06)

EC_E_MBXERR_NOMOREMEMORY

0x98110176: SLV: Mailbox error: Mailbox protocol can not be processed because of limited resources (Slave

error mailbox return value: 0x07)

EC_E_MBXERR_INVALIDSIZE

0x98110177: SLV: Mailbox error: The length of data is inconsistent (Slave error mailbox return value: 0x08)

EC_E_DC_SLAVES_BEFORE_REF_CLOCK

0x98110178: ENI: Slaves with DC configured present on bus before reference clock (e.g. The first DC Slave was not configured as potential reference clock)

EC_E_DATA_TYPE_CONVERSION_FAILED

0x98110179: Data type conversion failed

EC_E_LINE_CROSSED

0x9811017B: Line crossed (cabling wrong)

EC_E_LINE_CROSSED_SLAVE_INFO

0x9811017C: Line crossed at slave (obsolete)

EC_E_ADO_NOT_SUPPORTED

0x9811017E: SLV: ADO for slave identification not supported (e.g. Request ID mechanism (ADO 0x134) not supported by slave)

EC_E_FRAMELOSS_AFTER_SLAVE

0x9811017F: Frameloss after Slave (opening port destroys communication)

EC_E_OEM_SIGNATURE_MISMATCH

0x98130008: ENI, OEM: Manufacturer signature mismatch

EC_E_ENI_ENCRYPTION_WRONG_VERSION

0x98130009: ENI, OEM: ENI encryption algorithm version not supported

EC_E_ENI_ENCRYPTED

0x9813000A: OEM: Loading encrypted ENI needs OEM key

EC_E_OEM_KEY_MISMATCH

0x9813000B: RAS, APP: OEM key mismatch

EC_E_OEM_KEY_MISSING

0x9813000C: APP: OEM key access needs OEM key set (e.g. Application must call esSetOemKey (HiL) or set EC_T_LINK_PARMS_SIMULATOR::qwOemKey (SiL))

EC_E_S2SMBX_NOT_CONFIGURED

0x98130020: S2S: Not Configured

EC_E_S2SMBX_NO_MEMORY

0x98130021: S2S: No Memory

EC_E_S2SMBX_NO_DESCRIPTOR

0x98130022: S2S: No Descriptor

EC_E_S2SMBX_DEST_SLAVE_NOT_FOUND

0x98130023: S2S: Destination Slave not found

EC_E_MASTER_RED_STATE_INACTIVE

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

EC_E_MASTER_RED_STATE_ACTIVE

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

EC_E_JUNCTION_RED_LINE_BREAK

0x98130026: Junction redundancy line break

EC_E_VALIDATION_ERROR

0x98130027: Validation error (validation data mismatch)

EC_E_TIMEOUT_WAITING_FOR_DC

0x98130028: Timeout waiting for DC

EC_E_TIMEOUT_WAITING_FOR_DCM

0x98130029: Timeout waiting for DCM

EC_E_SIGNATURE_MISMATCH

0x98130030: Signature mismatch

EC_E_PDIWATCHDOG

0x98130031: PDI watchdog expired

EC_E_BAD_CONNECTION

0x98130032: Bad connection

EC_E_XML_INCONSISTENT

0x98130033: ENI: Inconsistent content

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 too small

DCM_E_DRIFT_TO_HIGH

0x981201CA: DCM controller - Drift between local timer and ref clock to high

DCM_E_BUS_CYCLE_WRONG

0x981201CB: DCM controller - Bus cycle time (dwBusCycleTimeUsec) doesn't match real cycle

DCX_E_NO_EXT_CLOCK

0x981201CC: DCX controller - No external synchronization clock found

DCM_E_INVALID_DATA

0x981201CD: DCM controller - Invalid data

8.4 ADS over EtherCAT® (AoE) Error Codes

EC_E_AOE_NO_RUNTIME

0x9813000D: AoE: No Rtime

EC_E_AOE_LOCKED_MEMORY

0x9813000E: AoE: Allocation locked memory

EC_E_AOE_MAILBOX

0x9813000F: AoE: Insert mailbox error

EC_E_AOE_WRONG_HMSG

0x98130010: AoE: Wrong receive HMSG

EC_E_AOE_BAD_TASK_ID

0x98130011: AoE: Bad task ID

EC_E_AOE_NO_IO

0x98130012: AoE: No IO

EC_E_AOE_UNKNOWN_AMS_COMMAND

0x98130013: AoE: Unknown ADS command

EC_E_AOE_WIN32

0x98130014: AoE: Win 32 error

EC_E_AOE_LOW_INSTALL_LEVEL

0x98130015: AoE: Low installation level

EC_E_AOE_NO_DEBUG

0x98130016: AoE: No debug available

EC_E_AOE_AMS_SYNC_WIN32

0x98130017: AoE: Sync Win 32 error

EC_E_AOE_AMS_SYNC_TIMEOUT

0x98130018: AoE: Sync Timeout

EC_E_AOE_AMS_SYNC_AMS

0x98130019: AoE: Sync AMS error

EC_E_AOE_AMS_SYNC_NO_INDEX_MAP

0x9813001A: AoE: Sync no index map

EC_E_AOE_TCP_SEND

0x9813001B: AoE: TCP send error

EC_E_AOE_HOST_UNREACHABLE

0x9813001C: AoE: Host unreachable

EC_E_AOE_INVALIDAMSFRAGMENT

0x9813001D: AoE: Invalid AMS fragment

EC_E_AOE_NO_LOCKED_MEMORY

0x9813001E: AoE: No allocation locked memory

EC_E_AOE_MAILBOX_FULL

0x9813001F: AoE: Mailbox full

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 a hardware error (CoE abort code 0x06060000 of slave)

EC_E_SDO_ABORTCODE_DATA_LENGTH_NOT_MATCH

0x98110050: SLV: SDO: Data type does not match, length of service parameter does not match (CoE abort code 0x06070010 of slave)

EC_E_SDO_ABORTCODE_DATA_LENGTH_TOO_HIGH

0x98110051: SLV: SDO: Data type does not match, length of service parameter too high (CoE abort code 0x06070012 of slave)

EC_E_SDO_ABORTCODE_DATA_LENGTH_TOO_LOW

0x98110052: SLV: SDO: Data type does not match, length of service parameter too low (CoE abort code 0x06070013 of slave)

EC_E_SDO_ABORTCODE_OFFSET

0x98110053: SLV: SDO: Sub-index does not exist (CoE abort code 0x06090011 of slave)

EC_E_SDO_ABORTCODE_VALUE_RANGE

0x98110054: SLV: SDO: Value range of parameter exceeded (only for write access) (CoE abort code 0x06090030 of slave)

EC_E_SDO_ABORTCODE_VALUE_TOO_HIGH

0x98110055: SLV: SDO: Value of parameter written too high (CoE abort code 0x06090031 of slave)

EC_E_SDO_ABORTCODE_VALUE_TOO_LOW

0x98110056: SLV: SDO: Value of parameter written too low (CoE abort code 0x06090032 of slave)

EC_E_SDO_ABORTCODE_MINMAX

0x98110057: SLV: SDO: Maximum value is less than minimum value (CoE abort code 0x06090036 of slave)

EC_E_SDO_ABORTCODE_GENERAL

0x98110058: SLV: SDO: General error (CoE abort code 0x08000000 of slave)

EC_E_SDO_ABORTCODE_TRANSFER

0x98110059: SLV: SDO: Data cannot be transferred or stored to the application (CoE abort code 0x08000020 of slave)

EC_E_SDO_ABORTCODE_TRANSFER_LOCAL_CONTROL

0x9811005A: SLV: SDO: Data cannot be transferred or stored to the application because of local control (CoE abort code 0x08000021 of slave)

EC_E_SDO_ABORTCODE_TRANSFER_DEVICE_STATE

0x9811005B: SLV: SDO: Data cannot be transferred or stored to the application because of the present device state (CoE abort code 0x08000022 of slave)

EC_E_SDO_ABORTCODE_DICTIONARY

0x9811005C: SLV: SDO: Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of a file error) (CoE abort code 0x08000023 of slave)

EC_E_SDO_ABORTCODE_UNKNOWN

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

EC_E_SDO_ABORTCODE_MODULE_ID_LIST_NOT_MATCH

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

EC_E_SDO_ABORTCODE_SI_NOT_WRITTEN

0x98130004: SLV: SDO: Sub Index cannot be written, 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 opcode)

EC_E_SOE_ERRORCODE_ALREADY_ACTIVE

0x9811009F: ERROR SoE: Command already active

EC_E_SOE_ERRORCODE_NOT_INTERRUPT

0x98110100: ERROR SoE: Command not interruptible

EC_E_SOE_ERRORCODE_CMD_NOT_AVAIL

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

EC_E_SOE_ERRORCODE_CMD_NOT_AVAIL1

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

EC_E_SOE_ERRORCODE_DRIVE_NO

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

EC_E_SOE_ERRORCODE_IDN

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

EC_E_SOE_ERRORCODE_FRAGMENT_LOST

0x98110105: ERROR SoE: At least one fragment lost

EC_E_SOE_ERRORCODE_BUFFER_FULL

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

EC_E_SOE_ERRORCODE_NO_DATA

0x98110107: ERROR SoE: No data state

EC_E_SOE_ERRORCODE_NO_DEFAULT_VALUE

0x98110108: ERROR SoE: No default value

EC_E_SOE_ERRORCODE_DEFAULT_LONG

0x98110109: ERROR SoE: Default value transmission too long

EC_E_SOE_ERRORCODE_DEFAULT_WP

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

EC_E_SOE_ERRORCODE_INVL_DRIVE_NO

0x9811010B: ERROR SoE: Invalid drive number

EC_E_SOE_ERRORCODE_GENERAL_ERROR

0x9811010C: ERROR SoE: General error

EC_E_SOE_ERRCODE_NO_ELEM_ADR

0x9811010D: ERROR SoE: No element addressed

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_INVALIDDATARECEIVED

0x98110189: RAS: Invalid data received (communication corrupted)

EMRAS_EVT_SERVERSTOPPED

0x98110191: RAS: Server stopped (e.g. connection dropped because of Remote API Server stop)

EMRAS_EVT_WDEXPIRED

0x98110192: RAS: Watchdog expired (e.g. connection dropped because of missing keep-alive messages)

EMRAS_EVT_RECONEXPIRED

0x98110193: RAS: Reconnect expired (obsolete)

EMRAS_EVT_CLIENTLOGON

0x98110194: RAS Server: Client logged on

EMRAS_EVT_RECONNECT

0x98110195: RAS: obsolete

EMRAS_EVT SOCKCHANGE

0x98110196: RAS: Socket exchanged after reconnect (obsolete)

EMRAS_EVT_CLNTDISC

0x98110197: RAS: Client disconnect

EMRAS_E_ACCESS_NOT_FOUND

0x98110198: RAS: Access not configured for this call (e.g. SPoC access configuration missing)

EMRAS_E_TOKEN_MISSING

0x98110199: RAS: Token missing

EMRAS_E_TOKEN_INVALID

0x9811019A: RAS: Token invalid

EMRAS_E_TOKEN_DENIED

0x9811019B: RAS: Token denied

EMRAS_E_TLS_CERTIFICATE_ERROR

0x9811019C: RAS: TLS certificate error

EMRAS_E_TLS_PRIVATEKEY_ERROR

0x9811019D: RAS: TLS private key error

EMRAS_E_TLS_HANDSHAKE_FAILED

0x9811019E: RAS: TLS handshake failed