



acontis technologies GmbH

SOFTWARE

EC-Master

.NET Programming Interface

V3.3

AT3305

Edition: September 30, 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	4
1.1	Requirements	4
1.2	Architecture	5
2	Programmers Guide	6
2.1	Sample Application	6
2.2	Sample Code	6
2.3	Wrapper	6
2.3.1	Modules	6
2.3.2	Return code vs. exception handling	7
3	Application programming interface, reference	8
3.1	CEcWrapperDotNet	8
3.2	Types	102
3.3	Error Codes	155
4	Examples	179
4.1	CoeSdoUpload	179
4.2	CoeSdoDownload	179
4.3	ReadSlaveEEPROM	180
4.4	FoeFileDownload	180
5	FAQ	182

1 Introduction

The .NET Wrapper provides a .NET interface to use EC-Master, EC-Monitor, EC-Simulator and RAS Client/Server.

1.1 Requirements

Microsoft .NET Framework

- **Windows (x86/x64)**
 - Microsoft Windows 10 and above
 - Microsoft .NET Framework 4 Client Profile
 - Microsoft Visual C++ 2015 Runtime

- **Linux (x86/x64)**
 - Mono 3.2 and above (install mono-complete)

Microsoft .NET Core

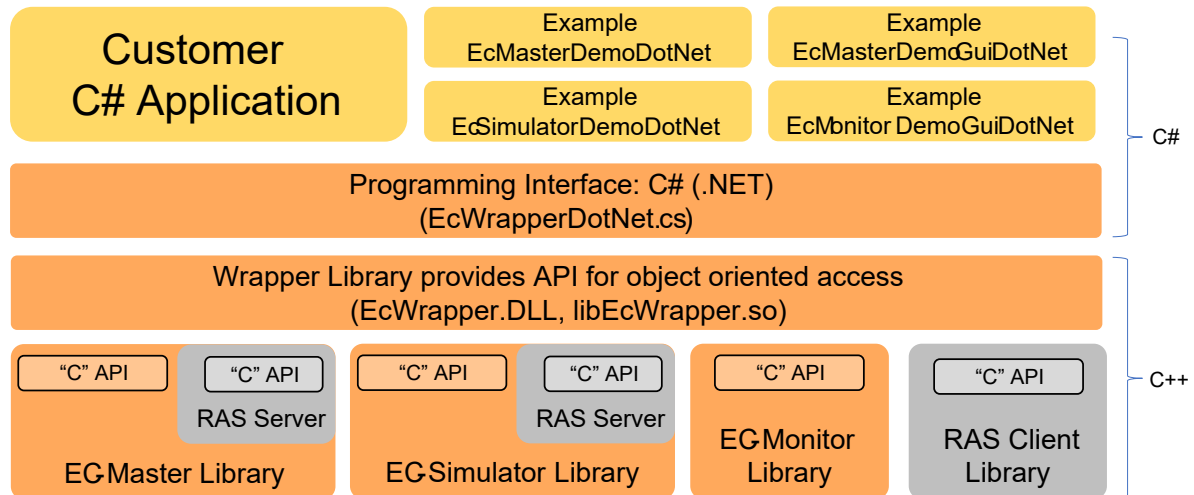
- **Windows (x86/x64)**
 - Microsoft Windows 10 and above
 - Microsoft Visual C++ 2015 Runtime

- **Linux (x64/ARM)**
 - Ubuntu 18.04 and above

- **.NET Core 3.1 Runtime**

- <https://dotnet.microsoft.com/en-us/download/dotnet>

1.2 Architecture



The architecture contains 4 basic layers:

Customer .NET application or our examples (`EcMasterDemoDotNet`, ...)

- Demo application, written in C#

Programming Interface (`EcWrapperDotNet`)

- Provides an object oriented API written in .NET

Wrapper Library (`EcWrapper`)

- Native wrapper library, which provides API for object oriented access

Native Libraries

- EC-Master Library
- EC-Monitor Library
- EC-Simulator Library
- RAS Client Library

2 Programmers Guide

2.1 Sample Application

There is currently 1 script available:

EcMasterDemoDotNet.bat
Starts the console demo application

2.2 Sample Code

The .NET demo application contains of 1 class:

EcDemoApp.cs:
Console demo application

2.3 Wrapper

2.3.1 Modules

The .NET Wrapper contains of 5 modules:

CEcWrapperDotNet.cs

```
class CEcWrapperDotNet
    EC-Wrapper base class

class CEcMasterDotNet
    EC-Master

class CEcMasterRasServerDotNet
    RAS Server for EC-Master

class CEcMasterMbxGatewayClientDotNet
    Mailbox Gateway Client for EC-Master

class CEcMasterMbxGatewayServerDotNet
    Mailbox Gateway Server for EC-Master

class CEcSimulatorDotNet
    EC-Simulator

class CEcSimulatorRasServerDotNet
    RAS Server for EC-Simulator

class CEcRasClientDotNet
    RAS Client for EC-Master / EC-Simulator
```

CEcMotionDotNet.cs

```
class CEcMotionDotNet
    EC-Motion interface
```

CEcWrapperDotNetTypes.cs
.NET types

CEcWrapper.cs
.NET C based interface (internal)

CEcWrapperTypes.cs

.NET C based types (internal)

2.3.2 Return code vs. exception handling

The most of all API functions returns a return code for error handling. This behaviour can be changed to throw an exception in error case by simply setting:

```
CEcWrapperDotNet.EnableExceptionHandling = true; // default is false
```

3 Application programming interface, reference

Function prototypes, definitions etc. of the API can be found in `CEcWrapperDotNet.cs` and `CEcWrapperDotNetTypes.cs`

3.1 CEcWrapperDotNet

class `EcWrapperDotNet.CEcWrapperDotNet` : `IDisposable`

Subclassed by `EcWrapperDotNet.CEcMasterMbxGatewayClientDotNet`, `EcWrapperDotNet.CEcMasterMbxGatewayServerDotNet`, `EcWrapperDotNet.CEcMasterRasServerDotNet`, `EcWrapperDotNet.CEcMonitorDotNet`, `EcWrapperDotNet.CEcMonitorRasServerDotNet`, `EcWrapperDotNet.CEcRasClientDotNet`, `EcWrapperDotNet.CEcSimulatorDotNet`, `EcWrapperDotNet.CEcSimulatorRasServerDotNet`, `EcWrapperDotNet.CEcWrapperDotNetEx`

Public Functions

void `Dispose` ()

summary> Throws the EtherCAT-Notifications. The Type of the event depends on the notification code.

ECError `SetCycFrameRxCallback` (`CycFrameRxCallback pfnCallback`)

Set cyclic frame RX callback function.

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.

Param `pfnCallback`

Callback function

Return

`EC_E_NOERROR` or error code

`EcRunMode` `GetRunMode` ()

Return run mode.

Return

Run mode

ECError `RegisterClient` (`out DN_EC_T_REGISTERRESULTS pRegRes`)

Register a client with the EtherCAT Master.

Param `pRegRes`

out Registration results, a pointer to a structure of type `REGISTERRESULTS`

Return

Error Code

ECError `UnregisterClient` ()

Unregister a client from the EtherCAT master.

Return

Error code

ECError `DcmGetLog` (`out string pszLog`)

Get logging information from the DCM controller.

Return

Error code

ECError **RasGetConnectionInfo** (out *DN_EC_T_RAS_CONNECTION_INFO* pConInfo)

Get actual connection information /summary>

param pConInfo

connection information

return

EC_E_NOERROR on success, otherwise an error code.

ECError **MbxGatewayCoeSdoDownload** (

ushort wAddress,

ushort wObIndex,

byte byObSubIndex,

byte[] pbyData,

uint dwDataLen,

uint dwTimeout,

uint dwFlags

)

Initiate CoE SDO download transfer to mailbox gateway server.

Param wAddress

slave address to download the SDO

Param wObIndex

object index

Param byObSubIndex

object sub-index

Param pbyData

Param dwDataLen

Param dwTimeout

Timeout in milliseconds

Param dwFlags

mailbox transfer flags

Return

Depends on the implementation of the function OsSetLastError(), normally returns EC_E_NOERROR if it succeeds or error code if it fails

ECError **MbxGatewayCoeSdoUpload** (

ushort wAddress,

ushort wObIndex,

byte byObSubIndex,

byte[] pbyData,

uint dwDataLen,

out uint pdwOutDataLen,

uint dwTimeout,

uint dwFlags

)

Initiate CoE SDO upload transfer from mailbox gateway server.

Param wAddress

slave address to upload the SDO

Param wObIndex

object index

Param byObSubIndex

object sub-index

Param pbyData

Param dwDataLen

Param pdwOutDataLen

Param dwTimeout

Timeout in milliseconds

Param dwFlags

mailbox transfer flags, see EC_MAILBOX_FLAG

Return

Depends on the implementation of the function OsSetLastError(), normally returns EC_E_NOERROR if it succeeds or error code if it fails

void **EnablePerformanceMeasuring** (bool *bEnable*)

Enables performance monitoring. For every master function, the notification “PerfNotification” will be called.

Param bEnable

True = Enables performance monitoring

void **EnableTranslation** (bool *bEnable*)

Enables translation. For every master string, the notification “TranslateNotification” will be called.

Param bEnable

True = Enables translation

ECError **OptimizeProcessPerformance** ()

Optimize process performance.

Return

EC_E_NOERROR on success, otherwise an error code.

String **GetErrorText** (*ECError eErrorCode*)

Return text tokens by Error code from master stack.

Param eErrorCode

Error code

Return

Error text for supplied error code

ECError **ConfigureNetwork** (FileInfo *oConfigFile*)

Configure the Master

param oConfigFile

File with configuration parameters

Return

Error code

ECError **ConfigureNetwork** (*DN_EC_T_CNF_TYPE tConfigType*, Stream *oConfigData*)

Configure the Master

param tConfigType

Enum type of configuration data provided

param oConfigData

Configuration data (stream)

Return

Error code

ECError **ExecJob** (*DN_EC_T_USER_JOB eUserJob*)

Execute or initiate the requested master job.

Param eUserJob

Requested job.

Return

Error code

ECError **ExecJobProcessAllRxFrames** (*out bool bPrevCycProcessed*)

calls the process all rx frame job

Param bPrevCycProcessed

true: previous send frame was received and processed, false: otherwise

Return

EC_NOERROR or error code

ECError **ExecJobProcessAllRxFrames** ()

calls the process all rx frame job

Return

EC_NOERROR or error code

ECError **ExecJobSendCycFramesByTaskId** (*UInt32 dwTaskId*)

sends a cycle frame by its task id

Param dwTaskId

task id of the cyclic frame

Return

EC_NOERROR or error code

string **ConvertMasterStringFromBytes** (*byte[] abyData, int dwOffset, int dwLen*)

Converts a master string into UTF8 format.

Param abyData

Byte array to convert

Param dwOffset

Offset

Param dwLen

Length

Return

Converted string

string **ConvertMasterString** (*String val*)

Converts a master string into UTF8 format.

Param val

String to convert.

Return

Converted string.

bool **WaitForTimingTaskEvent** ()

Waits for timing task event.

Return

Wait event was triggered

ECError **EoeInstallEndpoint** (

bool *bCreateTxEvent*,

String *strInterfaceName*,

String *strInterfaceGuid*

)

Install EoE endpoint.

Param bCreateTxEvent

Create TX Event

Param strInterfaceName

Interface name (optional)

Param strInterfaceGuid

Interface guid (optional)

Return

Error code

ECError **EoeUninstallEndpoint** ()

Uninstall EoE endpoint.

Return

Error code

ECError **EoeTriggerTxEvent** ()

Trigger TX Event, which must be called from JobTask after ExecJobProcessAllRxFrames()

Return

Error code

string **ESCTypeText** (byte *byEscType*, bool *bLong*)

Gets the text of corresponding ESC type.

Param byEscType

ESC type

Param bLong

true: long text version, false: short text version

Return

Text of corresponding ESC type

string **SlaveVendorText** (uint *dwVendorId*)

Gets the text of slave vendor.

Param dwVendorId

Vendor ID

Return

Text of slave vendor

string **SlaveProdCodeText** (uint *dwVendorId*, uint *dwProductCode*)

Gets the text of slave product code.

Param dwVendorId

Vendor ID

Param dwProductCode

Product code

Return

Text of slave product code

ECError **SetBusCnfReadProp** (*DN_ESC_SII_REG* *eEscSiiReg*, uint *dwTimeout*)

Sets read property for bus configuration.

Param eEscSiiReg

SII register

Param dwTimeout

Time out of bus scan

Return

Error code

ECError **RestartScanBus** (uint *dwTimeout*, bool *bReadRevisionNo*, bool *bReadSerialNo*)

Trigger Bus Scan.

Param dwTimeout

Time out of bus scan

Param bReadRevisionNo

Read revision number

Param bReadSerialNo

Read serial number

Return

Error code

ECError **GetBusScanSlaveInfoDesc** (
short *wAutoIncAddr*,
out *DN_EC_T_SB_SLAVEINFO_DESC* *oSlaveInfoDesc*
)

This call will return the basic slave info determined in the last bus scan.

Param wAutoIncAddr

Auto increment address of the slave

Param oSlaveInfoDesc

Out parameter that contains different slave information after the call

Return

Error code

ECError **GetScanBusStatus** (out *DN_EC_T_SB_STATUS_NOTIFY_DESC* *oSbStatus*)

Gets the status of the last bus scan.

Param oSbStatus

The last bus scan status

Return

Error code

ECError **CoeGetODList** (
UInt32 *dwSlaveId*,
DN_EC_T_COE_ODLIST_TYPE *eListType*,
out UInt16[] *pList*,
UInt32 *dwLen*,
out UInt32 *pdwOutLen*,
UInt32 *dwTimeout*
)

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

Param dwSlaveId

slave ID

Param eListType

Object list type. eODListType_ALL: Get all objects eODListType_RxPdoMap: Get PDO mappable objects eODListType_TxPdoMap: Get objects that can be changed eODListType_StoredFRepl: Get objects that are stored for a device replacement eODListType_StartupParm: Get objects that can be used as startup parameter

Param pList

out Filled OD list

Param dwLen

Length of list

Param pdwOutLen

out Length of output list

Param dwTimeout

Timeout

Return

Error code

```

ECError CoeGetObjectDesc (
    UInt32 dwSlaveId,
    UInt16 wIndex,
    out DN_EC_T_COE_OBDESC oObjDesc,
    UInt32 dwTimeout
)

```

Determines the description of a specific object.

Param dwSlaveId

slave ID

Param wIndex

Object index

Param oObjDesc

out Description of a specific object

Param dwTimeout

Timeout

Return

Error code

```

ECError CoeGetEntryDesc (
    UInt32 dwSlaveId,
    UInt16 wIndex,
    Byte bySubIndex,
    Byte byValueInfoType,
    out DN_EC_T_COE_ENTRYDESC oEntryDesc,
    UInt32 dwTimeout
)

```

Determines the description of a specific object entry.

Param dwSlaveId

slave ID

Param wIndex

Object index

Param bySubIndex

Object subindex

Param byValueInfoType

Bit mask to define which information to determine. The bit values are defined as follows:
 EC_COE_ENTRY_ObjAccess - Object access rights EC_COE_ENTRY_ObjCategory - Object category
 EC_COE_ENTRY_PdoMapping - Information if the object is PDO map-pable EC_COE_ENTRY_UnitType - Unit
 EC_COE_ENTRY_DefaultValue - Default value EC_COE_ENTRY_MinValue - Minimum value EC_COE_ENTRY_MaxValue - Maximum value

Param oEntryDesc

out Description of a specific object entry

Param dwTimeout

Timeout

Return

Error code

ECError **MbxTferCreate** (UInt32 *dwTferId*, UInt32 *dwBufferSize*, **out** IntPtr *pMbxTfer*)

Creates a mailbox transfer object.

Param dwTferId

transfer ID (optional, can be 0)

Param dwBufferSize

buffer size

Param pMbxTfer

out mailbox transfer object

Return

Error code

ECError **MbxTferReset** (IntPtr *pMbxTfer*)

Creates a mailbox transfer object.

Param pMbxTfer

mailbox transfer object

Return

Error code

void **MbxTferDelete** (IntPtr *pMbxTfer*)

Deletes a mailbox transfer object.

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

Param pMbxTfer

Mailbox transfer object created with emMbxTferCreate

ECError **MbxTferWait** (IntPtr *pMbxTfer*)

Waits until mailbox transfer is finished.

Param pMbxTfer

mailbox transfer object

Return

Error code

ECError **MbxTferCopyTo** (IntPtr *pMbxTfer*, byte[] *abyData*, UInt32 *dwDataLen*)

Copies data to the mailbox transfer buffer.

Param pMbxTfer

pMbxTfer

Param abyData

abyData

Param dwDataLen

dwDataLen

Return

Error code

ECError **MbxTferCopyFrom** (
 IntPtr *pMbxTfer*,
 byte[] *abyData*,
 UInt32 *dwDataLen*,
out UInt32 *pdwOutDataLen*
)

Copies data from the mailbox transfer buffer.

Param pMbxTfer
pMbxTfer

Param abyData
abyData

Param dwDataLen
dwDataLen

Param pdwOutDataLen
pdwOutDataLen

Return
Error code

```
ECError NotifyApp (
    UInt32 dwCode,
    byte[] pInData,
    UInt16 wDataLen,
    out byte[] pOutData,
    UInt16 wOutLen,
    out UInt32 pdwOutDataLen
)
```

Notifies the master application.

Param dwCode
Code

Param pInData
Input data

Param wDataLen
Length of in data

Param pOutData
Output data

Param wOutLen
Length of output data field

Param pdwOutDataLen
Length of actual out data

Return
Error code

```
ECError InitMotion ()
Initialize the motion.
```

Return
Error code

```
ECError DeinitMotion ()
Deinitialize the motion.
```

Return
Error code

```
ECError ControlMotion (
    EC_T_MOTION_CMD dwCmd,
    byte[] pInData,
    UInt16 wDataLen,
    out byte[] pOutData,
    UInt16 wOutLen
)
```

Controls the motion.

Param dwCmd
Command

Param pObj
Object

Param pInData
Input data

Param wDataLen
Length of in data

Param pOutData
Output data

Param wOutLen
Length of output data field

Return
Error code

void **ReadIdentifyObj** (ushort *wFixedAddr*)
Reads the identify object of a slave.

Param wFixedAddr
Fixed station address

void **AddDbgMsg** (string *strDbgMsg*)
Adds a debug message to the print message queue.

Param strDbgMsg
debug message

```
ECError GetSlaveInfoEx (
    DN_EC_T_SB_SLAVEINFO_REQ_DESC oReq,
    out DN_EC_T_SB_SLAVEINFO_RES_DESC oRes
)
```

Gets the extended slave info determined in the last bus scan.

Param oReq
Request parameter

Param oRes
The extended slave information structure

Return
Error code

```
ECError GetMasterParams (out DN_EC_T_INIT_MASTER_PARAMS pParams)
```

Get current Master initialization parameters.

If the given buffer is larger than the actual size of structure EC_T_INIT_MASTER_PARAMS, the parameters of EC_T_INIT_MASTER_PARAMS.pOsParams, EC_T_INIT_MASTER_PARAMS.pLinkParams and EC_T_INIT_MASTER_PARAMS.pLinkParamsRed are appended.

Param pParms

Buffer to store Master parameters

Return

- EC_E_NOERROR on success
- EC_E_INVALIDSTATE if master isn't initialized
- EC_E_INVALIDPARAM if buffer pParms is too small

ECError **SetMasterParms** (*DN_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

Param pParms

New Master parameters

Return

- EC_E_NOERROR on success
- EC_E_INVALIDSTATE if master isn't initialized

ECError **ForceSlvStatCollection** ()

Sends datagrams to collect slave statistics counters.

Return

Error code

ECError **GetSlvStatistics** (
 UInt32 dwSlaveId,
 out *DN_EC_T_SLVSTATISTICS_DESC* oStatistics
)

Returns slave statistics counters.

Param dwSlaveId

Slave ID

Param oStatistics

out Statistics counters

Return

Error code

ECError **GetCyclicConfigInfo** (out *DN_EC_T_CYC_CONFIG_DESC*[] oCyclicConfigInfo)

Returns an array of cyclic tasks.

Param oCyclicConfigInfo

out Array of cyclic tasks

Return

Error code

ECError SetAllSlavesMustReachState (bool *bAllSlavesMustReachState*)

Sets flag that all slaves must reach the requested master state.

Param bAllSlavesMustReachState

EC_TRUE: All slaves must reach the requested master state

Return

Error code

ECError SetAdditionalVariablesForSpecificDataTypesEnabled (
bool *bAdditionalVariablesForSpecificDataTypes*
)

Sets flag to enable additional variables for specific data types.

Param bAdditionalVariablesForSpecificDataTypes

EC_TRUE: Enable additional variables for specific data types

Return

Error code

ECError EnableNotification (UInt32 *dwClientId*, *DN_NotifyCode* *tNotifyCode*, bool *bEnable*)

Enables notification.

Param dwClientId

Client Id

Param tNotifyCode

Code of notification, which should be enabled

Param bEnable

EC_TRUE: Enables notification

Return

Error code

ECError ThrottleNotification (*DN_NotifyCode* *tNotifyCode*, uint *dwTimeout*)

Throttles notification.

Param tNotifyCode

Code of notification, which should be throttled

Param dwTimeout

0 = Not throttled, > 0 = Throttle timeout in ms

Return

Error code

ECError SdoUploadMasterOd (UInt16 *wObIndex*, UInt32 *dwTimeout*, out object *pobjMasterOd*)

SdoUpload from Master Object Dictionary.

Param wObIndex

Object index

Param dwTimeout

Timeout in milliseconds

Param pobjMasterOd

Read object (e.g. DN_EC_T_OBJ3XXX, EC_T_OBJ8XXX, EC_T_OBJ9XXX ...)

Return

Error code

```

ECError ReadSlaveEEPROM (
    bool bFixedAddressing,
    UInt16 wSlaveAddress,
    UInt16 wEEPROMStartOffset,
    byte[] pwReadData,
    UInt32 dwReadLen,
    out UInt32 pdwNumOutData,
    UInt32 dwTimeout
)

```

Read EEPROM data from slave.

Param bFixedAddressing

EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing

Param wSlaveAddress

Slave Address, station or auto increment address depending on bFixedAddressing

Param wEEPROMStartOffset

Address to start EEPROM Read from

Param pwReadData

Pointer to ushort array to carry the read data

Param dwReadLen

Size of the ushort array provided at pwReadData (in ushorts)

Param pdwNumOutData

out Pointer to uint carrying actually read data (in ushorts) after completion

Param dwTimeout

Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT

Return

Error code

```

ECError ReadSlaveEEPROMWithReq (
    uint dwClientId,
    uint dwTferId,
    bool bFixedAddressing,
    UInt16 wSlaveAddress,
    UInt16 wEEPROMStartOffset,
    byte[] pwReadData,
    UInt32 dwReadLen,
    out UInt32 pdwNumOutData,
    UInt32 dwTimeout
)

```

Read EEPROM data from slave (REQ)

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param dwTferId

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.

Param bFixedAddressing

EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing

Param wSlaveAddress

Slave Address, station or auto increment address depending on bFixedAddressing

Param wEEPROMStartOffset

Address to start EEPROM Read from

Param pwReadData

Pointer to ushort array to carry the read data

Param dwReadLen

Size of the ushort array provided at pwReadData (in ushorts)

Param pdwNumOutData

out Pointer to uint carrying actually read data (in ushorts) after completion

Param dwTimeout

Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT

Return

Error code

```
ECError WriteSlaveEEPROM (
    bool bFixedAddressing,
    UInt16 wSlaveAddress,
    UInt16 wEEPROMStartOffset,
    byte[] pwWriteData,
    UInt32 dwWriteLen,
    UInt32 dwTimeout
)
```

Write EEPROM data from slave.

Param bFixedAddressing

EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing

Param wSlaveAddress

Slave Address, station or auto increment address depending on bFixedAddressing

Param wEEPROMStartOffset

Address to start EEPROM Write from

Param pwWriteData

Pointer to WORD array carrying the write data.

Param dwWriteLen

Sizeof Write Data WORD array (in WORDS)

Param dwTimeout

Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT

Return

Error code

```
ECError WriteSlaveEEPROMWithReq (
    uint dwClientId,
    uint dwTferId,
    bool bFixedAddressing,
    UInt16 wSlaveAddress,
    UInt16 wEEPROMStartOffset,
    byte[] pwWriteData,
    UInt32 dwWriteLen,
    UInt32 dwTimeout
)
```

Write EEPROM data from slave (REQ)

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param dwTferId

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_NTIFY_DESC.

Param bFixedAddressing

EC_TRUE: use station addressing, EC_FALSE: use auto increment addressing

Param wSlaveAddress

Slave Address, station or auto increment address depending on bFixedAddressing

Param wEEPROMStartOffset

Address to start EEPROM Write from

Param pwWriteData

Pointer to WORD array carrying the write data.

Param dwWriteLen

Sizeof Write Data WORD array (in WORDS)

Param dwTimeout

Timeout in milliseconds. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT

Return

Error code

ECError **InitInstance** (*DN_EC_T_INIT_PARMS* oParms)

Initializes EtherCAT wrapper.

Param oParms

Parameters

Return

Error code

ECError **DeinitInstance** ()

Deinitialize EtherCAT wrapper.

Return

Error code

ECError **PerfMeasAppCreate** (
DN_EC_T_PERF_MEAS_APP_PARMS pPerfMeasAppParms,
out IntPtr ppvPerfMeas
)

Create PerfMeas object and bind it to the master instance.

This API can be called multiple times to create PerfMeas objects. The performance counters in each of the objects can be accessed in the following two ways:

- by passing the PerfMeas object and the index of the performance measurement. The index ranges from [0-pPerfMeasAppParms->dwNumAppMeas]
- by passing EC_NULL instead of a PerfMeas object and an index. In this case the index works across all PerfMeas objects bound to the master instance.

Param pPerfMeasAppParms

Pointer to parameter definitions

Param ppvPerfMeas

Created PerfMeas object.

Return

EC_E_NOERROR or an error code

ECError **PerfMeasAppDelete** (IntPtr *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

Param *pvPerfMeas*

PerfMeas object to delete

Return

EC_E_NOERROR or an error code

ECError **PerfMeasAppStart** (IntPtr *pvPerfMeas*, uint *dwIndex*)

Start application performance measurement.

Param *pvPerfMeas*

PerfMeas object or EC_NULL to use continuous index

Param *dwIndex*

Index of the performance measurement

Return

EC_E_NOERROR or an error code

ECError **PerfMeasAppEnd** (IntPtr *pvPerfMeas*, uint *dwIndex*)

Stop application performance measurement.

Param *pvPerfMeas*

PerfMeas object or EC_NULL to use continuous index

Param *dwIndex*

Index of the performance measurement

Return

EC_E_NOERROR or an error code

ECError **PerfMeasAppReset** (IntPtr *pvPerfMeas*, uint *dwIndex*)

Reset application performance measurement.

Param *pvPerfMeas*

PerfMeas object or EC_NULL to use continuous index

Param *dwIndex*

Index of the performance measurement, use 0xFFFFFFFF to reset all

Return

EC_E_NOERROR or an error code

ECError **PerfMeasAppGetRaw** (
 IntPtr *pvPerfMeas*,
 uint *dwIndex*,
 DN_EC_T_PERF_MEAS_VAL[] *pPerfMeasVal*,
 DN_EC_T_PERF_MEAS_HISTOGRAM[] *pPerfMeasHistogram*,
 uint *dwPerfMeasNumOf*
)

Get raw data of one/all application performance measurement.

Param *pvPerfMeas*

PerfMeas object or EC_NULL to use continuous index

Param dwIndex

Index of the performance measurement, use 0xFFFFFFFF to get all

Param pPerfMeasVal

Pointer to a buffer receiving one/all performance measurement values or EC_NULL

Param pPerfMeasHistogram

Pointer to a buffer receiving one/all performance measurement histograms or EC_NULL

Param dwPerfMeasNumOf

Number of elements allocated in pPerfMeasVal and pPerfMeasHistogram

Return

EC_E_NOERROR or an error code

```
ECError PerfMeasAppGetInfo (
    IntPtr pvPerfMeas,
    uint dwIndex,
    out DN_EC_T_PERF_MEAS_INFO[] pPerfMeasInfo,
    uint dwPerfMeasNumOf
)
```

Get general info about one/all application performance measurement.

Param pvPerfMeas

PerfMeas object or EC_NULL to use continuous index

Param dwIndex

Index of the performance measurement information, use 0xFFFFFFFF to get all

Param pPerfMeasInfo

Pointer to a buffer receiving one/all performance measurement information

Param dwPerfMeasNumOf

Number of elements allocated in pPerfMeasInfo

Return

EC_E_NOERROR or an error code

```
ECError PerfMeasAppGetNumOf (IntPtr pvPerfMeas, out uint pdwNumOf)
```

Reset number of application performance measurement.

Param pvPerfMeas

PerfMeas object or EC_NULL to get the number of performance measurements in all PerfMeas objects

Param pdwNumOf

Number of performance measurements

Return

EC_E_NOERROR or an error code

```
ECError PerfMeasResetByTaskId (uint dwTaskId, uint dwIndex)
```

Reset internal performance measurement.

Param dwTaskId

Task Job ID

Param dwIndex

Index of the performance measurement, use 0xFFFFFFFF to reset all

Return

EC_E_NOERROR or an error code

```

ECError PerfMeasGetRawByTaskId (
    uint dwTaskId,
    uint dwIndex,
    DN_EC_T_PERF_MEAS_VAL[] pPerfMeasVal,
    DN_EC_T_PERF_MEAS_HISTOGRAM[] pPerfMeasHistogram,
    uint dwPerfMeasNumOf
)

```

Get raw data of one/all application performance measurement.

Param *dwTaskId*

Task Job ID

Param *dwIndex*

Index of the performance measurement, use 0xFFFFFFFF to get all

Param *pPerfMeasVal*

Pointer to a buffer receiving one/all performance measurement values or EC_NULL

Param *pPerfMeasHistogram*

Pointer to a buffer receiving one/all performance measurement histograms or EC_NULL

Param *dwPerfMeasNumOf*

Number of elements allocated in *pPerfMeasVal* and *pPerfMeasHistogram*

Return

EC_E_NOERROR or an error code

```

ECError PerfMeasGetInfoByTaskId (
    uint dwTaskId,
    uint dwIndex,
    out DN_EC_T_PERF_MEAS_INFO[] pPerfMeasInfo,
    uint dwPerfMeasNumOf
)

```

Get general info about one/all internal performance measurement.

Param *dwTaskId*

Task Job ID

Param *dwIndex*

Index of the performance measurement, use 0xFFFFFFFF to get all

Param *pPerfMeasInfo*

Pointer to a buffer receiving one/all performance measurement infos

Param *dwPerfMeasNumOf*

Number of elements allocated in *pPerfMeasInfo*

Return

EC_E_NOERROR or an error code

```

ECError PerfMeasGetNumOfByTaskId (uint dwTaskId, out uint pdwNumOf)

```

Reset number of internal performance measurement.

Param *dwTaskId*

Task Job ID

Param *pdwNumOf*

Number of performance measurements

Return

EC_E_NOERROR or an error code

```

ECError SetLogParms (DN_EC_T_LOG_PARMS pLogParms)

```

Sets log parameters. Used to change the parameters provided by `emInitMaster()`.

EC_MONITOR Sets log parameters. Used to change the parameters provided by `emonInitMonitor()`.

EC_SIMULATOR Sets log parameters. Used to change the parameters provided by `esInitSimulator()`.

Param pLogParms

New Log parameters

Return

ECError **SetMasterRedStateReq** (bool *bActive*)

Requests Master Redundancy State ACTIVE / INACTIVE.

Param bActive

Return

- EC_E_NOERROR if successful
- EC_E_INVALIDSTATE if EtherCAT stack isn't initialized or MasterRedParms.bEnabled = EC_FALSE
- EC_E_INVALIDPARAM if dwInstanceID is out of range
- EC_E_NOTSUPPORTED if EC-Master stack does not include Master Redundancy support

ECError **GetMasterRedState** (out bool *pbActive*)

Gets Master Redundancy State (ACTIVE / INACTIVE).

Param pbActive

Pointer to variable of type EC_T_BOOL

Return

- EC_E_NOERROR if successful
- EC_E_INVALIDSTATE if EtherCAT stack isn't initialized or MasterRedParms.bEnabled = EC_FALSE
- EC_E_INVALIDPARAM if dwInstanceID is out of range
- EC_E_NOTSUPPORTED if EC-Master stack does not include Master Redundancy support

byte[] **GetMasterRedProcessImageInputPtr** ()

Gets the Master Redundancy process data input image pointer.

Return

Master Redundancy process data input image pointer

byte[] **GetMasterRedProcessImageOutputPtr** ()

Gets the Master Redundancy process data output image pointer.

Return

Master Redundancy process data output image pointer

ECError **IoCtl** (
 uint *dwCode*,
 byte[] *pbInBuf*,
 uint *dwInBufSize*,
 byte[] *pbOutBuf*,
 uint *dwOutBufSize*,
 out uint *pdwNumOutData*
)

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

Param dwCode

IOCTL code (EC_IOCTL...)

Param pbyInBuf

IOCTL input parameters

Param dwInBufSize

Size of IOCTL input parameters in bytes

Param pbyOutBuf

Buffer for IOCTL output

Param dwOutBufSize

Size of buffer at pbyOutBuf in bytes

Param pdwNumOutData

Amount of bytes written to pbyOutBuf by IOCTL. EC_NULL: amount not set by IOCTL.

Return

- 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

ECError **ScanBus** (uint 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.

Param dwTimeout

Timeout [ms]

Return

- 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_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

ECError **RescueScan** (uint *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.

Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

ECError **GetMasterInfo** (out *DN_EC_T_MASTER_INFO* *pMasterInfo*)

Get generic information about the Master.

Param pMasterInfo

Master information

Return

- 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

ECError **ConfigureNetwork** (
DN_EC_T_CNF_TYPE *eCnfType*,
 byte[] *pbyCnfData*,
 uint *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.

A client must not be registered prior to calling this function. Existing client registrations will be dropped.

Param eCnfType

Type of configuration data provided

Param pbyCnfData

Filename / configuration data, or EC_NULL if eCnfType is

Param dwCnfDataLen

Length of configuration data in byte, or zero if eCnfType is

Return

- EC_E_NOERROR if successful
- EC_E_INVALIDSTATE if EtherCAT stack isn't initialized or eCnfType is eCnfType_GenPreopen or eCnfType_GenOpen and link is disconnected

- **EC_E_INVALIDPARM** if dwInstanceID is out of range or pParms is EC_NULL contains some values out of range
- **EC_E_LINK_DISCONNECTED** if link is disconnected
- **EC_E_FEATURE_DISABLED** if a configured feature is not included in the license key
- **EC_E_NOTSUPPORTED** if a configured feature is not supported (e.g not compiled in the library)
- **EC_E_CFGFILENOTFOUND** if the ENI file cannot be found
- **EC_E_WRONG_FORMAT** if 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_SIZE_MISMATCH** 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

ECError ConfigLoad (*DN_EC_T_CNF_TYPE* eCnfType, byte[] pbyCnfData, uint dwCnfDataLen)

Load the network configuration.

In combination with emConfigApply, this function replaces emConfigureNetwork and must be called after the initialization. Among others the EtherCAT topology defined in the given XML configuration file will be stored internally.

A client must not be registered prior to calling this function. Existing client registrations will be dropped.

Param eCnfType

Type of configuration data provided

Param pbyCnfData

Configuration data

Param dwCnfDataLen

Length of configuration data in byte

Return

- **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
- EC_E_NOMEMORY if some memory cannot be allocated
- EC_E_ADS_IS_RUNNING if the ADS server is running

ECError ConfigExcludeSlave (ushort *wStationAddress*)

Exclude a slave from the network configuration.

It has to be called after emConfigLoad and prior to calling emConfigApply.

Param wStationAddress

Station address of the slave to be excluded. A value of 0 excludes all slaves.

Return

EC_E_NOERROR or error code

ECError ConfigIncludeSlave (ushort *wStationAddress*)

Include a slave in the network configuration.

Slaves that were previously excluded with emConfigSlaveExclude can be added again. It has to be called after emConfigLoad and prior to calling emConfigApply.

Param wStationAddress

Station address of the slave to be included. A value of 0 includes all slaves.

Return

EC_E_NOERROR or error code

ECError ConfigSetPreviousPort (

ushort *wStationAddress*,
ushort *wStationAddressPrev*,
ushort *wPortPrev*

)

Set previous port information of a slave.

It has to be called after emConfigLoad and prior to calling emConfigApply.

Param wStationAddress

Station address of the slave

Param wStationAddressPrev

Previous slave station address

Param wPortPrev

Previous port

Return

EC_E_NOERROR or error code

ECError ConfigAddJunctionRedundancyConnection (

ushort *wHeadStationAddress*,
ushort *wHeadRedPort*,
ushort *wTailStationAddress*,
ushort *wTailRedPort*

)

Add a junction redundancy connection.

Since there is no mechanism to configure junction redundancy in the ENI, this API allows adding junction redundancy connections which will be validated by the EC-Master stack. It has to be called after emConfigLoad and prior to calling emConfigApply. Calling this API enables junction redundancy support implicitly.

Param wHeadStationAddress

Station address of the junction redundancy head slave. Typically this is an EtherCAT junction.

Param wHeadRedPort

Port at head slave to which the junction redundancy cable is connected. Must be ESC_PORT_B.

Param wTailStationAddress

Station address of the junction redundancy tail slave

Param wTailRedPort

Port at tail slave to which the junction redundancy cable is connected. May not be ESC_PORT_A.

Return

EC_E_NOERROR or error code

ECError **ConfigAddCableRedundancyConnection** (ushort wStationAddress, ushort wRedPort)

Add a cable redundancy connection.

This API defines where the redundancy cable is plugged in, in which slave and at which port. It has to be called after emConfigLoad and prior to calling emConfigApply.

Param wStationAddress

Station address of the cable redundancy slave

Param wRedPort

Slave port to which the junction redundancy cable is connected.

Return

EC_E_NOERROR or error code

ECError **ConfigApply** ()

Apply the network configuration.

It has to be called after emConfigLoad.

Return

EC_E_NOERROR or error code

ECError **ConfigExtend** (bool bResetConfig, uint 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.

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

Param bResetConfig

EC_TRUE: Extended configuration will be removed

Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

ECError **IsConfigured** (out bool pbIsConfigured)

Returns if configuration has been applied.

Param pbIsConfigured

EC_TRUE if configuration has been applied

Return

EC_E_NOERROR or error code

ECError **SetMasterState** (uint *dwTimeout*, *DN_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.

Param dwTimeout

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.

Param eReqState

Requested System state

Return

- 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

ECError **SetMasterStateReq** (uint *dwTimeout*, *DN_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.

Param dwTimeout

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.

Param eReqState

Requested System state

Return

- 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

DN_EC_T_STATE GetMasterState ()

Get the EtherCAT master current state.

Return

EtherCAT master state

ECError GetMasterStateEx (out ushort pwCurrState, out ushort 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

Param pwCurrState

Current master state

Param pwReqState

Requested master state

Return

- 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

uint GetSlaveId (ushort wStationAddress)

Determines the slave ID using the slave station address.

Param wStationAddress

Station address of the slave

Return

Slave ID or INVALID_SLAVE_ID if the slave could not be found or the stack is not initialized

ECError GetSlaveFixedAddr (uint *dwSlaveId*, out ushort *pwFixedAddr*)

Determine slave station address according to its slave ID.

Param dwSlaveId

Slave ID

Param pwFixedAddr

Corresponding fixed address

Return

- 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 dwSlaveId can be found

uint **GetSlaveIdAtPosition** (ushort *wAutoIncAddress*)

Determines the slave ID using the slave auto increment address.

Param wAutoIncAddress

Auto increment address of the slave

Return

Slave ID or INVALID_SLAVE_ID if no slave matching wAutoIncAddress can be found

bool **GetSlaveProp** (uint *dwSlaveId*, out *DN_EC_T_SLAVE_PROP* *pSlaveProp*)

Determines the properties of the slave device.

Param dwSlaveId

Slave ID

Param pSlaveProp

Slave properties

Return

EC_TRUE if the slave exists, EC_FALSE if no slave matching dwSlaveId can be found

ECError GetSlavePortState (uint *dwSlaveId*, out ushort *pwPortState*)

Returns the state of the slave ports.

Param dwSlaveId

Slave ID

Param pwPortState

Slave port state.Format: wwwwww xxxx yyyy zzzz (each nibble : port 3210)

Return

- 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 dwSlaveId can be found

ECError GetSlaveState (uint *dwSlaveId*, out ushort *pwCurrDevState*, out ushort *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.

Param dwSlaveId

Slave ID

Param pwCurrDevState

Current slave state

Param pwReqDevState

Requested slave state

Return

- 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
- EC_E_SLAVE_NOT_PRESENT if the slave is not present
- EC_E_NOTFOUND if no slave matching dwSlaveId can be found

ECError **SetSlaveState** (uint dwSlaveId, ushort wDeviceState, uint 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).

Param dwSlaveId

Slave ID

Param wDeviceState

Requested device state. See DEVICE_STATES

Param dwTimeout

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.

Return

- 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

ECError **SetSlaveStateReq** (uint *dwSlaveId*, ushort *wDeviceState*, uint *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`.

Param *dwSlaveId*

Slave ID

Param *wDeviceState*

Requested device state. See `DEVICE_STATES`

Param *dwTimeout*

Timeout [ms]

Return

- `EC_E_NOERROR` if successful
- `EC_E_INVALIDSTATE` if EtherCAT stack isn't initialized or denies the requested state, see comments below
- `EC_E_INVALIDPARM` if `dwInstanceId` is out of range or `BOOTSTRAP` was requested for a slave that does not support it
- `EC_E_NOTFOUND` if no slave matching `dwSlaveId` can be found
- `EC_E_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

ECError **GetProcessVarInfoNumOf** (
DN_EC_T_VAR_DIRECTION *eVarDirection*,
DN_EC_T_VAR_SOURCE *eVarSource*,
 bool *bFixedAddress*,
 ushort *wSlaveAddress*,
 out uint *pdwProcessVarInfoNumOf*
)

Get process variables information.

Param *eVarDirection*

INPUTs, OUTPUTs, ... See `EC_T_VAR_DIRECTION` .

Param *eVarSource*

Slave, Master, ... See `EC_T_VAR_SOURCE` .

Param *bFixedAddress*

Use station address if `EC_TRUE`. Otherwise use `AutoInc` address.

Param *wSlaveAddress*

Slave address according to `bFixedAddress`

Param *pdwProcessVarInfoNumOf*

Process variables count

Return

`EC_E_NOERROR` or error code

```

ECError GetProcessVarInfoEx (
    DN_EC_T_VAR_DIRECTION eVarDirection,
    DN_EC_T_VAR_SOURCE eVarSource,
    bool bFixedAddress,
    ushort wSlaveAddress,
    out DN_EC_T_PROCESS_VAR_INFO_EX[] aoVarInfo,
    uint dwMaxProcessVarInfoNumOf,
    out uint pdwProcessVarInfoNumOf
)

```

Get process variables information.

Param eVarDirection

INPUTs, OUTPUTs, ... See EC_T_VAR_DIRECTION .

Param eVarSource

Slave, Master, ... See EC_T_VAR_SOURCE .

Param bFixedAddress

Use station address if EC_TRUE. Otherwise use AutoInc address.

Param wSlaveAddress

Slave address according to bFixedAddress

Param aoVarInfo

The read process variable extended information entries

Param dwMaxProcessVarInfoNumOf

Param pdwProcessVarInfoNumOf

Return

EC_E_NOERROR or error code

```

ECError TferSingleRawCmd (
    DN_EC_T_CMD eCmd,
    uint dwMemoryAddress,
    byte[] pbyData,
    ushort wLen,
    uint 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.

Param eCmd

EtherCAT command type. See EC_T_CMD

Param dwMemoryAddress

Slave memory address, depending on the command to be sent this is either a physical or a logical address

Param pbyData

[in, out] Buffer containing or receiving transferred data

Param wLen

Number of bytes to transfer

Param dwTimeout

Timeout [ms]

Return

- 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_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

***ECError* ReadSlaveRegister (**

bool *bFixedAddressing*,
 ushort *wSlaveAddress*,
 ushort *wRegisterOffset*,
 byte[] *pbyData*,
 ushort *wLen*,
 uint *dwTimeout*

)

Reads data from the ESC memory that have so far been transferred to a slave and received by the EC-Monitor.

!(EC_MONITOR) Reads data from the ESC memory of a specified slave.

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

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wRegisterOffset

Register offset. I.e. use 0x0130 to read the AL Status register.

Param pbyData

Buffer receiving transferred data

Param wLen

Number of bytes to receive

Param dwTimeout

Timeout [ms]

Return

- 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 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.

ECError ReadSlaveRegisterReq (

```
uint dwClientId,
uint dwTferId,
bool bFixedAddressing,
ushort wSlaveAddress,
ushort wRegisterOffset,
byte[] pbyData,
ushort 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.

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param dwTferId

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.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wRegisterOffset

Register offset, e.g. use 0x0130 to read the AL Status register

Param pbyData

Buffer receiving transferred data

Param wLen

Number of bytes to receive

Return

- 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.

ECError WriteSlaveRegister (

```
bool bFixedAddressing,
ushort wSlaveAddress,
ushort wRegisterOffset,
byte[] pbyData,
ushort wLen,
uint dwTimeout
```

)

Writes data into the ESC memory of a specified slave.

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

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

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wRegisterOffset

Register offset. E.g. use 0x0120 to write to the AL Control register.

Param pbyData

Buffer containing transferred data

Param wLen

Number of bytes to send

Param dwTimeout

Timeout [ms]

Return

- 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.

```
ECError WriteSlaveRegisterReq (
    uint dwClientId,
    uint dwTferId,
    bool bFixedAddressing,
    ushort wSlaveAddress,
    ushort wRegisterOffset,
    byte[] pbyData,
    ushort 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.

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

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param dwTferId

Transfer ID. The application can set this ID to identify the transfer. It will be passed back to the application within EC_T_SLAVEREGISTER_TRANSFER_NTFY_DESC.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wRegisterOffset

Register offset. E.g. use 0x0120 to write to the AL Control register.

Param pbyData

Buffer containing transferred data

Param wLen

Number of bytes to send

Return

- 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.

```
ECError QueueRawCmd (
    ushort wInvokeId,
    DN_EC_T_CMD eCmd,
    uint dwMemoryAddress,
    byte[] pbyData,
    ushort 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.

Param wInvokeId

Invoke ID to reassign the results to the sent CMD

Param eCmd

EtherCAT command

Param dwMemoryAddress

Slave memory address, depending on the command to be sent this is either a physical or a logical address

Param 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.

Param wLen

Number of bytes to transfer

Return

EC_E_NOERROR or error code

```
ECError CIntQueueRawCmd (
    uint dwClientId,
    ushort wInvokeId,
    DN_EC_T_CMD eCmd,
    uint dwMemoryAddress,
    byte[] pbyData,
    ushort wLen
)
```

Transfers a raw EtherCAT command to one or multiple slaves.

Using this function it is possible to exchange data between the master and the slaves. When the response to the queued frame is received, the notification EC_NOTIFY_RAWCMD_DONE is given for the appropriate client. This function queues a single EtherCAT command. Queued raw commands will be sent after sending cyclic process data values. If a timeout occurs the corresponding frame will be sent again, the timeout value and retry counter can be set using the master configuration parameters EC_T_INIT_MASTER_PARMS.dwEcatCmdTimeout and EC_T_INIT_MASTER_PARMS.dwEcatCmdMaxRetries. Using auto increment addressing (APRD, APWR, APRW) may lead to unexpected results in case the selected slave does not increment the working counter. In such cases the EtherCAT command would be handled by the slave directly behind the selected one. This function may not be called from within the JobTask's context.

Param dwClientId

Client ID to be notified (0 if all registered clients shall be notified)

Param wInvokeId

Invoke ID to reassign the results to the sent CMD

Param eCmd

EtherCAT command

Param dwMemoryAddress

Slave memory address, depending on the command to be sent this is either a physical or a logical address

Param 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.

Param wLen

Number of bytes to transfer

Return

- 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
- 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.

uint **GetNumConfiguredSlaves** ()

Returns the number of slaves which are configured in the ENI.

Return

Number of slaves

ECError **MbxTferAbort** (IntPtr *pMbxTfer*)

Abort a running mailbox transfer.

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

Param pMbxTfer

Mailbox transfer object created with emMbxTferCreate

Return

EC_E_NOERROR if successful

ECError **CIntSendRawMbx** (

uint *dwCIntId*,
byte[] *pbyMbxCmd*,
uint *dwMbxCmdLen*,
uint *dwTimeout*

)

Send a raw mailbox command.

Param dwCIntId

Client ID

Param pbyMbxCmd

Buffer containing the raw mailbox command starting with mailbox header

Param dwMbxCmdLen

Length of pbyMbxCmd buffer

Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

ECError **CoeSdoDownloadReq** (

IntPtr *pMbxTfer*,
uint *dwSlaveId*,
ushort *wObIndex*,
byte *byObSubIndex*,
uint *dwTimeout*,
uint *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.

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param wObIndex

Object index

Param byObSubIndex

Object sub index. If Complete Access only 0 or 1 allowed.

Param dwTimeout

Timeout [ms]

Param dwFlags

Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Return

- 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
- EC_ERROR_CODES_COE "CoE SDO error code"

***ECError* CoeSdoDownload (**

```
uint dwSlaveId,
ushort wObIndex,
byte byObSubIndex,
byte[] pbyData,
uint dwDataLen,
uint dwTimeout,
uint 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.

Param dwSlaveId

Slave ID

Param wObIndex

Object index

Param byObSubIndex

Object sub index. If Complete Access only 0 or 1 allowed.

Param pbyData**Param dwDataLen****Param dwTimeout**

Timeout [ms]

Param dwFlags

Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Return

- 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
- EC_ERROR_CODES_COE "CoE SDO error code"

ECError CoeSdoUploadReq (

IntPtr pMbxTfer,
uint dwSlaveId,
ushort wObIndex,
byte byObSubIndex,
uint dwTimeout,
uint 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.

Param pMbxTfer

Mailbox transfer object created with emMbxTferCreate

Param dwSlaveId

Slave ID

Param wObIndex

Object index

Param byObSubIndex

Object sub index. If Complete Access only 0 or 1 allowed.

Param dwTimeout

Timeout [ms]

Param dwFlags

Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Return

- 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 the ADS server is running
- EC_ERROR_CODES_COE "CoE SDO error code"

```
ECError CoeSdoUpload (
    uint dwSlaveId,
    ushort wObIndex,
    byte byObSubIndex,
    byte[] pbyData,
    uint dwDataLen,
    out uint pdwOutDataLen,
    uint dwTimeout,
    uint 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.

Param dwSlaveId

Slave ID

Param wObIndex

Object index

Param byObSubIndex

Object sub index. If Complete Access only 0 or 1 allowed.

Param pbyData
Param dwDataLen
Param pdwOutDataLen
Param dwTimeout

Timeout [ms]

Param dwFlags

Mailbox Flags. Bit 0: set if Complete Access (EC_MAILBOX_FLAG_SDO_COMPLETE).

Return

- 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 the ADS server is running
- EC_ERROR_CODES_COE "CoE SDO error code"

```

ECError CoeGetODListReq (
    IntPtr pMbxTfer,
    uint dwSlaveId,
    DN_EC_T_COE_ODLIST_TYPE eListType,
    uint dwTimeout
)

```

Initiates retrieval of a list of object IDs that have so far been transferred to a slave and received by the EC-Monitor and returns immediately.

!(EC_MONITOR) 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.

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.

Param pMbxTfer

Mailbox transfer

Param dwSlaveId

Slave ID

Param eListType

Which object types shall be transferred

Param dwTimeout

Timeout [ms]

Return

- 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
- EC_ERROR_CODES_COE “CoE SDO error code”

```
ECError CoeGetObjectDescReq (
    IntPtr pMbxTfer,
    uint dwSlaveId,
    ushort wObIndex,
    uint 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.

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param wObIndex

Object index

Param dwTimeout

Timeout [ms]

Return

- 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
- EC_ERROR_CODES_COE “CoE SDO error code”

```
ECError CoeGetEntryDescReq (
    IntPtr pMbxTfer,
    uint dwSlaveId,
    ushort wObIndex,
    byte byObSubIndex,
    byte byValueInfo,
    uint 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.

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param wObIndex

Object index

Param byObSubIndex

Object sub-index

Param byValueInfo

The value info bit mask includes which elements shall be in the response. See EC_COE_ENTRY_VALUEINFO “Value info flags” for available values.

Param dwTimeout

Timeout [ms]

Return

- 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
- EC_ERROR_CODES_COE “CoE SDO error code”

ECError CoeRxPdoTfer (IntPtr pMbxTfer, uint dwSlaveId, uint dwNumber, uint dwTimeout)

Initiate CoE RxPDO transfer.

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param dwNumber

PDO number

Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

```

ECError CoeProfileGetChannelInfo (
    bool bStationAddress,
    ushort wSlaveAddress,
    uint dwChannel,
    DN_EC_T_PROFILE_CHANNEL_INFO pInfo
)

```

Return information about a configured CoE profile channel from the ENI file.

Param *bStationAddress*

Param *wSlaveAddress*

Slave address according *bFixedAddressing*

Param *dwChannel*

Channel

Param *pInfo*

Channel info

Return

EC_E_NOERROR or error code

```

ECError FoeFileUpload (
    uint dwSlaveId,
    string achFileName,
    uint dwFileNameLen,
    byte[] pbData,
    uint dwDataLen,
    out uint pdwOutDataLen,
    uint dwPassword,
    uint 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.

Param *dwSlaveId*

Slave ID

Param *achFileName*

File name of slave file to read

Param *dwFileNameLen*

Length of slave file name in bytes

Param *pbData*

Buffer receiving transferred data

Param *dwDataLen*

Buffer length [bytes]

Param *pdwOutDataLen*

Length of received data [byte]

Param *dwPassword*

Slave password

Param *dwTimeout*

Timeout [ms]. The function will block at most for this time.

Return

- 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_TIMEOUT if dwTimeout elapsed during the API call
- EC_E_MASTER_RED_STATE_INACTIVE if Master Redundancy is configured and master is inactive
- EC_ERROR_CODES_FOE "FoE error code"

ECError **FoeFileDownload** (

```
uint dwSlaveId,
string achFileName,
uint dwFileNameLen,
byte[] pbyData,
uint dwDataLen,
uint dwPassword,
uint 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.

Param dwSlaveId

Slave ID

Param achFileName

File name of slave file to write

Param dwFileNameLen

Length of slave file name in bytes

Param pbyData

Buffer containing transferred data

Param dwDataLen

Buffer length [bytes]

Param dwPassword

Slave password

Param dwTimeout

Timeout [ms]. The function will block at most for this time.

Return

- 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_FOE_ERRCODE_NOTINBOOTSTRAP if slave in BOOTSTRAP and file-name not accepted by slave
- EC_E_MASTER_RED_STATE_INACTIVE if Master Redundancy is configured and master is inactive
- EC_ERROR_CODES_FOE "FoE error code"

```
ECError FoeUploadReq (
    IntPtr pMbxTfer,
    uint dwSlaveId,
    string achFileName,
    uint dwFileNameLen,
    uint dwPassword,
    uint 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.

EC_NOWAIT as a timeout is still accepted for reasons of compatibility and sets the timeout to 10 seconds

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param achFileName

File name of slave file to read

Param dwFileNameLen

Length of slave file name in bytes

Param dwPassword

Slave password

Param dwTimeout

Timeout [ms]

Return

- 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
- EC_ERROR_CODES_FOE "FoE error code"

```
ECError FoeSegmentedUploadReq (
    IntPtr pMbxTfer,
    uint dwSlaveId,
    string szFileName,
    uint dwFileNameLen,
    uint dwFileSize,
    uint dwPassword,
    uint 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.

EC_NOWAIT as a timeout is still accepted for reasons of compatibility and sets the timeout to 10 seconds

Param pMbxTfer

Pointer to the corresponding mailbox transfer object. EC_T_MBXTFER.pbyMbxTferData: next chunk, EC_T_MBXTFER.dwDataLen: next chunk size.

Param dwSlaveId

Slave ID

Param szFileName

File name of slave file to write. Only evaluated when initiating the request.

Param dwFileNameLen

Length of slave file name in bytes

Param dwFileSize

Used only for progress information

Param dwPassword

Slave password. Only evaluated when initiating the request.

Param dwTimeout

Overall timeout [ms] of the FoE transfer.

Return

- 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
- EC_ERROR_CODES_FOE "FoE error code"

ECError **FoeDownloadReq** (

IntPtr pMbxTfer,
uint dwSlaveId,
string achFileName,
uint dwFileNameLen,
uint dwPassword,
uint 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.

EC_NOWAIT as a timeout is still accepted for reasons of compatibility and sets the timeout to 10 seconds

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param achFileName

File name of slave file to write

Param dwFileNameLen

Length of slave file name in bytes

Param dwPassword

Slave password

Param dwTimeout

Timeout [ms]

Return

- 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
- EC_ERROR_CODES_FOE "FoE error codes"

ECError FoeSegmentedDownloadReq (

```

IntPtr pMbxTfer,
uint dwSlaveId,
string szFileName,
uint dwFileNameLen,
uint dwFileSize,
uint dwPassword,
uint 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.

EC_NOWAIT as a timeout is still accepted for reasons of compatibility and sets the timeout to 10 seconds

Param pMbxTfer

Pointer to the corresponding mailbox transfer object. EC_T_MBXTFER.pbyMbxTferData: next chunk, EC_T_MBXTFER.dwDataLen: next chunk size.

Param dwSlaveId

Slave ID

Param szFileName

File name of slave file to write. Only evaluated when initiating the request.

Param dwFileNameLen

Length of slave file name in bytes

Param dwFileSize

Complete file size (mandatory). Used also for progress information. Only evaluated when initiating the request.

Param dwPassword

Slave password. Only evaluated when initiating the request.

Param dwTimeout

Overall timeout [ms] of the FoE transfer.

Return

- 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
- EC_ERROR_CODES_FOE "FoE error code"

```
ECError SoeWrite (
    uint dwSlaveId,
    byte byDriveNo,
    byte[] pbyElementFlags,
    ushort wIDN,
    byte[] pbyData,
    uint dwDataLen,
    uint 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.

Param dwSlaveId

Slave ID

Param byDriveNo

Drive number to address inside slave device

Param pbyElementFlags

[in/out] SoE ElementFlags. Flag indicating elements to address.

Param wIDN

IDN of the object to address

Param pbyData

Buffer containing transferred data

Param dwDataLen

Buffer length [byte]

Param dwTimeout

Timeout [ms]

Return

- 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
- EC_ERROR_CODES_SOE “SoE error code”

```

ECError SoeRead (
    uint dwSlaveId,
    byte byDriveNo,
    byte[] pbyElementFlags,
    ushort wIDN,
    byte[] pbyData,
    uint dwDataLen,
    out uint pdwOutDataLen,
    uint 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.

Param dwSlaveId

Slave ID

Param byDriveNo

Drive number to address inside slave device

Param pbyElementFlags

[in/out] SoE ElementFlags. Flag indicating elements to address.

Param wIDN

IDN of the object to address

Param pbyData

Buffer receiving transferred data

Param dwDataLen

Buffer length [byte]

Param pdwOutDataLen

Length of received data [byte]

Param dwTimeout

Timeout [ms]

Return

- 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
- EC_ERROR_CODES_SOE “SoE error code”

```
ECError SoeAbortProcCmd (
    uint dwSlaveId,
    byte byDriveNo,
    byte[] pbyElementFlags,
    ushort wIDN,
    uint 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.

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

Param dwSlaveId

Slave ID

Param byDriveNo

Drive number to address inside slave device

Param pbyElementFlags

[in/out] SoE ElementFlags. Flag indicating elements to address.

Param wIDN

IDN of the object to address

Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

```
ECError SoeWriteReq (
    IntPtr pMbxTfer,
    uint dwSlaveId,
    byte byDriveNo,
    byte[] pbyElementFlags,
    ushort wIDN,
    uint dwTimeout
)
```

Requests an SoE SSC Write and returns immediately.

This function may be called from within the JobTask’s context.

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param byDriveNo

Drive number to address inside slave device

Param pbyElementFlags

[in/out] SoE ElementFlags. Flag indicating elements to address.

Param wIDN

IDN of the object to address

Param dwTimeout

Timeout [ms]

Return

- 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
- EC_ERROR_CODES_SOE "SoE error code"

```
ECError SoeReadReq (
    IntPtr pMbxTfer,
    uint dwSlaveId,
    byte byDriveNo,
    byte[] pbyElementFlags,
    ushort wIDN,
    uint dwTimeout
)
```

Requests an SoE SSC Read and returns immediately.

This function may be called from within the JobTask's context.

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param byDriveNo

Drive number to address inside slave device

Param pbyElementFlags

[in/out] SoE ElementFlags. Flag indicating elements to address.

Param wIDN

IDN of the object to address

Param dwTimeout

Timeout [ms]. Must not be set to EC_NOWAIT.

Return

- 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
- EC_ERROR_CODES_SOE "SoE error code"

ECError **AoeGetSlaveNetId** (uint dwSlaveId, out *DN_EC_T_AOE_NETID* poAoeNetId)

Retrieve the NetID of a specific EtherCAT device.

Param dwSlaveId

Slave ID

Param poAoeNetId

AoE NetID of the corresponding slave

Return

- 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

ECError **AoeRead** (
 uint dwSlaveId,
DN_EC_T_AOE_NETID poTargetNetId,
 ushort wTargetPort,
 uint dwIndexGroup,
 uint dwIndexOffset,
 uint dwDataLen,
 byte[] pbyData,
 out uint pdwDataOutLen,
 out uint pdwErrorCode,
 out uint pdwCmdResult,
 uint dwTimeout
)

Execute an AoE mailbox read request to an EtherCAT slave device.

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

Param dwSlaveId

Slave ID

Param poTargetNetId

Target NetID

Param wTargetPort

Target port

Param dwIndexGroup

AoE read command index group

Param dwIndexOffset

AoE read command index offset

Param dwDataLen

Buffer length [bytes]

Param pbyData

Buffer receiving transferred data

Param pdwDataOutLen

Number of bytes read from the target device

Param pdwErrorCode

AoE response error code

Param pdwCmdResult

AoE read command result code

Param dwTimeout

Timeout [ms]. The function will block at most for this time.

Return

- 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

```

ECError AoeReadReq (
    IntPtr pMbxTfer,
    uint dwSlaveId,
    DN_EC_T_AOE_NETID poTargetNetId,
    ushort wTargetPort,
    uint dwIndexGroup,
    uint dwIndexOffset,
    uint 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.

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param poTargetNetId

Target NetID

Param wTargetPort

Target port

Param dwIndexGroup

AoE read command index group

Param dwIndexOffset

AoE read command index offset

Param dwTimeout

Timeout [ms]

Return

- 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

```
ECError AoeWrite (
    uint dwSlaveId,
    DN_EC_T_AOE_NETID poTargetNetId,
    ushort wTargetPort,
    uint dwIndexGroup,
    uint dwIndexOffset,
    uint dwDataLen,
    byte[] pbyData,
    out uint pdwErrorCode,
    out uint pdwCmdResult,
    uint dwTimeout
)
```

Execute an AoE mailbox write request to an EtherCAT slave device.

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

Param dwSlaveId

Slave ID

Param poTargetNetId

Target NetID

Param wTargetPort

Target port

Param dwIndexGroup

AoE write command index group

Param dwIndexOffset

AoE write command index offset

Param dwDataLen

Buffer length [bytes]

Param pbyData

Buffer containing transferred data

Param pdwErrorCode

Pointer to AoE response error code

Param pdwCmdResult

Pointer to AoE write command result code

Param dwTimeout

Timeout [ms]. The function will block at most for this time.

Return

- 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

```
ECError AoeWriteReq (
    IntPtr pMbxTfer,
    uint dwSlaveId,
    DN_EC_T_AOE_NETID poTargetNetId,
    ushort wTargetPort,
    uint dwIndexGroup,
    uint dwIndexOffset,
    uint 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.

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param poTargetNetId

Target NetID

Param wTargetPort

Target port

Param dwIndexGroup

AoE write command index group

Param dwIndexOffset

AoE write command index offset

Param dwTimeout

Timeout [ms]

Return

- 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

```

ECError AoeReadWrite (
    uint dwSlaveId,
    DN_EC_T_AOE_NETID poTargetNetId,
    ushort wTargetPort,
    uint dwIndexGroup,
    uint dwIndexOffset,
    uint dwReadDataLen,
    uint dwWriteDataLen,
    byte[] pbyData,
    out uint pdwDataOutLen,
    out uint pdwErrorCode,
    out uint pdwCmdResult,
    uint dwTimeout
)

```

Execute an AoE mailbox read/write request to an EtherCAT slave device.

Param *dwSlaveId*

Slave ID

Param *poTargetNetId*

Target NetID

Param *wTargetPort*

Target port

Param *dwIndexGroup*

AoE read/write command index group

Param *dwIndexOffset*

AoE read/write command index offset

Param *dwReadDataLen*

Number of bytes to read from the target device

Param *dwWriteDataLen*

Number of bytes to write to the target device

Param *pbyData*

[in, out] Buffer containing and receiving transferred data

Param *pdwDataOutLen*

Number of bytes read from the target device

Param *pdwErrorCode*

Pointer to AoE response error code

Param *pdwCmdResult*

Pointer to AoE write command result code

Param *dwTimeout*

Timeout [ms]. The function will block at most for this time. EC_NOWAIT is not valid.

Return

- 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

```

ECError AoeWriteControl (
    uint dwSlaveId,
    DN_EC_T_AOE_NETID poTargetNetId,
    ushort wTargetPort,
    ushort wAoEState,
    ushort wDeviceState,
    uint dwDataLen,
    byte[] pbyData,
    out uint pdwErrorCode,
    out uint pdwCmdResult,
    uint dwTimeout
)

```

Execute an AoE mailbox write control request to an EtherCAT slave device.

Param *dwSlaveId*

Slave ID

Param *poTargetNetId*

Target NetID. The Target NetID of a AoE slave device can be retrieved by `ecatAoeGetSlaveNetId()`.

Param *wTargetPort*

Target port

Param *wAoEState*

AoE state

Param *wDeviceState*

Device specific state

Param *dwDataLen*

Buffer length [bytes]

Param *pbyData*

Buffer containing transfered data

Param *pdwErrorCode*

Pointer to AoE response error code

Param *pdwCmdResult*

Pointer to AoE write command result code

Param *dwTimeout*

Timeout [ms]. The function will block at most for this time. `EC_NOWAIT` is not valid.

Return

- `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

```

ECError VoeRead (
    uint dwSlaveId,
    byte[] pbyData,
    uint dwDataLen,
    out uint pdwOutDataLen,
    uint 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.

Param dwSlaveId

Slave ID

Param pbyData

Buffer receiving transferred data

Param dwDataLen

Buffer length [byte]

Param pdwOutDataLen

Length of received data [byte]

Param dwTimeout

Timeout [ms]. The function will block at most for this time. EC_NOWAIT returns the function immediately.

Return

- EC_E_NOERROR if the VoE slave has provided some VoE data
- 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_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

ECError **VoeWrite** (uint dwSlaveId, byte[] pbyData, uint dwDataLen, uint 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.

Param dwSlaveId

Slave ID

Param pbyData

Buffer containing transferred data

Param dwDataLen

Buffer length [bytes]. The maximum data length including 6 bytes for the mailbox header is given at EC_T_CFG_SLAVE_INFO.dwMbxOutSize.

Param dwTimeout

Timeout [ms]. The function will block at most for this time.

Return

EC_E_NOERROR or error code

ECError **VoeWriteReq** (IntPtr pMbxTfer, uint dwSlaveId, uint 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.

Param pMbxTfer

Mailbox transfer object

Param dwSlaveId

Slave ID

Param dwTimeout

Timeout [ms], EC_NOWAIT returns the function immediately

Return

EC_E_NOERROR or error code

ECError **GetProcessData** (

bool *bOutputData*,

uint *dwOffset*,

byte[] *pbData*,

uint *dwLength*,

uint *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.

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

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

Param bOutputData

EC_TRUE: read output data, EC_FALSE: read input data

Param dwOffset

Byte offset in Process data to read from

Param pbData

Buffer receiving transferred data

Param dwLength
Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

ECError **SetProcessData** (

bool *bOutputData*,

uint *dwOffset*,

byte[] *pbData*,

uint *dwLength*,

uint *dwTimeout*

)

Write Process data synchronized.

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

Param bOutputData

EC_TRUE: write output data, EC_FALSE: write input data

Param dwOffset

Byte offset in Process data to write to

Param pbData

Buffer containing transferred data

Param dwLength
Param dwTimeout
Timeout [ms]

Return
EC_E_NOERROR or error code

```
ECError SetProcessDataBits (
    bool bOutputData,
    uint dwBitOffsetPd,
    byte[] pbDataSrc,
    uint dwBitLengthSrc,
    uint 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.

Param bOutputData
EC_TRUE: write output data, EC_FALSE: write input data

Param dwBitOffsetPd
Bit offset in Process data image

Param pbDataSrc
Param dwBitLengthSrc

Param dwTimeout
Timeout [ms]. The timeout value must not be set to EC_NOWAIT.

Return
EC_E_NOERROR or error code

```
ECError GetProcessDataBits (
    bool bOutputData,
    uint dwBitOffsetPd,
    byte[] pbDataDst,
    uint dwBitLengthDst,
    uint 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.

Param bOutputData
EC_TRUE: read output data, EC_FALSE: write input data

Param dwBitOffsetPd
Bit offset in Process data image

Param pbDataDst
Param dwBitLengthDst

Param dwTimeout
Timeout [ms]. The timeout value must not be set to EC_NOWAIT.

Return
EC_E_NOERROR or error code

```
ECError ForceProcessDataBits (
    uint dwClientId,
    bool bOutputData,
    uint dwBitOffsetPd,
    ushort wBitLength,
    byte[] pbData,
    uint 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.

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param bOutputData

EC_TRUE: write output data, EC_FALSE: write input data

Param dwBitOffsetPd

Bit offset in Process data image

Param wBitLength

Param pbData

Buffer containing transferred data

Param dwTimeout

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

Return

EC_E_NOERROR or error code

```
ECError ReleaseProcessDataBits (
    uint dwClientId,
    bool bOutputData,
    uint dwBitOffsetPd,
    ushort wBitLength,
    uint 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.

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param bOutputData

EC_TRUE: write output data, EC_FALSE: write input data

Param dwBitOffsetPd

Bit offset in Process data image

Param wBitLength

Number of bits that shall be written to the process image.

Param dwTimeout

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

Return

EC_E_NOERROR or error code

ECError **ReleaseAllProcessDataBits** (uint *dwClientId*, uint *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.

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param dwTimeout

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

Return

EC_E_NOERROR or error code

uint **GetNumConnectedSlaves** ()

Get number of currently connected slaves.

Return

Number of connected slaves

uint **GetNumConnectedSlavesMain** ()

Get the number of currently connected Slaves to main interface.

Return

Number of connected slaves at main interface

uint **GetNumConnectedSlavesRed** ()

Get the number of currently connected Slaves to redundancy interface.

Return

Number of connected slaves at redundancy interface

ECError **ReloadSlaveEEPROM** (bool *bFixedAddressing*, ushort *wSlaveAddress*, uint *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.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param dwTimeout

Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

```
ECError ReloadSlaveEEPROMReq (
    uint dwClientId,
    uint dwTferId,
    bool bFixedAddressing,
    ushort wSlaveAddress,
    uint 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.

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param dwTferId

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.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

```
ECError ResetSlaveController (bool bFixedAddressing, ushort wSlaveAddress, uint 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_PARAMS.dwMaxAcycFramesPerCycle must be at least 3, otherwise an error is returned. This function may not be called from within the JobTask's context.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param dwTimeout

Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

- EC_E_NOERROR or error code
- EC_E_NOTSUPPORTED if EC_T_INIT_MASTER_PARAMS.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

ECError **AssignSlaveEEPROM** (

bool *bFixedAddressing*,
 ushort *wSlaveAddress*,
 bool *bSlavePDIAccessEnable*,
 bool *bForceAssign*,
 uint *dwTimeout*

)

Set EEPROM Assignment to PDI or EtherCAT Master.

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

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param bSlavePDIAccessEnable

EC_TRUE: EEPROM assigned to slave PDI application,

Param bForceAssign

Force Assignment of EEPROM (only for ECat Master Assignment)

Param dwTimeout

Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

ECError **AssignSlaveEEPROMReq** (

uint *dwClientId*,
 uint *dwTferId*,
 bool *bFixedAddressing*,
 ushort *wSlaveAddress*,
 bool *bSlavePDIAccessEnable*,
 bool *bForceAssign*,
 uint *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.

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param dwTferId

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.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param bSlavePDIAccessEnable

EC_TRUE: EEPROM assigned to slave PDI application, EC_FALSE: EEPROM assigned to EC-Master

Param bForceAssign

Force Assignment of EEPROM (only for ECat Master Assignment)

Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

```
ECError ActiveSlaveEEPROM (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    out bool pbSlavePDIAccessActive,
    uint dwTimeout
)
```

Check whether EEPROM is marked access active by Slave PDI application.

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

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param pbSlavePDIAccessActive

Pointer to Boolean value: EC_TRUE: EEPROM active by PDI application, EC_FALSE: EEPROM not active

Param dwTimeout

Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

```
ECError ActiveSlaveEEPROMReq (
    uint dwClientId,
    uint dwTferId,
    bool bFixedAddressing,
    ushort wSlaveAddress,
    out bool pbSlavePDIAccessActive,
    uint 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.

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param dwTferId

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.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param pbSlavePDIAccessActive

Pointer to Boolean value: EC_TRUE: EEPROM active by PDI application, EC_FALSE: EEPROM not active. Must be valid until operation complete.

Param dwTimeout

Timeout [ms]. The function will block at most for this time. The timeout value must not be set to EC_NOWAIT.

Return

EC_E_NOERROR or error code

***ECError* HCAcceptTopoChange ()**

Accept last detected topology change.

If Hot connect is configured in manual mode by EC_IOCTL_HC_SETMODE, the master will generate the notifications EC_NOTIFY_HC_PROBEALLGROUPS or EC_NOTIFY_HC_DETECTADDGROUPS after a topology change was detected. This function will set all new detected slaves to the current master state.

Return

EC_E_NOERROR or error code

***ECError* HCGetNumGroupMembers (uint dwGroupIndex)**

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

Param dwGroupIndex

Index of Hot-Connect group, 0 is the mandatory group

Return

Number of slaves

***ECError* HCGetSlaveIdsOfGroup (**

uint dwGroupIndex,
out uint adwSlaveId,
uint dwMaxNumSlaveIds

)

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

Param dwGroupIndex

Index of Hot-Connect group, 0 is the mandatory group

Param adwSlaveId

Preallocated Slave ID list buffer

Param dwMaxNumSlaveIds

Size of Slave ID list buffer

Return

EC_E_NOERROR or error code

***ECError* GetHcGroupNumOf ()**

Get number of Hot-Connect groups.

Return

number of HC groups

***ECError* GetHcGroupInfo (uint dwHcGroupIdx, DN_EC_T_HC_GROUP_INFO pHcGroupInfo)**

Get informations of a specific Hot-Connect group.

Param dwHcGroupIdx
Param pHcGroupInfo

Hot connect group info

Return

EC_E_NOERROR or error code

```
ECError SetSlavePortState (
    uint dwSlaveId,
    ushort wPort,
    bool bClose,
    bool bForce,
    uint 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.

Param *dwSlaveId*

Slave ID

Param *wPort*

Port to open or close. Can be ESC_PORT_A, ESC_PORT_B, ESC_PORT_C, ESC_PORT_D.

Param *bClose*

EC_TRUE: close port, EC_FALSE: open port

Param *bForce*

EC_TRUE: port will be closed or open, EC_FALSE: port will be set in AutoClose mode

Param *dwTimeout*

Timeout [ms]

Return

- 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
- EC_E_MASTER_RED_STATE_INACTIVE if Master Redundancy is configured and master is inactive

```
ECError SetSlavePortStateReq (
    uint dwClientId,
    uint dwTferId,
    uint dwSlaveId,
    ushort wPort,
    bool bClose,
    bool bForce,
    uint 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.

Param *dwClientId*

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param *dwTferId*

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.

Param dwSlaveId

Slave ID

Param wPort

Port to open or close. Can be ESC_PORT_A, ESC_PORT_B, ESC_PORT_C, ESC_PORT_D.

Param bClose

EC_TRUE: close port, EC_FALSE: open port

Param bForce

EC_TRUE: port will be closed or open, EC_FALSE: port will be set in AutoClose mode

Param dwTimeout

Timeout [ms]

Return

- 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

ECError **SlaveSerializeMbxTfers** (uint 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.

Param dwSlaveId

Slave ID

Return

- 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
- EC_E_NO_MBX_SUPPORT if slave does not support mailbox transfers

ECError **SlaveParallelMbxTfers** (uint 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.

Param dwSlaveId

Slave ID

Return

- 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
- EC_E_NO_MBX_SUPPORT if slave does not support mailbox transfers

ECError **DcEnable** ()

Enable DC Support.

Return

EC_E_NOERROR or error code

ECError **DcDisable** ()

Disable DC Support.

Return

EC_E_NOERROR or error code

ECError **DcIsEnabled** (out bool *pbDcIsEnabled*)

Determines if DC is enabled and used.

Param pbDcIsEnabled

EC_TRUE if DC is enabled

Return

EC_E_NOERROR or error code

ECError **DcConfigure** (*DN_EC_T_DC_CONFIGURE* *pDcConfigure*)

Configure the distributed clocks.

- Set the DC synchronization settling time ([ms]).
- Set the DC slave limit for the wire or'ed clock deviation value. This value determines whether the slave clocks are synchronized or not.
- Configure the ARMW burst frames to compensate the static deviations of the clock speeds.

Param pDcConfigure

Configuration parameter a pointer to a structure of type EC_T_DC_CONFIGURE

Return

EC_E_NOERROR or error code

ECError **DcContDelayCompEnable** ()

Enable the continuous propagation delay compensation.

Calling this function will generate a propagation delay measurement every 30s. The result of the measurement is used to correct the propagation delay values on the bus.

Return

EC_E_NOERROR or error code

ECError **DcContDelayCompDisable** ()

Disable the continuous propagation delay compensation.

Return

EC_E_NOERROR or error code

ECError **DcmConfigure** (*DN_EC_T_DCM_CONFIG* *pDcmConfig*, uint *dwInSyncTimeout*)

Configure DC master synchronization.

Param pDcmConfig

Configuration information, a pointer to a structure of type EC_T_DCM_CONFIG

Param dwInSyncTimeout

Currently not implemented

Return

EC_E_NOERROR or error code

```

ECError DcmGetStatus (
    out uint pdwErrorCode,
    out int pnDiffCur,
    out int pnDiffAvg,
    out int pnDiffMax
)

```

Get DC master synchronization controller status.

Param *pdwErrorCode*

Pointer to current error code of the DCM controller. Possible values are:

Param *pnDiffCur*

Pointer to current difference between set value and actual value of controller in nanoseconds

Param *pnDiffAvg*

Pointer to average difference between set value and actual value of controller in nanoseconds

Param *pnDiffMax*

Pointer to maximum difference between set value and actual value of controller in nanoseconds

Return

- EC_E_NOERROR if status retrieval was successful
- EC_E_NOTSUPPORTED if the DC feature is not supported/switched off. EC-Master stack has to be compiled with DC support see “define INCLUDE_DC_SUPPORT”.
- EC_NULL does not appear in normal flow, if EC_NULL is returned we are in a very exceptional case where `m_poDcm == NULL` (SW error detection)

```

ECError DcxGetStatus (
    out uint pdwErrorCode,
    out int pnDiffCur,
    out int pnDiffAvg,
    out int pnDiffMax,
    out long pnTimeStampDiff
)

```

Get DC master external synchronization controller status.

Param *pdwErrorCode*

DCX controller error code

Param *pnDiffCur*

Current difference between set value and actual value of controller in nanoseconds

Param *pnDiffAvg*

Average difference between set value and actual value of controller in nanoseconds

Param *pnDiffMax*

Maximum difference between set value and actual value of controller in nanoseconds

Param *pnTimeStampDiff*

Difference between external and internal timestamps

Return

EC_E_NOERROR or error code

```

ECError DcmResetStatus ()

```

Reset DC master synchronization controller status, average and maximum difference between set value and actual value.

Return

EC_E_NOERROR or error code

ECError **DcmGetBusShiftConfigured** (**out** bool *pbBusShiftConfigured*)

Determines if DCM Bus Shift is configured/possible in configuration (ENI file).

Param pbBusShiftConfigured

EC_TRUE if DCM bus shift mode is supported by the current configuration

Return

EC_E_NOERROR or error code

ECError **DcmShowStatus** ()

Show DC master synchronization status as DbgMsg (for development purposes only).

Return

EC_E_NOERROR or error code

ECError **DcmGetAdjust** (**out** int *pnAdjustPermil*)

Returns the current adjustment value for the timer.

bCtlOff must be set to EC_TRUE in EC_T_DCM_CONFIG to enable external adjustment.

Param pnAdjustPermil

Current adjustment value of the timer

Return

EC_E_NOERROR or error code

ECError **GetSlaveInfo** (
 bool *bFixedAddressing*,
 ushort *wSlaveAddress*,
out *DN_EC_T_GET_SLAVE_INFO* *pGetSlaveInfo*
)

Get Slave Info.

Use emGetCfgSlaveInfo or emGetBusSlaveInfo instead

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param pGetSlaveInfo

Slave information

Return

EC_E_NOERROR or error code

ECError **GetCfgSlaveInfo** (
 bool *bStationAddress*,
 ushort *wSlaveAddress*,
out *DN_EC_T_CFG_SLAVE_INFO* *pSlaveInfo*
)

Return information about a configured slave from the ENI file.

Param bStationAddress
Param wSlaveAddress

Slave address according bFixedAddressing

Param pSlaveInfo

Information about the slave

Return

EC_E_NOERROR or error code

```

ECError GetCfgSlaveEoeInfo (
    bool bStationAddress,
    ushort wSlaveAddress,
    out DN_EC_T_CFG_SLAVE_EOE_INFO pSlaveEoeInfo
)

```

Return EoE information about a configured slave from the ENI file.

Param bStationAddress

Param wSlaveAddress

Slave address according bFixedAddressing

Param pSlaveEoeInfo

Information about the slave

Return

- 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

```

ECError GetCfgSlaveSmInfo (
    bool bStationAddress,
    ushort wSlaveAddress,
    out DN_EC_T_CFG_SLAVE_SM_INFO pSlaveSmInfo
)

```

Return SyncManager information of a configured slave from the ENI file.

Param bStationAddress

Param wSlaveAddress

Slave address according bFixedAddressing

Param pSlaveSmInfo

Information about the slave.

Return

EC_E_NOERROR or error code

```

ECError GetBusSlaveInfo (
    bool bStationAddress,
    ushort wSlaveAddress,
    out DN_EC_T_BUS_SLAVE_INFO pSlaveInfo
)

```

Return information about a slave connected to the EtherCAT bus.

Param bStationAddress

Param wSlaveAddress

Slave address according bFixedAddressing

Param pSlaveInfo

Information from the slave

Return

- 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

```
ECError GetSlaveInpVarInfoNumOf (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    out ushort pwSlaveInpVarInfoNumOf
)
```

Gets the number of input variables of a specific slave.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param pwSlaveInpVarInfoNumOf

Number of found process variable entries

Return

- 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

```
ECError GetSlaveOutpVarInfoNumOf (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    out ushort pwSlaveOutpVarInfoNumOf
)
```

Gets the number of output variables of a specific slave.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param pwSlaveOutpVarInfoNumOf

Number of found process variables

Return

EC_E_NOERROR or error code

```
ECError GetSlaveInpVarInfo (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    ushort wNumOfVarsToRead,
    out DN_EC_T_PROCESS_VAR_INFO[] pSlaveProcVarInfoEntries,
    out ushort pwReadEntries
)
```

Gets the number of input variables of a specific slave.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wNumOfVarsToRead
Param pSlaveProcVarInfoEntries
Param pwReadEntries
Return

- 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

```
ECError GetSlaveInpVarInfoEx (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    ushort wNumOfVarsToRead,
    out DN_EC_T_PROCESS_VAR_INFO_EX[] pSlaveProcVarInfoEntries,
    out ushort pwReadEntries
)
```

Gets the input process variable extended information entries of a specific slave.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wNumOfVarsToRead

Number process variable entries that have been stored in pSlaveProcVarInfoEntries

Param pSlaveProcVarInfoEntries

Number process variable entries that have been stored in pSlaveProcVarInfoEntries

Param pwReadEntries

The number of read process variable information entries

Return

- 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

```
ECError GetSlaveOutpVarInfo (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    ushort wNumOfVarsToRead,
    out DN_EC_T_PROCESS_VAR_INFO[] pSlaveProcVarInfoEntries,
    out ushort pwReadEntries
)
```

Gets the number of output variables of a specific slave.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wNumOfVarsToRead

Param pSlaveProcVarInfoEntries

Param pwReadEntries

Return

EC_E_NOERROR or error code

```
ECError GetSlaveOutpVarInfoEx (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    ushort wNumOfVarsToRead,
    out DN_EC_T_PROCESS_VAR_INFO_EX[] pSlaveProcVarInfoEntries,
    out ushort pwReadEntries
)
```

Gets the output process variable extended information entries of a specific slave.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wNumOfVarsToRead

Number of process variable information entries

Param pSlaveProcVarInfoEntries

The read process extended variable entries

Param pwReadEntries

The number of read process variable information entries

Return

EC_E_NOERROR or error code

```
ECError GetSlaveOutpVarByObjectEx (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    ushort wIndex,
    ushort wSubIndex,
    out DN_EC_T_PROCESS_VAR_INFO_EX pProcessVarInfoEntry
)
```

Gets the input process variable extended information entry by object index, subindex of a specific slave.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wIndex

Object index

Param wSubIndex

Object sub index

Param pProcessVarInfoEntry

Process variable extended information entry

Return

EC_E_NOERROR or error code

```

ECError GetSlaveInpVarByObjectEx (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    ushort wIndex,
    ushort wSubIndex,
    out DN_EC_T_PROCESS_VAR_INFO_EX pProcessVarInfoEntry
)

```

Gets the input process variable extended information entry by object index, subindex of a specific slave.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wIndex

Object index

Param wSubIndex

Object sub index

Param pProcessVarInfoEntry

Process variable extended information entry

Return

EC_E_NOERROR or error code

```

ECError FindOutpVarByName (
    string szVariableName,
    out DN_EC_T_PROCESS_VAR_INFO pSlaveOutpVarInfo
)

```

Finds an output process variable information entry by the variable name.

Param szVariableName

Variable name

Param pSlaveOutpVarInfo**Return**

EC_E_NOERROR or error code

```

ECError FindOutpVarByNameEx (
    string szVariableName,
    out DN_EC_T_PROCESS_VAR_INFO_EX pProcessVarInfoEntry
)

```

Finds an output process variable extended information entry by the variable name.

Param szVariableName

Variable name

Param pProcessVarInfoEntry

Process variable extended information entry

Return

EC_E_NOERROR or error code

```

ECError FindInpVarByName (
    string szVariableName,
    out DN_EC_T_PROCESS_VAR_INFO pProcessVarInfoEntry
)

```

Finds an input process variable information entry by the variable name.

Param szVariableName

Variable name

Param pProcessVarInfoEntry

Process variable information entry

Return

EC_E_NOERROR or error code

```
ECError FindInpVarByNameEx (
    string szVariableName,
    out DN_EC_T_PROCESS_VAR_INFO_EX pProcessVarInfoEntry
)
```

Finds an input process variable extended information entry by the variable name.

Param szVariableName

Variable name

Param pProcessVarInfoEntry

Process variable extended information entry

Return

EC_E_NOERROR or error code

```
ECError EthDbgMsg (byte byEthTypeByte0, byte byEthTypeByte1, string szMsg)
```

Send a debug message to the EtherCAT Link Layer.

This feature can be used for debugging purposes.

Param byEthTypeByte0

Ethernet type byte 0

Param byEthTypeByte1

Ethernet type byte 1

Param szMsg

Message to send to link layer

Return

EC_E_NOERROR or error code

```
ECError BlockNode (DN_EC_T_SB_MISMATCH_DESC pMisMatch, uint dwTimeout)
```

Blocks a slave on a specific port.

If an invalid slave node is connected, which causes the bus topology scan to fail, the previous port, where this node is connected to can be shut down with this call. This allows a Hot-Connect system to not be disturbed, if unknown nodes are connected. If this function will be executed on a Hot-Connect member (a slave which is part of a hot connect group) the complete hot connect group will be excluded from the bus. This function may only be called from within the JobTask's context with parameter dwTimeout set to EC_NOWAIT.

Param pMisMatch

Pointer to EC_T_SB_MISMATCH_DESC carrying mismatch descriptor.

Param dwTimeout

Timeout [ms]. The function will block at most for this time.

Return

EC_E_NOERROR or error code

```
ECError OpenBlockedPorts (uint dwTimeout)
```

Opens all blocked ports.

This call allows re-opening all blocked ports to check whether mismatch cause is removed from bus. This function may only be called from within the JobTask's context with parameter dwTimeout set to EC_NOWAIT.

Param dwTimeout

Timeout [ms]. The function will block at most for this time.

Return

EC_E_NOERROR or error code

ECError ForceTopologyChange ()

Force changed topology.

Trigger HC State Machine.

Return

EC_E_NOERROR or error code

ECError IsTopologyChangeDetected (out bool pbTopologyChangeDetected)

Returns whether topology change detected.

Param pbTopologyChangeDetected

Pointer to EC_T_BOOL value: EC_TRUE if Topology Change Detected, EC_FALSE if not

Return

- 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

ECError IsTopologyKnown (out bool pbTopologyKnown)

Returns whether topology known.

Param pbTopologyKnown

Topology known

Return

EC_E_NOERROR or error code

ECError GetBusTime (out ulong pqwBusTime)

This function returns the actual bus time in nanoseconds.

Param pqwBusTime

Bus time [ns]

Return

EC_E_NOERROR or error code

ECError IsSlavePresent (uint dwSlaveId, out bool pbPresence)

Returns whether a specific slave is currently connected to the Bus.

This function may be called from within the JobTask.

Param dwSlaveId

Slave ID

Param pbPresence

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

Return

- 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_NOTFOUND if no slave matching dwSlaveId can be found

DN_EC_PTS_STATE PassThroughSrvGetStatus ()

Gets the status of the Pass-through server.

Return

EC_E_NOERROR or error code

ECError **PassThroughSrvStart** (
DN_EC_T_PTS_SRV_START_PARMS *poPtsStartParams*,
uint *dwTimeout*
)

Starts the Pass Through Server.

Param poPtsStartParams

Pass through server start parameter

Param dwTimeout

Timeout

Return

EC_E_NOERROR or error code

ECError **PassThroughSrvStop** (uint *dwTimeout*)

Stops the Pass Through Server.

Param dwTimeout

Timeout

Return

EC_E_NOERROR or error code

ECError **PassThroughSrvEnable** (uint *dwTimeout*)

Enables the Pass-through server.

Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

ECError **PassThroughSrvDisable** (uint *dwTimeout*)

Disables the Pass-through server.

Param dwTimeout

Timeout [ms]

Return

EC_E_NOERROR or error code

ECError **AdsAdapterStart** (
DN_EC_T_ADS_ADAPTER_START_PARMS *poStartParams*,
uint *dwTimeout*
)

Starts the ADS Pass Through Server.

- #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

- #EC_E_NOMEMORY if the mailbox protocol queue of the slave is full
- #EC_E_ADS_IS_RUNNING if the ADS server is running

Param poStartParams

Pass through server start parameter

Param dwTimeout

Timeout

Return

ECError **AdsAdapterStop** (uint *dwTimeout*)

Stops the ADS Pass Through Server.

Param dwTimeout

Timeout

Return

EC_E_NOERROR or error code

ECError **GetDiagnosisImageSize** ()

Gets the diagnosis image size.

Return

Diagnosis image size

ECError **GetSrcMacAddress** (out *DN_ETHERNET_ADDRESS* *pMacSrc*)

Gets the source MAC address.

Param pMacSrc

6-byte buffer to write source MAC address to

Return

EC_E_NOERROR or error code

ECError **GetLicenseFingerprint** (byte *byMethod*, string *szLicenseFingerprint*)

Gets the fingerprint needed for license key generation.

Param byMethod

Method used to generate the fingerprint

Param szLicenseFingerprint

Buffer for fingerprint string. Size must be at least EC_LICENSE_FINGERPRINT_STRSIZE.

Return

EC_E_NOERROR or error code

ECError **SetLicenseKey** (string *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.

Param szLicenseKey

License key as zero terminated string with 26, 53 or 56 characters

Return

- 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_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

ECError GetVersion (out uint pdwVersion, out uint pdwVersionType)

Gets the version information.

Param pdwVersion

Pointer to EC_T_DWORD to carry out version number as a 32-bit value

Param pdwVersionType

Pointer to EC_T_DWORD to carry out version type. See EC_VERSION_TYPES "EC_VERSION_TYPE".

Return

- 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

ECError TraceDataConfig (ushort wTraceDataSize)

Configures a trace data buffer and enables it for transmission.

Must be called after initialization and before configuration.

If wTraceDataSize is too large, configuration will fail with return code #EC_E_XML_CYCCMDS_SIZEMISMATCH.

Param wTraceDataSize

Size of Trace Data in bytes

Return

- 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_NOTSUPPORTED if eCycFrameLayout_FIXED is configured

ECError TraceDataGetInfo (DN_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.

Param pTraceDataInfo

Information about trace data

Return

EC_E_NOERROR or error code

ECError FastModeInit ()

Return

ECError **FastSendAllCycFrames** ()

Return

ECError **FastProcessAllRxFrames** (out bool *pbAreAllCycFramesProcessed*)

Param pbAreAllCycFramesProcessed

Return

ECError **ReadSlaveIdentification** (

bool *bFixedAddressing*,

ushort *wSlaveAddress*,

ushort *wAdo*,

out ushort *pwValue*,

uint *dwTimeout*

)

Read identification value from a slave.

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

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wAdo

ADO used for identification command

Param pwValue

Pointer to Word value containing the Identification value

Param dwTimeout

Timeout [ms]

Return

- 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_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

```
ECError ReadSlaveIdentificationReq (
    uint dwClientId,
    uint dwTferId,
    bool bFixedAddressing,
    ushort wSlaveAddress,
    ushort wAdo,
    out ushort pwValue,
    uint 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.

Param dwClientId

Client ID returned by RegisterClient (0 if all registered clients shall be notified)

Param dwTferId

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_NOTIFY_DESC.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param wAdo

ADO used for identification command

Param pwValue

Pointer to Word value containing the Identification value, must be valid until the request complete.

Param dwTimeout

Timeout [ms]

Return

- 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

```
ECError SetSlaveDisabled (bool bFixedAddressing, ushort wSlaveAddress, 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.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param bDisabled

EC_TRUE: Disable slave, EC_FALSE: Enable slave

Return

EC_E_NOERROR or error code

```
ECError SetSlavesDisabled (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    DN_EC_T_SLAVE_SELECTION eSlaveSelection,
    bool bDisabled
)
```

Enable or disable a specific group of slaves.

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 change to PREOP. The information about the last requested state is lost and is set to PREOP too.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param eSlaveSelection

Slave selection criteria for following slaves

Param bDisabled

EC_TRUE: Disable slaves, EC_FALSE: Enable slaves

Return

EC_E_NOERROR or error code

```
ECError SetSlaveDisconnected (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    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.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param bDisconnected

EC_TRUE: Mark slave for disconnection, EC_FALSE: Mark slave for (re-)connection

Return

EC_E_NOERROR or error code

```
ECError SetSlavesDisconnected (
    bool bFixedAddressing,
    ushort wSlaveAddress,
    DN_EC_T_SLAVE_SELECTION eSlaveSelection,
    bool bDisconnected
)
```

Mark a specific group of slaves for connection or disconnection.

Param bFixedAddressing

EC_TRUE: use station address, EC_FALSE: use AutoInc address

Param wSlaveAddress

Slave address according bFixedAddressing

Param eSlaveSelection

Slave selection criteria

Param bDisconnected

EC_TRUE: mark slaves for disconnection, EC_FALSE: mark slaves for connection

Return

- 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

ECError **GetMemoryUsage** (**out** uint *pdwCurrentUsage*, **out** uint *pdwMaxUsage*)

Returns information about memory usage.

All calls to malloc/free and new/delete are monitored.

Param pdwCurrentUsage

Current memory usage in Bytes at the time where this function is called

Param pdwMaxUsage

Maximum memory usage in Bytes since initialization at the time where this function is called

Return

EC_E_NOERROR or error code

ECError **GetSlaveStatistics** (
 uint *dwSlaveId*,
out *DN_EC_T_SLVSTATISTICS_DESC* *pSlaveStatisticsDesc*
)

Get Slave's statistics counter.

Param dwSlaveId

Slave id

Param pSlaveStatisticsDesc

Pointer to structure EC_T_SLVSTATISTICS_DESC

Return

EC_E_NOERROR or error code

ECError **ClearSlaveStatistics** (uint *dwSlaveId*)

Clears all error registers of a slave.

Param dwSlaveId

Slave Id, INVALID_SLAVE_ID clears all slaves

Return

EC_E_NOERROR or error code

uint **GetMasterSyncUnitInfoNumOf** ()

Get number of Master Sync Units info entries.

Return

Number of Master Sync Units info entries

ECError GetMasterSyncUnitInfo (ushort *wMsuId*, out *DN_EC_T_MSU_INFO pMsuInfo*)

Get number of Master Sync Units info entries.

Param wMsuId

Param pMsuInfo

Return

Number of Master Sync Units info entries

ECError GetMasterDump (byte[] *pbyBuffer*, uint *dwBufferSize*, out uint *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.

Param pbyBuffer

Preallocated buffer to dump log data

Param dwBufferSize

Size of preallocated buffer

Param pdwDumpSize

Size of master dump

Return

- EC_E_NOERROR
- EC_E_NOMEMORY if buffer too small

ECError BadConnectionsDetect (bool *bRefreshSlaveStatistics*, uint *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.

Param bRefreshSlaveStatistics

Param dwTimeout

Timeout [ms]. not be EC_NOWAIT!

Return

EC_E_NOERROR or error code

ECError SelfTestScan (*DN_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.

Param pParms

Self-test scan parameters

Return

- 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_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

ECError LinkInputVarByName (string *szVarName*, *DN_EC_T_LINK_VAR_INFO pLinkVarInfo*)

Link application variable to input process image. Input process variable is given by its name.

Param szVarName

Name of variable

Param pLinkVarInfo

Information about linked variable

Return

EC_E_NOERROR or an error code

ECError LinkInputVarByObject (bool *bFixedAddressing*, ushort *wSlaveAddress*, ushort *wIndex*, ushort *wSubIndex*, *DN_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.

Param bFixedAddressing

Use station address if EC_TRUE. Otherwise use AutoInc address.

Param wSlaveAddress

Slave address according to bFixedAddressing

Param wIndex

Object index

Param wSubIndex

Object subindex

Param pLinkVarInfo

Information about linked variable

Return

EC_E_NOERROR or an error code

ECError **LinkOutputVarByName** (string *szVarName*, *DN_EC_T_LINK_VAR_INFO* *pLinkVarInfo*)

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

Param szVarName

Name of variable

Param pLinkVarInfo

Information about linked variable

Return

EC_E_NOERROR or an error code

ECError **LinkOutputVarByObject** (
 bool *bFixedAddressing*,
 ushort *wSlaveAddress*,
 ushort *wIndex*,
 ushort *wSubIndex*,
DN_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.

Param bFixedAddressing

Use station address if EC_TRUE. Otherwise use AutoInc address.

Param wSlaveAddress

Slave address according to bFixedAddressing

Param wIndex

Object index

Param wSubIndex

Object subindex

Param pLinkVarInfo

Information about linked variable

Return

EC_E_NOERROR or an error code

ECError **EoeSendFrame** (ushort *wCfgFixedAddress*, byte[] *pbyFrame*, uint *dwFrameLen*)

Send EoE frame (queued)

Param wCfgFixedAddress
Param pbyFrame
Param dwFrameLen
Return

_ #EC_E_NOERROR or error code

Properties

uint **s_EC_VERSION** { **get**; **set**; }

EC_VERSION.

summary>ATECAT_SIGNATURE

uint **s_ATECAT_SIGNATURE** { **get**; **set**; }

summary>INVALID_SLAVE_ID

```

uint s_INVALID_SLAVE_ID { get; set; }
    summary>MASTER_SLAVE_ID

uint s_MASTER_SLAVE_ID { get; set; }
    summary>MAX_NUMOF_MASTER_INSTANCES

uint s_MAX_NUMOF_MASTER_INSTANCES { get; set; }
    summary>MASTER_RED_SLAVE_ID

uint s_MASTER_RED_SLAVE_ID { get; set; }
    summary>EL9010_SLAVE_ID

uint s_EL9010_SLAVE_ID { get; set; }
    summary>FRAMELOSS_SLAVE_ID

uint s_FRAMELOSS_SLAVE_ID { get; set; }
    summary>JUNCTION_RED_FLAG

```

Events

```

event EcNotification OnEcNotification
    summary> Occurs when a application notification is raised.

event AppNotification OnAppNotification
    summary> Occurs when a RAS notification is raised.

event RasNotification OnRasNotification
    summary> Occurs when a translate notification is raised.

event TranslateNotification OnTranslateNotification
    summary> Occurs when a performance monitoring notification is raised.

event PerfNotification OnPerfNotification
    summary> Occurs when a debug message notification is raised.

```

Public Static Functions

```

bool IsRemoteServerUp (byte[] abyIpAddr, ushort wPort)
    Checks if remote server is up by sending a “ping”.

    Param abyIpAddr
        IPAddress

    Param wPort
        Port

    Return
        true, if server is up, else false if server is down

void CopyStream (Stream source, Stream target)
    Copy a stream.

    Param source
        Source.

```

Param target

Target.

ECError **CreateObjectFromBytes** (byte[] *bytes*, ref object *obj*)

Creates object from byte array.

Param bytes

Object data as byte array

Param obj

Object which should be filled

Return

Error code

ECError **GetSizeOfObject** (object *obj*, out uint *size*)

Get byte size of object (as required from CoeSdoUpload)

Param obj

Object

Param size

Byte size of object

Return

Error code

uint **GetBitLenOfDataType** (*DN_EC_T_DEFTYPE* *type*)

Get bit length of data type.

Param type

Data type

Return

Bit length

object **RunUnitTest** (string *cmd*, object *data*)

Run unit test (for internal use)

Param cmd

Command

Param data

Input data

Return

Output data

ECError **ConvValueToBytes** (*DN_EC_T_DEFTYPE* *type*, object *value*, out byte[] *bytes*)

Converts value to bytes.

Param type

Data type

Param value

Value

Param bytes

Bytes

Return

Error code

ECError **ConvValueFromBytes** (byte[] *bytes*, *DN_EC_T_DEFTYPE* *type*, out object *value*)

Converts value from bytes.

Param bytes

Bytes

Param type

Data type

Param value

Value

Return

Error code

ECError **ReadValueFromBytes** (

```
byte[] bytes,
uint bitOffset,
uint bitLength,
DN_EC_T_DEFTYPE type,
out object value
```

)

Read value from bytes.

Param bytes

Bytes

Param bitOffset

Bit offset

Param bitLength

Bit length

Param type

Data type

Param value

Value

Return

Error code

ECError **ReadValueFromAddress** (

```
IntPtr address,
uint bitOffset,
uint bitLength,
DN_EC_T_DEFTYPE type,
out object value
```

)

Read value from address.

Param address

Address

Param bitOffset

Bit offset

Param bitLength

Bit length

Param type

Data type

Param value

Value

Return

Error code

```
EError WriteValueToBytes (
    byte[] bytes,
    uint bitOffset,
    uint bitLength,
    DN_EC_T_DEFTYPE type,
    object value
)
```

Write value to bytes.

Param bytes

Bytes

Param bitOffset

Bit offset

Param bitLength

Bit length

Param type

Data type

Param value

Value

Return

Error code

```
EError WriteValueToAddress (
    IntPtr address,
    uint bitOffset,
    uint bitLength,
    DN_EC_T_DEFTYPE type,
    object value
)
```

Write value to address.

Param address

Address

Param bitOffset

Bit offset

Param bitLength

Bit length

Param type

Data type

Param value

Value

Return

Error code

```
object ConvValueFromString (DN_EC_T_DEFTYPE type, string value)
```

Converts value from string.

Param type

Type

Param value

Value

Return

Value as object

string **ConvValueToString** (*DN_EC_T_DEFTYPE* type, object value)

Converts value to string.

Param type

Type

Param value

Value

Return

Value as string

string **ConvStringFromArray** (byte[] data)

Converts byte array to string (e.g. 01 02)

Param data

Data

Return

Data as string

byte[] **ConvStringToArray** (string data)

Converts byte array from string (e.g. 01 02)

Param data

Data

Return

Data as byte array

void **ReadPdbitsFromAddress** (IntPtr address, uint bitOffset, uint bitLength, out byte[] pdBits)

Reads process data bits from address.

Param address

Address

Param bitOffset

Bit offset

Param bitLength

Bit length

Param pdBits

[out] Process data bits

void **WritePdbitsToAddress** (IntPtr address, uint bitOffset, uint bitLength, byte[] pdBits)

Writes process data bits to address.

Param address

Address

Param bitOffset

Bit offset

Param bitLength

Bit length

Param pdBits

Process data bits

void **ReadPdbitsFromBytes** (byte[] bytes, uint bitOffset, uint bitLength, out byte[] pdBits)

Reads process data bits from bytes.

Param bytes

Bytes

Param bitOffset

Bit offset

Param bitLength

Bit length

Param pdBits

[out] Process data bits

void **WritePdBitsToBytes** (byte[] *bytes*, uint *bitOffset*, uint *bitLength*, byte[] *pdBits*)

Writes process data bits to bytes.

Param bytes

Bytes

Param bitOffset

Bit offset

Param bitLength

Bit length

Param pdBits

Process data bits

void **ReadPdBytesFromAddress** (IntPtr *address*, uint *offset*, uint *length*, out byte[] *pdBytes*)

Reads process data bytes from address.

Param address

Address

Param offset

Offset

Param length

Length

Param pdBytes

[out] Process data bytes

void **WritePdBytesToAddress** (IntPtr *address*, uint *offset*, byte[] *pdBytes*)

Writes process data bytes to address.

Param address

Address

Param offset

Offset

Param pdBytes

Process data bytes

void **ReadPdBytesFromBytes** (byte[] *bytes*, uint *offset*, uint *length*, out byte[] *pdBytes*)

Reads process data bytes from bytes.

Param bytes

Bytes

Param offset

Offset

Param length

Length

Param pdBytes

[out] Process data bytes

void **WritePdBytesToBytes** (byte[] *bytes*, uint *offset*, byte[] *pdBytes*)

Writes process data bytes to bytes.

Param bytes

Bytes

Param offset

Offset

Param pdBytes

Process data bytes

Public Static Attributes

bool **EnableExceptionHandling** = false

Enables exception handling (functions will throw an exception instead of returning an error code)

3.2 Types

class EcWrapperDotNet.DN_EC_T_INIT_PARMS

Abstract initialization parameters.

Subclassed by EcWrapperDotNet.DN_EC_T_INIT_PARMS_DAQ, EcWrapperDotNet.DN_EC_T_INIT_PARMS_DAQ_READER, EcWrapperDotNet.DN_EC_T_INIT_PARMS_MASTER, EcWrapperDotNet.DN_EC_T_INIT_PARMS_MASTER_RAS_SERVER, EcWrapperDotNet.DN_EC_T_INIT_PARMS_MBXGATEWAY_CLIENT, EcWrapperDotNet.DN_EC_T_INIT_PARMS_MBXGATEWAY_SERVER, EcWrapperDotNet.DN_EC_T_INIT_PARMS_MONITOR, EcWrapperDotNet.DN_EC_T_INIT_PARMS_MONITOR_RAS_SERVER, EcWrapperDotNet.DN_EC_T_INIT_PARMS_RAS_CLIENT, EcWrapperDotNet.DN_EC_T_INIT_PARMS_SIMULATOR, EcWrapperDotNet.DN_EC_T_INIT_PARMS_SIMULATOR_RAS_SERVER

class EcWrapperDotNet.DN_EC_T_INIT_MASTER_PARMS

DN_EC_T_INIT_MASTER_PARMS.

Public Members

DN_EC_T_LINK_PARMS oLinkParms

oLinkParms

DN_EC_T_LINK_PARMS oLinkParmsRed

oLinkParmsRed

uint **dwBusCycleTimeNsec**

[in] Bus cycle time [ns]

uint **dwMaxBusSlaves**

[in] Maximum pre-allocated bus slave objects (default:
256)

uint **dwMaxAcycFramesQueued**

[in] Maximum queued acyclic frames

uint **dwAdditionalEoEEndpoints**

[in] Additional EoE endpoints

uint **dwMaxAcycBytesPerCycle**

[in] Maximum bytes sent during eUsrJob_SendAcycFrames per cycle (default:
4096)

uint **dwMaxAcycFramesPerCycle**

[in] Maximum frames sent during eUsrJob_SendAcycFrames per cycle (default:
32)

uint **dwMaxAcycCmdsPerCycle**

[in] Maximum commands sent during eUsrJob_SendAcycFrames per cycle (default:
124)

uint **dwMaxSlavesProcessedPerCycle**

[in] Maximum slave-related state machine calls per cycle (default = all)

uint **dwEcatCmdMaxRetries**

[in] Maximum retries to send pending EtherCAT command frames (default:
3)

uint **dwEcatCmdTimeout**

[in] Timeout [ms] to send pending EtherCAT command frames (default:
3 * bus cycle time, at least 2 ms)

DN_EC_LOG_LEVEL **dwLogLevel**

dwLogLevel

DN_EC_T_MASTER_RED_PARMS **MasterRedParms**

[in] Master Redundancy parameters

uint **dwMaxS2SMbxSize**

[in] Size of the queued S2S mailbox in bytes (default:
0)

uint **dwMaxQueuedS2SMbxTfer**

[in] S2S Fifo number of entries (default:
0)

ushort **wMaxSlavesProcessedPerBusScanStep**

[in] Maximum slave-related calls per cycle during bus scans (default = all)

bool **bApiLockByApp**

[in] Lock pending API against emDeinitMaster(). EC_FALSE (default): locked internally. EC_TRUE:
application is responsible for locking.

DN_EC_T_PERF_MEAS_INTERNAL_PARMS **PerfMeasInternalParms**

[in] Internal performance measurement parameters

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.

```
class EcWrapperDotNet.DN_EC_T_LINK_PARMS
```

```
class EcWrapperDotNet.DN_EC_T_MASTER_RED_PARMS
```

Master Redundancy initialization parameters, see `ecatInitMaster`.

Public Members

bool **bEnabled**

[in] Set to EC_TRUE if using Master Redundancy

ushort **wMasterPdOutSize**

[in] ACTIVE to INACTIVE Master Process Data (in bytes)

ushort **wMasterPdInSize**

[in] INACTIVE to ACTIVE Master Process Data (in bytes)

uint **dwMaxAcycFramesPerCycle**

[in] Maximum acyclic Master Red frames sent per cycle

bool **bUpdateSlavePdOut**

[in] Set to EC_TRUE to update Slave OUTPUT Process Data Image at INACTIVE Master (from CSF)

bool **bUpdateSlavePdIn**

[in] Set to EC_TRUE to update Slave INPUT Process Data Image at INACTIVE Master (from CMF)

bool **bSwitchTopology**

[in] Set to EC_TRUE to enable switch topology

```
class EcWrapperDotNet.DN_EC_T_PERF_MEAS_INTERNAL_PARMS
```

DN_EC_T_PERF_MEAS_INTERNAL_PARMS.

Public Members

bool **bEnabled**

[in] Enable/disable internal performance counters

[*DN_EC_T_PERF_MEAS_COUNTER_PARMS*](#) **CounterParms**

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

[*DN_EC_T_PERF_MEAS_HISTOGRAM_PARMS*](#) **HistogramParms**

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

```
class EcWrapperDotNet.DN_EC_T_PERF_MEAS_COUNTER_PARMS
```

DN_EC_T_PERF_MEAS_COUNTER_PARMS.

Public Members

ulong **qwFrequency**

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

```
class EcWrapperDotNet.DN_EC_T_PERF_MEAS_HISTOGRAM_PARMS
    DN_EC_T_PERF_MEAS_HISTOGRAM_PARMS.
```

Public Members

uint **dwBinCount**

[in] Amount of bins to use for the histogram

ulong **qwMinTicks**

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

ulong **qwMaxTicks**

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

```
class EcWrapperDotNet.DN_EC_T_LOG_PARMS
    DN_EC_T_LOG_PARMS.
```

Public Members

uint **dwLogLevel**

[in] Log level. See EC_LOG_LEVEL...

```
enum EcWrapperDotNet.DN_EC_LOG_LEVEL
    DN_EC_LOG_LEVEL.
```

Values:

SILENT

SILENT.

ANY

ANY.

CRITICAL

CRITICAL.

ERROR

ERROR.

WARNING

WARNING.

INFO

INFO.

INFO_API

INFO_API.

VERBOSE

VERBOSE.

VERBOSE_CYC

VERBOSE_CYC.

UNDEFINED

UNDEFINED.

class EcWrapperDotNet.DN_EC_T_ADS_ADAPTER_START_PARMS

DN_EC_T_ADS_ADAPTER_START_PARMS.

Public Members

ulong **cpuAffinityMask**

cpuAffinityMask

uint **dwThreadPriority**

dwThreadPriority

uint **dwStackSize**

dwStackSize

DN_EC_T_AOE_NETID **targetNetID**

targetNetID

ushort **targetPort**

targetPort

class EcWrapperDotNet.DN_EC_T_AOE_NETID

AoE NetID.

Public Members

byte[] **aby**

AoE net id.

class EcWrapperDotNet.DN_EC_T_BUS_SLAVE_INFO

DN_EC_T_BUS_SLAVE_INFO.

Public Members

uint **dwSlaveId**

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

uint[] **adwPortSlaveIds**

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

ushort **wPortState**

wPortState

ushort **wAutoIncAddress**

[out] The slave's auto increment address

bool **bDcSupport**

[out] Slave supports DC (Bus Topology Scan)

bool **bDc64Support**

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

uint **dwVendorId**

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

uint **dwProductCode**

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

uint **dwRevisionNumber**

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

uint **dwSerialNumber**

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

byte **byESCType**

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

byte **byESCRevision**

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

ushort **wESCBuild**

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

byte **byPortDescriptor**

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

ushort **wFeaturesSupported**

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

ushort **wStationAddress**

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

ushort **wAliasAddress**

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

ushort **wAlStatus**

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

ushort **wAlStatusCode**

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

uint **dwSystemTimeDifference**

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

ushort **wMbxSupportedProtocols**

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

ushort **wDlStatus**

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

ushort **wPrevPort**

[out] Connected port of the previous slave

ushort **wIdentifyData**

[out] Last read identification value see EC_T_CFG_SLAVE_INFO.wIdentifyAdo

bool **bLineCrossed**

[out] Line crossed was detected at this slave

uint **dwSlaveDelay**

[out] Delay behind slave [ns].

uint **dwPropagDelay**

[out] Propagation delay [ns]. ESC register 0x0928.

bool **bIsRefClock**

[out] Slave is reference clock

bool **bIsDeviceEmulation**

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

ushort **wLineCrossedFlags**

[out] Combination of Line crossed flags

uint **dwCyclicWkcErrorCnt**

[out] Counter for Cyclic WC Error

uint **dwSlaveAbsentCnt**

[out] Counter for Absent/Not Present Slaves

uint **dwUnexpectedStateCnt**

[out] Counter for Abnormal State Change

class EcWrapperDotNet.DN_EC_T_CFG_SLAVE_INFO

ecatGetCfgSlaveInfo

Public Members

uint **dwSlaveId**

[out] Slave's ID to bind bus slave and config slave information

string **abyDeviceName**
[out] Slave's configured name (80 Byte) (from ENI file)

uint **dwHCGroupIdx**
[out] Index of Hot Connect group, 0 for mandatory

bool **bIsPresent**
[out] Slave present on bus

bool **bIsHCGroupPresent**
[out] Slave's Hot Connect group present on bus

uint **dwVendorId**
[out] Vendor identification (from ENI file)

uint **dwProductCode**
[out] Product code (from ENI file)

uint **dwRevisionNumber**
[out] Revision number (from ENI file)

uint **dwSerialNumber**
[out] Serial number (from ENI file)

ushort **wStationAddress**
[out] Slave's configured station address (from ENI file)

ushort **wAutoIncAddress**
[out] Slave's auto increment address (may differ from ENI file)

uint **dwPdOffsIn**
[out] Process input data bit offset (from ENI file)

uint **dwPdSizeIn**
[out] Process input data bit size (from ENI file)

uint **dwPdOffsOut**
[out] Process output data bit offset (from ENI file)

uint **dwPdSizeOut**
[out] Process output data bit size (from ENI file)

uint **dwPdOffsIn2**
[out] 2nd sync unit process input data bit offset (from ENI file)

uint **dwPdSizeIn2**
[out] 2nd sync unit process input data bit size (from ENI file)

uint **dwPdOffsOut2**

[out] 2nd sync unit process output data bit offset (from ENI file)

uint **dwPdSizeOut2**

[out] 2nd sync unit process output data bit size (from ENI file)

uint **dwPdOffsIn3**

[out] 3rd sync unit process input data bit offset (from ENI file)

uint **dwPdSizeIn3**

[out] 3rd sync unit process input data bit size (from ENI file)

uint **dwPdOffsOut3**

[out] 3rd sync unit process output data bit offset (from ENI file)

uint **dwPdSizeOut3**

[out] 3rd sync unit process output data bit size (from ENI file)

uint **dwPdOffsIn4**

[out] 4th sync unit process input data bit offset (from ENI file)

uint **dwPdSizeIn4**

[out] 4th sync unit process input data bit size (from ENI file)

uint **dwPdOffsOut4**

[out] 4th sync unit process output data bit offset (from ENI file)

uint **dwPdSizeOut4**

[out] 4th sync unit process output data bit size (from ENI file)

uint **dwMbxSupportedProtocols**

[out] Mailbox protocols supported by the slave (from ENI file). Combination of Supported mailbox protocols flags.

uint **dwMbxOutSize**

[out] Mailbox output byte size (from ENI file)

uint **dwMbxInSize**

[out] Mailbox input byte size (from ENI file)

uint **dwMbxOutSize2**

[out] Bootstrap mailbox output byte size (from ENI file)

uint **dwMbxInSize2**

[out] Bootstrap mailbox input byte size (from ENI file)

bool **bDcSupport**

[out] Slave supports DC (from ENI file)

ushort **wNumProcessVarsInp**

[out] Number of input process data variables (from ENI file)

ushort **wNumProcessVarsOutp**

[out] Number of output process data variables (from ENI file)

ushort **wPrevStationAddress**

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

ushort **wPrevPort**

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

ushort **wIdentifyAdo**

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

ushort **wIdentifyData**

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

byte **byPortDescriptor**

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

ushort[] **wWkcStateDiagOffsIn**

[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.

ushort[] **wWkcStateDiagOffsOut**

[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.

ushort[] **awMasterSyncUnitIn**

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

ushort[] **awMasterSyncUnitOut**

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

bool **bDisabled**

[out] Slave disabled by API SetSlaveDisabled / SetSlavesDisabled.

bool **bDisconnected**

[out] Slave disconnected by API SetSlaveDisconnected / SetSlavesDisconnected.

bool **bExtended**

[out] Slave generated by API ConfigExtend

bool **bDcReferenceClock**

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

bool **bDcPotentialRefClock**

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

uint **dwDcCycleTime0**

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

uint **dwDcCycleTime1**

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

int **nDcShiftTime**

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

string **szProductRevision**

[out] Slave's ProductRevision (ENI:Slave/Info/ProductRevision), e.g. "EK1100-0000-0018". Empty string if not present in ENI.

class EcWrapperDotNet.DN_EC_T_CFG_SLAVE_SM_INFO
DN_EC_T_CFG_SLAVE_SM_INFO.

Public Members

uint **dwSlaveId**

[out] Slave ID

uint **dwSmInfoNumOf**

[out] Number of available sync managers

DN_EC_T_CFG_SLAVE_SM_ENTRY[] **aoSmInfos**

[out] Sync managers info

class EcWrapperDotNet.DN_EC_T_CFG_SLAVE_SM_ENTRY
ecatGetCfgSlaveSmInfo

Public Members

ushort **wPhysAddr**

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

ushort **wLength**

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

byte **byOpMode**

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

byte **byDirection**

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

uint **dwPdbitOffs**

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

uint **dwPdBitSize**

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

ushort **wWkcStateDiagBitOffs**

[out] Offset of WkcState bit in diagnosis image

ushort **wMasterSyncUnit**

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

class EcWrapperDotNet.DN_EC_T_CFG_SLAVE_EOE_INFO

ecatGetCfgSlaveEoeInfo, ENI: Mailbox/EoE/InitCmds/InitCmd/Data

Public Members

uint **dwSlaveId**

[out] Slave ID

bool **bMacAddr**

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

byte[] **abyMacAddr**

[out] MAC address

bool **bIpAddr**

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

DN_EC_T_IPADDR **oIpAddr**

[out] IP address

bool **bSubnetMask**

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

DN_EC_T_IPADDR **oSubnetMask**

[out] Subnet mask

bool **bDefaultGateway**

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

DN_EC_T_IPADDR **oDefaultGateway**

[out] Default gateway

bool **bDnsServer**

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

DN_EC_T_IPADDR **oDnsServer**

[out] DNS server

bool **bDnsName**

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

string **szDnsName**
[out] DNS name

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

uint[] **dwRes**
dwRes

class EcWrapperDotNet.DN_EC_T_IPADDR
DN_EC_T_IPADDR.

Public Members

DN_EC_T_INNER_IPADDR **sAddr**
sAddr

class EcWrapperDotNet.DN_EC_T_INNER_IPADDR
DN_EC_T_INNER_IPADDR.

Public Members

byte[] **by**
IPv4 address (endianness independent)

enum EcWrapperDotNet.DN_EC_T_CNF_TYPE
DN_EC_T_CNF_TYPE.

Values:

Unknown
Unknown.

Filename
pbyCnfData: ENI filename to read

Data
pbyCnfData: ENI data

Datadiag
pbyCnfData: ENI data for diagnosis

GenPreopENI
Generate ENI based on bus-scan result to get into PREOP state.

GenPreopENIWithCRC
Same as eCnfType_GenPreopENI with CRC protection.

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”.

None

Reset configuration.

ConfigData

pbxCnfData: Binary structured configuration

GenOpENINoStrings

Generate ENI based on bus-scan result to get into OP state, does not read strings from EEPROM.

FileByApp

File access provided by user application, See EC_T_CNF_FILEBYAPP_DESC.

GenEBI

Generate EBI based on bus-scan result.

BCppDummy

BCppDummy.

```
class EcWrapperDotNet.DN_EC_T_COE_ENTRYDESC
```

Public Members

ushort **wObjIndex**

object index

byte **byObjSubIndex**

object sub-index

byte **byValueInfo**

value information, determines which of the following elements are valid

ushort **wDataType**

data type

ushort **wBitLen**

bit length of the object

byte **byObjAccess**

object access

bool **bRxPdoMapping**

EC_TRUE if it is PDO mappable

bool **bTxPdoMapping**
EC_TRUE if PDO mappable object can be changed

bool **bObCanBeUsedForBackup**
Parameter may be back upped

bool **bObCanBeUsedForSettings**
Parameter may be used for Settings

ushort **dwUnitType**
length of the data

string **szObName**
length of the data

class EcWrapperDotNet.DN_EC_T_COE_OBDESC

Public Members

ushort **wObIndex**
object index

ushort **wDataType**
see DEFTYPE_..., refer to “Extended Data Type Area” in ETG.1000.6

byte **byObjCode**
see ECAT_COE_INFO_OBJCODE_..., defined in CiA DS 301, “Object code usage”

byte **byObjCategory**
category: 0=optional, 1=mandatory

byte **byMaxNumSubIndex**
maximum number of sub-indexes

string **pchObName**
object name (not a NULL terminated string!)

enum EcWrapperDotNet.DN_EC_T_COE_ODLIST_TYPE

EtherCAT CoE OD list type values.

Values:

Lengths
Lengths of each list type.

ALL
List contains all objects.

RxPdoMap

List with PDO mappable objects.

TxPdoMap

List with objects that can be changed.

StoredFRepl

Only stored for a device replacement objects.

StartupParm

Only startup parameter objects.

BCppDummy

BCppDummy.

```
class EcWrapperDotNet.DN_EC_T_CYC_CONFIG_DESC
```

Descriptor for EC_IOCTL_GET_CYCLIC_CONFIG_INFO.

Public Members

```
uint dwTaskId
```

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

```
uint dwPriority
```

[out] Priority of selected cyclic entry

```
uint dwCycleTime
```

[out] Cycle time of selected cyclic entry

```
class EcWrapperDotNet.DN_EC_T_DCM_CONFIG
```

DN_EC_T_DCM_CONFIG.

Public Members

```
DN_EC_T_DCM_MODE eMode
```

[in] DCM mode

```
DN_EC_T_DCM_CONFIG_BUSSHIFT BusShift
```

[in] BusShift configuration. Valid if eMode is set to eDcmMode_BusShift.

```
DN_EC_T_DCM_CONFIG_MASTERSHIFT MasterShift
```

[in] MasterShift configuration. Valid if eMode is set to eDcmMode_MasterShift.

```
DN_EC_T_DCM_CONFIG_LINKLAYERREFCLOCK LinkLayerRefClock
```

[in] LinkLayerRefClock configuration. Valid if eMode is set to eDcmMode_LinkLayerRefClock.

```
DN_EC_T_DCM_CONFIG_MASTERREFCLOCK MasterRefClock
```

[in] MasterRefClock configuration. Valid if eMode is set to eDcmMode_MasterRefClock.

DN_EC_T_DCM_CONFIG_DCX **Dcx**

[in] DCX configuration. Valid if eMode is set to eDcmMode_Dcx.

enum EcWrapperDotNet.DN_EC_T_DCM_MODE

DN_EC_T_DCM_MODE.

Values:

Off

DCM disabled.

BusShift

DCM BusShift mode.

MasterShift

DCM MasterShift mode.

LinkLayerRefClock

DCM LinkLayer Ref Clock mode.

MasterRefClock

DCM Master Ref Clock mode.

Dcx

DCM DCX External synchronization mode.

MasterShiftByApp

DCM MasterShift controlled by application mode.

BCppDummy

BCppDummy.

class EcWrapperDotNet.DN_EC_T_DCM_CONFIG_BUSSHIFT

DN_EC_T_DCM_CONFIG_BUSSHIFT.

Public Members

int nCtl1SetVal

[in] Controller set value [ns]. This is the time distance between the cyclic frame send time and the DC base on bus (SYNC0 if shift is zero).

int nCtl1Gain

[in] Proportional gain in ppt (part per thousand). Default is value 2. A value of 0 sets the default value.

int nCtl1DriftErrorGain

[in] Multiplier for drift error. Default value is 3. A value of 0 sets the default value.

int nMaxValidVal

[in] Error inputs above this value are considered invalid. If error input prediction is valid then the difference between the error input and the expected value is taken. Default value is 3000. A value of 0 sets the default value.

bool **bLogEnabled**

[in] If set to EC_TRUE, logging information are generated and can be got by calling emDcmGetLog()

uint **dwInSyncLimit**

[in] Limit [ns] for InSync monitoring. Default value is 20% of the cycle time. A value of 0 sets the default value.

uint **dwInSyncSettleTime**

[in] Settle time [ms] for InSync monitoring. Default value is 1500ms. A value of 0 sets the default value.

bool **bCtlOff**

[in] If set to EC_TRUE, control loop is disabled. Combined with bLogEnabled, it makes possible to analyze the natural drift between the stack cycle and the reference clock.

bool **bUseDcLoopCtlStdValues**

[in] If set to EC_TRUE, the values of ESC DC time loop control register 0x930 and 0x934 are not changed by master. This could increase the time it takes to get the InSync. Use only if there are problems with the reference clock to get InSync.

uint **dwInSyncStartDelayCycle**

[in] Delay time [ms] before InSync monitoring start

IntPtr **pGetTimeElapsedSinceCycleStartContext**

[in] Optional context for the pfnGetTimeElapsedSinceCycleStart function, will be passed as first parameter to this function/static method

class EcWrapperDotNet.DN_EC_T_DCM_CONFIG_MASTERSHIFT

DN_EC_T_DCM_CONFIG_MASTERSHIFT.

Public Members

int **nCtlSetVal**

[in] Controller set value [ns]. This is the time distance between the cyclic frame send time and the DC base on bus (SYNC0 if shift is zero).

int **nCtlGain**

[in] Proportional gain in ppt (part per thousand). Default is value 2. A value of 0 sets the default value.

int **nCtlDriftErrorGain**

[in] Multiplier for drift error. Default value is 3. A value of 0 sets the default value.

int **nMaxValidVal**

[in] Error inputs above this value are considered invalid. If error input prediction is valid then the difference between the error input and the expected value is taken. Default value is 3000. A value of 0 sets the default value.

bool **bLogEnabled**

[in] If set to EC_TRUE, logging information are generated and can be got by calling emDcmGetLog()

uint **dwInSyncLimit**

[in] Limit [ns] for InSync monitoring. Default value is 20% of the cycle time. A value of 0 sets the default value.

uint **dwInSyncSettleTime**

[in] Settle time [ms] for InSync monitoring. Default value is 1500ms. A value of 0 sets the default value.

bool **bCtlOff**

[in] If set to EC_TRUE, control loop is disabled. Combined with bLogEnabled, it makes possible to analyze the natural drift between the stack cycle and the reference clock. Also it provides reading of current adjustment value using emDcmGetAdjust function.

uint **dwInSyncStartDelayCycle**

[in] Delay time [ms] before InSync monitoring start

IntPtr **pGetTimeElapsedSinceCycleStartContext**

[in] Optional context for the pfnGetTimeElapsedSinceCycleStart function, will be passed as first parameter to this function/static method

IntPtr **pAdjustCycleTimeContext**

[in] Optional context for the pfnAdjustCycleTime function, will be passed as first parameter to this function/static method

class EcWrapperDotNet.DN_EC_T_DCM_CONFIG_LINKLAYERREFCLOCK
DN_EC_T_DCM_CONFIG_LINKLAYERREFCLOCK.

Public Members

int **nCtlSetVal**

[in] Controller set value [ns]. This is the time distance between the cyclic frame send time and the DC base on bus (SYNC0 if shift is zero).

bool **bLogEnabled**

[in] If set to EC_TRUE, logging information are generated and can be got by calling emDcmGetLog()

uint **dwInSyncLimit**

[in] Limit [ns] for InSync monitoring. Default value is 20% of the cycle time. A value of 0 sets the default value.

uint **dwInSyncSettleTime**

[in] Settle time [ms] for InSync monitoring. Default value is 1500ms. A value of 0 sets the default value.

DN_EC_T_DC_STARTTIME_CB_DESC DcStartTimeCallbackDesc

[in] If not null, DC start time calculated by application, otherwise by master. See also EC_T_DC_STARTTIME_CB_DESC. Shift value configured in ENI will still be applied.

class EcWrapperDotNet.DN_EC_T_DC_STARTTIME_CB_DESC
DN_EC_T_DC_STARTTIME_CB_DESC.

Public Members

IntPtr **pvContext**

[in] Arbitrarily application-defined parameter passed to pfnCallback

```
class EcWrapperDotNet.DN_EC_T_DCM_CONFIG_MASTERREFCLOCK
    DN_EC_T_DCM_CONFIG_MASTERREFCLOCK.
```

Public Members

int **nCtlSetVal**

[in] Controller set value [ns]. This is the time distance between the cyclic frame send time and the DC base on bus (SYNC0 if shift is zero).

bool **bLogEnabled**

[in] If set to EC_TRUE, logging information are generated and can be got by calling emDcmGetLog()

uint **dwInSyncLimit**

[in] Limit [ns] for InSync monitoring. Default value is 20% of the cycle time. A value of 0 sets the default value.

uint **dwInSyncSettleTime**

[in] Settle time [ms] for InSync monitoring. Default value is 1500ms. A value of 0 sets the default value.

IntPtr **pGetHostTimeContext**

[in] Optional context for the pfnGetHostTime function, will be passed as first parameter to this function/static method

DN_EC_T_DC_STARTTIME_CB_DESC **DcStartTimeCallbackDesc**

[in] If not null, DC start time calculated by application, otherwise by master. See also EC_T_DC_STARTTIME_CB_DESC. Shift value configured in ENI will still be applied.

```
class EcWrapperDotNet.DN_EC_T_DCM_CONFIG_DCX
    DN_EC_T_DCM_CONFIG_DCX.
```

Public Members

DN_EC_T_DCM_CONFIG_MASTERSHIFT **MasterShift**

[in] DCM MasterShift configuration

int **nCtlSetVal**

[in] Controller set value [ns]. This is the time distance between the cyclic frame send time and the DC base on bus (SYNC0 if shift is zero).

int **nCtlGain**

[in] Proportional gain in ppt (part per thousand). Default is value 2. A value of 0 sets the default value.

int **nCtlDriftErrorGain**

[in] Multiplier for drift error. Default value is 3. A value of 0 sets the default value.

int **nMaxValidVal**

[in] Error inputs above this value are considered invalid. If error input prediction is valid then the difference between the error input and the expected value is taken. Default value is 3000. A value of 0 sets the default value.

bool **bLogEnabled**

[in] If set to EC_TRUE, logging information are generated and can be got by calling emDcmGetLog()

uint **dwInSyncLimit**

[in] Limit [ns] for InSync monitoring. Default value is 20% of the cycle time. A value of 0 sets the default value.

uint **dwInSyncSettleTime**

[in] Settle time [ms] for InSync monitoring. Default value is 1500ms. A value of 0 sets the default value.

bool **bCtlOff**

[in] If set to EC_TRUE, control loop is disabled. Combined with bLogEnabled, it makes possible to analyze the natural drift between the stack cycle and the reference clock. Also it provides reading of current adjustment value using emDcmGetAdjust function.

ushort **wExtClockFixedAddr**

[in] Fixed address of external clock slave (publishing PDO 0x10F4) (optional if ENI is generated by EcEngineer)

uint **dwExtClockTimeout**

[in] Wait timeout for external clock slave

uint **dwInSyncStartDelayCycle**

[in] Delay time [ms] before InSync monitoring start

uint **dwMaxErrCompensableOnExtClockReconnect**

[in] Maximum error in nanoseconds that should be compensated after a reconnect of the external clock device. Synchronization restart if error exceeds this limit.

class EcWrapperDotNet.DN_EC_T_DC_CONFIGURE

DN_EC_T_DC_CONFIGURE.

Public Members

uint **dwClntId**

[in] Reserved

uint **dwTimeout**

[in] Timeout [ms] for the DC initialization in which time offsets and propagation delays are evaluated (default:

12 sec)

uint **dwDevLimit**

dwDevLimit

uint **dwSettleTime**

dwSettleTime

uint **dwTotalBurstLength**

[in] Overall amount of burst frames sent (default:
10000)

uint **dwBurstBulk**

[in] Amount of burst frames per cycle during initialization burst (default:
12)

bool **bBulkInLinkLayer**

bBulkInLinkLayer

bool **bAcycDistributionDisabled**

[in] If EC_TRUE, acyclic distribution is disabled

uint **dwDcStartTimeGrid**

[in] Time grid [ns] to align DC start time. With the help of the grid, several EtherCAT networks can be synchronized without a random shift value between the SYNC signals.

bool **bDcInitBeforeSlaveStateChange**

[in] If EC_TRUE, DC is initialized before slaves state change to PREOP

enum **EcWrapperDotNet.DN_EC_T_DEFTYPE**

DN_EC_T_DEFTYPE.

Values:

NULL

NULL.

BOOLEAN

BOOLEAN.

INTEGER8

INTEGER8.

INTEGER16

INTEGER16.

INTEGER32

INTEGER32.

UNSIGNED8

UNSIGNED8.

UNSIGNED16

UNSIGNED16.

UNSIGNED32

UNSIGNED32.

REAL32

REAL32.

VISIBLESTRING

VISIBLESTRING.

OCTETSTRING

OCTETSTRING.

UNICODESTRING

UNICODESTRING.

TIMEOFDAY

TIMEOFDAY.

TIMEDIFFERENCE

TIMEDIFFERENCE.

DOMAIN

DOMAIN.

INTEGER24

INTEGER24.

REAL64

REAL64.

INTEGER40

INTEGER40.

INTEGER48

INTEGER48.

INTEGER56

INTEGER56.

INTEGER64

INTEGER64.

UNSIGNED24

UNSIGNED24.

UNSIGNED40

UNSIGNED40.

UNSIGNED48

UNSIGNED48.

UNSIGNED56

UNSIGNED56.

UNSIGNED64

UNSIGNED64.

GUID

GUID.

BYTE

BYTE.

WORD

WORD.

DWORD

DWORD.

PDOMAPPING

PDOMAPPING.

IDENTITY

IDENTITY.

COMMAND

COMMAND.

PDOCOMPAR

PDOCOMPAR.

ENUM

ENUM.

SMPAR

SMPAR.

RECORD

RECORD.

BACKUP_PARAMETER

BACKUP_PARAMETER.

MODULAR_DEVICE_PROFILE

MODULAR_DEVICE_PROFILE.

BITARR8

BITARR8.

BITARR16

BITARR16.

BITARR32

BITARR32.

BIT1

BIT1.

BIT2

BIT2.

BIT3

BIT3.

BIT4

BIT4.

BIT5

BIT5.

BIT6

BIT6.

BIT7

BIT7.

BIT8

BIT8.

BIT9

BIT9.

BIT10

BIT10.

BIT11

BIT11.

BIT12

BIT12.

BIT13

BIT13.

BIT14

BIT14.

BIT15

BIT15.

BIT16

BIT16.

ARRAY_OF_BYTE

ARRAY_OF_BYTE.

ARRAY_OF_UINT

ARRAY_OF_UINT.

ARRAY_OF_INT

ARRAY_OF_INT.

ARRAY_OF_SINT

ARRAY_OF_SINT.

ARRAY_OF_DINT

ARRAY_OF_DINT.

ARRAY_OF_UDINT

ARRAY_OF_UDINT.

ERROR_SETTING

ERROR_SETTING.

HISTORY

HISTORY.

DIAGNOSIS_OBJECT

DIAGNOSIS_OBJECT.

EXTERNAL_SYNC_STATUS

EXTERNAL_SYNC_STATUS.

EXTERNAL_SYNC_SETTINGS

EXTERNAL_SYNC_SETTINGS.

FSOEFRAME

FSOEFRAME.

FSOECOMMPAR

FSOECOMMPAR.

class EcWrapperDotNet.DN_EC_T_GET_SLAVE_INFO

ecatGetSlaveInfo

Public Members

uint **dwScanBusStatus**

Status during last Bus Scan.

uint **dwVendorId**

Vendor Identification.

uint **dwProductCode**

Product Code.

uint **dwRevisionNumber**

Revision Number.

uint **dwSerialNumber**

Serial Number.

ushort **wPortState**

[out] port link state (SB Instance)

bool **bDcSupport**

[out] Slave does support DC

bool **bDc64Support**

[out] Slave does support 64Bit DC

ushort **wAliasAddress**

[out] Slave alias address

ushort **wPhysAddress**

[out] Slave station address

uint **dwPdOffsIn**

[out] Process data offset of Input Data (in bits)

uint **dwPdSizeIn**

[out] Process data size of Input Data (in bits)

uint **dwPdOffsOut**

[out] Process data offset of Output Data (in bits)

uint **dwPdSizeOut**

[out] Process data size of Output Data

uint **dwPdOffsIn2**

[out] Process data offset of Input data (in bits)

uint **dwPdSizeIn2**

[out] Process data size of Input Data (in bits)

uint **dwPdOffsOut2**
[out] Process data offset of Output Data (in bits)

uint **dwPdSizeOut2**
[out] Process data size of Output Data

uint **dwPdOffsIn3**
[out] Process data offset of Input Data (in bits)

uint **dwPdSizeIn3**
[out] Process data size of Input Data (in bits)

uint **dwPdOffsOut3**
[out] Process data offset of Output Data (in bits)

uint **dwPdSizeOut3**
[out] Process data size of Output Data

uint **dwPdOffsIn4**
[out] Process data offset of Input Data (in bits)

uint **dwPdSizeIn4**
[out] Process data size of Input Data (in bits)

uint **dwPdOffsOut4**
[out] Process data offset of Output Data (in bits)

uint **dwPdSizeOut4**
[out] Process data size of Output Data

ushort **wCfgPhyAddress**
[out] Slave configured station address

string **abyDeviceName**
abyDeviceName

bool **bIsMailboxSlave**
[out] Whether slave support mailboxes

uint **dwMbxOutSize**
[out] Mailbox 1 output size

uint **dwMbxInSize**
[out] Mailbox 1 input size

uint **dwMbxOutSize2**
[out] Mailbox 2 output size

```

uint dwMbxInSize2
    [out] Mailbox 2 input size

uint dwErrorCode
    [out] Last return code

uint dwSBErrorCode
    [out] Last return value from SB

byte byPortDescriptor
    [out] Port Descriptor (ESC register 0x0007)

byte byESCType
    [out] ESC Node Type

ushort wSupportedMbxProtocols
    [out] Supported mailbox protocols: AoE, EoE, CoE, FoE, SoE

ushort wAlStatusValue
    [out] AL Status Register Value (ESC register 0x0130)

ushort wAlStatusCode
    [out] AL Status Code (ESC register 0x0134)

bool bIsOptional
    [out] Slave is in an optional hot connect group

bool bIsPresent
    [out] Slave is currently present on bus

ushort wNumProcessVarsInp
    [out] Number of output process data variables

ushort wNumProcessVarsOutp
    [out] Number of input process data variables

uint dwSlaveId
    [out] Slave ID

bool bIsHCGroupPresent
    [out] The hot connect group of the slave is present

uint[] aPortSlaveIds
    [out] slave IDs connected to ports

uint dwSystemTimeDifference
    dwSystemTimeDifference

class EcWrapperDotNet.DN_EC_T_HC_GROUP_INFO
    DN_EC_T_HC_GROUP_INFO.

```

Public Members

uint **dwHcGroupIdx**

[out] Index of Hot Connect group

string **szGroupName**

[out] Hot connect group name (from ENI file)

uint **dwNumOfSlaves**

[out] Number of Slaves within the group

ushort **wHeadSlaveStationAddress**

[out] Physical address of the head slave of the HC group

ushort **wIdentificationAdo**

[out] Ado of Identification check

ushort **wIdentificationVal**

[out] Val of Identification check

bool **bIsPresent**

[out] EC_TRUE if HC group is present, EC_FALSE otherwise

bool **bIsMandatory**

[out] EC_TRUE if HC group is mandatory, EC_FALSE otherwise

class EcWrapperDotNet.DN_EC_T_LINK_VAR_INFO

DN_EC_T_LINK_VAR_INFO.

Public Members

IntPtr **pvContext**

[in] Arbitrarily application-defined parameter

IntPtr **pbyApplVar**

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

ushort **wApplVarType**

[in] Type of application variable

ushort **wApplVarBitSize**

[in] Size of variable (in bits)

uint **dwApplVarBitOffs**

[in] Bit offset of variable

class EcWrapperDotNet.DN_EC_T_MASTER_INFO

ecatGetMasterInfo

Public Members

uint **dwMasterVersion**

Master version.

DN_EC_T_BUS_DIAGNOSIS_INFO **BusDiagnosisInfo**

Bus diagnostics.

DN_EC_T_MAILBOX_STATISTICS **MailboxStatistics**

Mailbox statistics.

DN_EC_T_REDUNDANCY_DIAGNOSIS_INFO **RedundancyDiagnosisInfo**

Redundancy diagnosis info.

uint **dwMasterStateSummary**

Master state summary.

uint **dwMasterVersionType**

Master version type. See EC_VERSION_TYPE.

ushort **wMasterStateSummaryDiagBitOffset**

Bit offset of Master state summary in diagnosis image.

ushort **wMasterStateSummaryDiagBitSize**

Bit offset size of Master state summary in diagnosis image.

class **EcWrapperDotNet.DN_EC_T_BUS_DIAGNOSIS_INFO**

See also Object 0x2002 Bus Diagnosis Object.

Public Members

uint **dwCRC32ConfigChecksum**

CRC32 checksum of the loaded configuration.

uint **dwNumSlavesFound**

Number of slaves connected.

uint **dwNumDCSlavesFound**

Number of slaves with DC enabled connected.

uint **dwNumCfgSlaves**

Number of slaves in ENI.

uint **dwNumMbxSlaves**

Number of slaves in ENI with mailbox support.

uint **dwTXFrames**

Number of frames sent.

uint **dwRXFrames**

Number of frames received.

uint **dwLostFrames**

Number of lost frames.

uint **dwCyclicFrames**

Number of cyclic frames sent.

uint **dwCyclicDatagrams**

Number of cyclic datagrams / EtherCAT commands sent.

uint **dwAcyclicFrames**

Number of acyclic frames sent.

uint **dwAcyclicDatagrams**

Number of acyclic datagrams / EtherCAT commands sent.

uint **dwClearCounters**

Clear frame / datagram counter bit field.

uint **dwCyclicLostFrames**

Number of cyclic lost frames.

uint **dwAcyclicLostFrames**

Number of acyclic lost frames.

uint[] **dwRes**

dwRes

class EcWrapperDotNet.DN_EC_T_MAILBOX_STATISTICS

See also Object 0x2006 Mailbox Statistics Object.

Public Members

DN_EC_T_STATISTIC_TRANSFER_DUPLEX **Aoe**

AoE mailbox transfer statistics.

DN_EC_T_STATISTIC_TRANSFER_DUPLEX **Coe**

CoE mailbox transfer statistics.

DN_EC_T_STATISTIC_TRANSFER_DUPLEX **Eoe**

EoE mailbox transfer statistics.

DN_EC_T_STATISTIC_TRANSFER_DUPLEX **Foe**

FoE mailbox transfer statistics.

DN_EC_T_STATISTIC_TRANSFER_DUPLEX **Soe**

SoE mailbox transfer statistics.

DN_EC_T_STATISTIC_TRANSFER_DUPLEX **Voe**

VoE mailbox transfer statistics.

DN_EC_T_STATISTIC_TRANSFER_DUPLEX **RawMbx**

Raw mailbox transfer statistics.

DN_EC_T_STATISTIC_TRANSFER_DUPLEX **aRes**

aRes

class EcWrapperDotNet.DN_EC_T_STATISTIC_TRANSFER_DUPLEX

See also Object 0x2006 Mailbox Statistics Object.

Public Members

DN_EC_T_STATISTIC_TRANSFER **Read**

Number of read transfers.

DN_EC_T_STATISTIC_TRANSFER **Write**

Number of write transfers.

class EcWrapperDotNet.DN_EC_T_STATISTIC_TRANSFER

See also Object 0x2006 Mailbox Statistics Object.

Public Members

DN_EC_T_STATISTIC **Cnt**

Number of transfers.

DN_EC_T_STATISTIC **Bytes**

Number of bytes transferred.

class EcWrapperDotNet.DN_EC_T_STATISTIC

See also Object 0x2006 Mailbox Statistics Object.

Public Members

uint **dwTotal**

Total.

uint **dwLast**

Last.

class EcWrapperDotNet.DN_EC_T_REDUNDANCY_DIAGNOSIS_INFO

See also Object 0x2003 Redundancy Diagnosis Object.

Public Members

bool **bRedEnabled**

Cable Redundancy Enabled.

uint **dwMainSlaveCnt**
Main Line Slave Count.

uint **dwRedSlaveCnt**
Red Line Slave Count.

bool **bLineBreakDetected**
Line Break Detected.

uint[] **dwRes**
dwRes

class EcWrapperDotNet.DN_EC_T_MSU_INFO
DN_EC_T_MSU_INFO.

Public Members

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

uint **dwBitOffsIn**
[out] Process Data Image INPUTs bit offset

uint **dwBitSizeIn**
[out] Process Data Image INPUTs bit length

uint **dwBitOffsOut**
[out] Process Data Image OUTPUTs bit offset

uint **dwBitSizeOut**
[out] Process Data Image OUTPUTs bit length

ushort **wWkcStateDiagOffsIn**
[out] INPUTs WkcState bit offset in Diagnosis Image. (Bit values: 0 = Process Data valid, 1 = Process Data invalid)

ushort **wWkcStateDiagOffsOut**
[out] OUTPUTs WkcState bit offset in Diagnosis Image. (Bit values: 0 = Process Data valid, 1 = Process Data invalid)

class EcWrapperDotNet.DN_EC_T_PERF_MEAS_APP_PARMS

class EcWrapperDotNet.DN_EC_T_PERF_MEAS_HISTOGRAM
DN_EC_T_PERF_MEAS_HISTOGRAM.

Public Members

uint[] **aBinsObj**
aBinsObj

uint **dwBinCount**
Length of aBins.

ulong **qwMinTicks**
Results below qwMinTicks are stored in the first bin.

ulong **qwMaxTicks**
Results above qwMaxTicks are stored in the last bin.

class EcWrapperDotNet.DN_EC_T_PERF_MEAS_INFO
DN_EC_T_PERF_MEAS_INFO.

Public Members

string **szName**
Name of the benchmark.

ulong **qwFrequency**
Frequency in Hz used by the timer.

DN_EC_T_USER_JOB **eUserJob**
UserJob associated with the benchmark.

uint **dwBinCountHistogram**
Length of Histogram Bins.

uint **dwFlags**
Flags associated with the benchmark (See EC_T_PERF_MEAS_FLAG...)

class EcWrapperDotNet.DN_EC_T_PERF_MEAS_VAL
DN_EC_T_PERF_MEAS_VAL.

Public Members

ulong **qwCurrTicks**
[ticks]

ulong **qwMinTicks**
[ticks]

ulong **qwMaxTicks**
[ticks]

ulong **qwAvgTicks**
[ticks]

```
class EcWrapperDotNet.DN_EC_T_PROCESS_VAR_INFO
```

```
emGetSlaveInpVarInfo / emGetSlaveOutpVarInfo
```

Public Members

```
string szName
```

[out] Name of the found process variable

```
ushort wDataType
```

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

```
ushort wFixedAddr
```

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

```
int nBitSize
```

[out] Size in bits of the found process variable

```
int nBitOffs
```

[out] Bit offset in the process data image

```
bool bIsInputData
```

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

```
class EcWrapperDotNet.DN_EC_T_PROCESS_VAR_INFO_EX
```

```
DN_EC_T_PROCESS_VAR_INFO_EX.
```

Public Members

```
string szName
```

[out] Name of the found process variable

```
ushort wDataType
```

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

```
ushort wFixedAddr
```

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

```
int nBitSize
```

[out] Size in bits of the found process variable

```
int nBitOffs
```

[out] Bit offset in the process data image

```
bool bIsInputData
```

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

ushort **wIndex**

[out] Object index

ushort **wSubIndex**

[out] Object sub index

ushort **wPdoIndex**

[out] Index of PDO (process data object)

ushort **wWkcStateDiagOffs**

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

ushort **wMasterSyncUnit**

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

ushort **wRes1**

wRes1

uint **dwTaskId**

[out] ID of task where process variable is located

DN_EC_T_CYC_COPY_INFO **CopyInfo**

[out] Copy Info if applied to the variable

class EcWrapperDotNet.DN_EC_T_CYC_COPY_INFO

The master has to copy valid input data of this command from the source offset (bit offs in the complete process image) to a destination offset.

Public Members

uint **dwSrcBitOffs**

dwSrcBitOffs

uint **dwDstBitOffs**

dwDstBitOffs

ushort **wBitSize**

wBitSize

ushort **wTaskId**

wTaskId

ushort **wFlags**

wFlags

ushort **wRes**

wRes

```
uint[] dwRes
    dwRes
```

```
class EcWrapperDotNet.DN_EC_T_PROFILE_CHANNEL_INFO
    DN_EC_T_PROFILE_CHANNEL_INFO.
```

Public Members

```
ushort wProfileNo
    [out] ProfileNo: "low word of CoE object 0x1000"
```

```
ushort wAddInfo
    [out] AddInfo : "high word of CoE object 0x1000"
```

```
string szDisplayName
    [out] Display name
```

```
class EcWrapperDotNet.DN_EC_T_RAS_CONNECTION_INFO
    DN_EC_T_RAS_CONNECTION_INFO.
```

Public Members

```
uint dwAccessControlActive
    dwAccessControlActive
```

```
uint dwAccessLevel
    dwAccessLevel
```

```
class EcWrapperDotNet.DN_EC_T_REGISTERRESULTS
    Client register result.
```

Public Members

```
uint dwClntId
    [out] Client ID
```

```
IntPtr pbyPDIn
    [out] Pointer to process data input memory
```

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

```
IntPtr pbyPDOut
    [out] Pointer to process data output memory
```

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

```
class EcWrapperDotNet.DN_EC_T_SB_MISMATCH_DESC
    DN_EC_T_SB_MISMATCH_DESC.
```

Public Members

ushort **wPrevFixedAddress**

[in] Previous slave station address

ushort **wPrevPort**

[in] Previous slave station address

ushort **wPrevAIncAddress**

[in] Previous slave auto-increment address

ushort **wBusAIncAddress**

[in] Unexpected slave (bus) auto-inc address

uint **dwBusVendorId**

[in] Unexpected slave (bus) vendor ID

uint **dwBusProdCode**

[in] Unexpected slave (bus) product code

uint **dwBusRevisionNo**

[in] Unexpected slave (bus) revision number

uint **dwBusSerialNo**

[in] Unexpected slave (bus) serial number

ushort **wBusFixedAddress**

[in] Unexpected slave (bus) station address

bool **bIdentificationError**

[in] Identification command sent to slave but failed

ushort **wIdentificationAdo**

[in] Identification register

ushort **wIdentificationVal**

[in] Last identification value read from slave according to the last used identification method

ushort **wIdentificationValExpected**

[in] Identification expected value

ushort **wCfgFixedAddress**

[in] Missing slave (config) station Address

ushort **wCfgAIncAddress**

[in] Missing slave (config) Auto-Increment Address

uint **dwCfgVendorId**

[in] Missing slave (config) Vendor ID

uint **dwCfgProdCode**

[in] Missing slave (config) Product code

uint **dwCfgRevisionNo**

[in] Missing slave (config) Revision Number

uint **dwCfgSerialNo**

[in] Missing slave (config) Serial Number

```
class EcWrapperDotNet.DN_EC_T_SB_SLAVEINFO_DESC
    DN_EC_T_SB_SLAVEINFO_DESC.
```

Public Members

uint **dwScanBusStatus**

[out] Scan bus status (determined in the latest scan bus) emNotify EC_NOTIFY_SB_STATUS

uint **dwVendorId**

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

uint **dwProductCode**

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

uint **dwRevisionNumber**

[out] Revision number stored in the EEPROM at offset 0x000C (Not read by default!)

uint **dwSerialNumber**

[out] Serial number stored in the EEPROM at offset 0x000E (Not read by default!)

```
class EcWrapperDotNet.DN_EC_T_SB_SLAVEINFO_REQ_DESC
    DN_EC_T_SB_SLAVEINFO_REQ_DESC.
```

Public Members

DN_EC_T_eINFOENTRY **eInfoEntry**

[in] Info Entry to read

ushort **wAutoIncAddress**

[in] Auto-Increment address of the slave

```
enum EcWrapperDotNet.DN_EC_T_eINFOENTRY
    DN_EC_T_eINFOENTRY.
```

Values:

unknown

unknown

pdoffs_in

pdoffs_in

pdsi_{size_in}

pdsi_size_in

pdo_{ffs_out}

pdo_ffs_out

pdsi_{size_out}

pdsi_size_out

phys_{address}

phys_address

port_{state}

portstate

dc_{support}

dcsupport

dc64_{support}

dc64support

alias_{address}

alias_address

cfg_{phy_address}

cfgphy_address

device_{name}

device_name

is_{mailbox_slave}

ismailbox_slave

pdo_{ffs_in2}

pdo_ffs_in2

pdsi_{size_in2}

pdsi_size_in2

pdo_{ffs_out2}

pdo_ffs_out2

pdsi_{size_out2}

pdsi_size_out2

pdo_{ffs_in3}

pdo_ffs_in3

pdsisize_in3
pdsi

pdoffs_out3
pdoffs_out3

pdsisize_out3
pdsi_out3

pdoffs_in4
pdoffs_in4

pdsisize_in4
pdsi_in4

pdoffs_out4
pdoffs_out4

pdsisize_out4
pdsi_out4

mbx_outsize
mbx_outsize

mbx_insize
mbx_insize

mbx_outsize2
mbx_outsize2

mbx_insize2
mbx_insize2

isoptional
isoptional

ispresent
ispresent

esctype
esctype

BCppDummy
BCppDummy.

class EcWrapperDotNet.DN_EC_T_SB_SLAVEINFO_RES_DESC
DN_EC_T_SB_SLAVEINFO_RES_DESC.

Public Members

DN_EC_T_eINFOENTRY **eInfoEntry**

[out] Info entry read

uint **dwInfoLength**

[in, out] Length of Info Field (buffer, actually read length)

IntPtr **pbyInfo**

[out] Pointer to Info (-1 if no info found in XML file)

class EcWrapperDotNet.DN_EC_T_SB_STATUS_NOTIFY_DESC

DN_EC_T_SB_STATUS_NOTIFY_DESC.

Public Members

uint **dwResultCode**

[in] #EC_E_NOERROR: success, #EC_E_NOTREADY: no bus scan executed, #EC_E_BUSCONFIG_MISMATCH: bus configuration mismatch result of scanbus

uint **dwSlaveCount**

[in] Number of slaves connected to the bus

class EcWrapperDotNet.DN_EC_T_SELFTESTSCAN_PARMS

DN_EC_T_SELFTESTSCAN_PARMS.

Public Members

uint **dwTimeout**

[in] Timeout [ms], 0 or EC_NOWAIT defaults to 500ms

uint **dwFrameCount**

[in] Total number of frames sent during the self-test. Default value is 1500. A value of 0 sets the default value.

uint **dwFrameSizeMin**

[in] Min frame size [bytes]. Default value is 60. A value of 0 sets the default value.

uint **dwFrameSizeMax**

[in] Max frame size [bytes]. Default value is 1514. A value of 0 sets the default value.

uint **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.

bool **bDetectBadConnections**

[in] Execute the bad connection detection after self-test

ulong **qwFrameRoundtripTimeAvg**

[out] Roundtrip time average [us]. Time taken from sending to receiving the frame (master application level).

ulong **qwFrameRoundtripTimeMin**

[out] Roundtrip time minimum [us]. Time taken from sending to receiving the frame (master application level).

ulong **qwFrameRoundtripTimeMax**

[out] Roundtrip time maximum [us]. Time taken from sending to receiving the frame (master application level).

bool **bMeasureRoundtripTimeForSingleFrame**

[in] Execute roundtrip time calculation for single frame

bool **bClearSlaveErrorCounters**

[in] Clear all slaves' error counters before running the self-test

uint **dwDuration**

[in] Desired duration to run the self-test scan [s]. Stops as soon as dwDuration is reached. A value of 0 means no duration limit. If dwDuration is not zero, dwFrameCount is ignored.

enum **EcWrapperDotNet.DN_EC_T_SLAVE_SELECTION**

DN_EC_T_SLAVE_SELECTION.

Values:

eSlaveSelectionSingle

[in] Select only one slave

eSlaveSelectionTopoFollowers

[in] Select slave and his topological followers

eSlaveSelectionMasterSyncUnit

[in] Select slave and his topological followers

eSlaveSelectionDummy

eSlaveSelectionDummy

class **EcWrapperDotNet.DN_EC_T_SLAVE_PROP**

EtherCAT slave properties.

Public Members

ushort **wStationAddress**

Configured station address or INVALID_FIXED_ADDR.

ushort **wAutoIncAddr**

Configured auto increment address or INVALID_AUTO_INC_ADDR.

string **achName**

Configured name of the slave device (NULL terminated string)

class **EcWrapperDotNet.DN_EC_T_SLVSTATISTICS_DESC**

DN_EC_T_SLVSTATISTICS_DESC.

Public Members

byte[] **abyInvalidFrameCnt**

[out] Invalid Frame Counters per Slave Port

byte[] **abyRxErrorCnt**

[out] RX Error Counters per Slave Port

byte[] **abyFwdRxErrorCnt**

[out] Forwarded RX Error Counters per Slave Port

byte **byProcessingUnitErrorCnt**

[out] Processing Unit Error Counter

byte **byPdiErrorCnt**

[out] PDI Error Counter

ushort **wAlStatusCode**

[out] AL Status Code

byte[] **abyLostLinkCnt**

[out] Lost Link Counters per Slave Port

ulong **qwReadTime**

[out] Timestamp of the last read [ns]

ulong **qwChangeTime**

[out] Timestamp of the last counter change [ns]

enum EcWrapperDotNet.DN_EC_T_STATE

EtherCAT state.

Values:

UNKNOWN

Unknown state.

INIT

EtherCAT state INIT.

PREOP

EtherCAT state PREOP (pre-operational)

SAFEOP

EtherCAT state SAFEOP (safe operational)

OP

EtherCAT state OP (operational)

BOOTSTRAP

EtherCAT state BOOTSTRAP.

BCppDummy

BCppDummy.

enum EcWrapperDotNet.DN_EC_T_CMD

EtherCAT commands.

Values:

NOP

No operation.

APRD

Auto increment physical read.

APWR

Auto increment physical write.

APRW

Auto increment physical read write.

FPRD

Configured address physical read.

FPWR

Configured address physical write.

FPRW

Configured address physical read write.

BRD

Broadcast read.

BWR

Broadcast write.

BRW

Broadcast read write.

LRD

Logical memory read.

LWR

Logical memory write.

LRW

Logical memory read write.

ARMW

Auto increment physical read multiple write.

FRMW

Configured address physical read multiple write.

EXT

EXT.

BCppDummy

BCppDummy.

```
class EcWrapperDotNet.DN_EC_T_TRACE_DATA_INFO
    DN_EC_T_TRACE_DATA_INFO.
```

Public Members

IntPtr pbyData

[out] Process data output buffer, containing trace data

uint dwOffset

[out] Trace data offset in bytes

ushort wSize

[out] Trace data size in bytes

```
enum EcWrapperDotNet.DN_EC_T_USER_JOB
```

EtherCAT user controlled execution job.

Values:

Undefined

Undefined.

ProcessAllRxFrames

Receive frames and process all received data. Polling mode only.

SendAllCycFrames

Send all cyclic frames.

MasterTimer

Run internal master and slave state machines for generic management.

SendAcycFrames

Send acyclic frames.

SendCycFramesByTaskId

Send cyclic frames related to a specific task id (ENI: Cyclic/TaskId)

MasterTimerMinimal

Run minimal master timer routine: no state change possible.

ProcessRxFramesByTaskId

Receive frames and process received data related to a specific task id (ENI: Cyclic/TaskId)

ProcessAcycRxFrames

Receive frames and process received data related to acyclic frames.

SwitchEoeFrames

Switch queued EoE frames (see EC_IOCTL_SET_EOE_DEFERRED_SWITCHING_ENABLED)

StartTask

Start new task.

StopTask

Stop currently running task.

StampSendAllCycFrames

Obsolete.

StampSendCycFramesByTaskId

Obsolete.

SimulatorTimer

Run EC-Simulator timer routine (generic management)

MonitorTimer

Run EC-Monitor timer routine (generic management)

BCppDummy

BCppDummy.

enum EcWrapperDotNet.DN_EC_T_VAR_DIRECTION

DN_EC_T_VAR_DIRECTION.

Values:

Undefined

Undefined Direction.

INPUT

INPUTs.

OUTPUT

OUTPUTs.

All

INPUTs and OUTPUTs.

BCppDummy

BCppDummy.

enum EcWrapperDotNet.DN_EC_T_VAR_SOURCE

DN_EC_T_VAR_SOURCE.

Values:

Undefined

Undefined Source.

All

Slaves and Master/Monitor/Simulator.

AllSlaves

All Slaves.

Slave

Slave.

Master

Master.

Monitor

Monitor.

Simulator

Simulator.

BCppDummy

BCppDummy.

enum EcWrapperDotNet.DN_ESC_SII_REG

DN_ESC_SII_REG.

Values:

PDICONTROL

PDICONTROL.

PDICONFIG

PDICONFIG.

SYNCIMPULSELENGTH

SYNCIMPULSELENGTH.

EXTENDEDPDICONFIG

EXTENDEDPDICONFIG.

ALIASADDRESS

ALIASADDRESS.

CHECKSUM

CHECKSUM.

VENDORID

VENDORID.

PRODUCTCODE

PRODUCTCODE.

REVISIONNUMBER

REVISIONNUMBER.

REVISIONNUMBER_LO

REVISIONNUMBER_LO.

REVISIONNUMBER_HI

REVISIONNUMBER_HI.

SERIALNUMBER

SERIALNUMBER.

BOOT_RECV_MBX

BOOT_RECV_MBX.

BOOT_RECV_MBX_OFFSET

BOOT_RECV_MBX_OFFSET.

BOOT_RECV_MBX_SIZE

BOOT_RECV_MBX_SIZE.

BOOT_SEND_MBX

BOOT_SEND_MBX.

BOOT_SEND_MBX_OFFSET

BOOT_SEND_MBX_OFFSET.

BOOT_SEND_MBX_SIZE

BOOT_SEND_MBX_SIZE.

STD_RECV_MBX

STD_RECV_MBX.

STD_RECV_MBX_OFFSET

STD_RECV_MBX_OFFSET.

STD_RECV_MBX_SIZE

STD_RECV_MBX_SIZE.

STD_SEND_MBX

STD_SEND_MBX.

STD_SEND_MBX_OFFSET

STD_SEND_MBX_OFFSET.

STD_SEND_MBX_SIZE

STD_SEND_MBX_SIZE.

MBX_PROTOCOL

MBX_PROTOCOL.

FIRSTCATEGORYHDR

FIRSTCATEGORYHDR.

class EcWrapperDotNet.DN_ETHERNET_ADDRESS

DN_ETHERNET_ADDRESS.

Public Members

byte[] **b**

MAC address.

enum EcWrapperDotNet.DN_NotifyCode

Specifies EtherCAT errors codes.

Values:

UNDEFINED

STATECHANGED

ETH_LINK_CONNECTED

SB_STATUS

DC_STATUS

DC_SLV_SYNC

DCL_STATUS

DCM_SYNC

DCX_SYNC

SLAVE_STATECHANGED

SLAVES_STATECHANGED

RAWCMD_DONE

COE_TX_PDO

SLAVE_PRESENCE

SLAVES_PRESENCE

REFCLOCK_PRESENCE

MASTER_RED_STATECHANGED

MASTER_RED_FOREIGN_SRC_MAC

SLAVE_REGISTER_TRANSFER

PORT_OPERATION

SLAVE_IDENTIFICATION

RELEASE_FORCED_PROCESSDATA

SB_EEPROM_ACCESS_DENIED

SB_SLAVE_IDENT_TIMEOUT

CONFIG_LOAD

FSoE_CONNECTION_STATECHANGED

CYCCMD_WKC_ERROR

MASTER_INITCMD_WKC_ERROR

SLAVE_INITCMD_WKC_ERROR

EOE_MBXSEND_WKC_ERROR

COE_MBXSEND_WKC_ERROR

FOE_MBXSEND_WKC_ERROR

FRAME_RESPONSE_ERROR

SLAVE_INITCMD_RESPONSE_ERROR

MASTER_INITCMD_RESPONSE_ERROR

MBSLAVE_INITCMD_TIMEOUT

NOT_ALL_DEVICES_OPERATIONAL

ETH_LINK_NOT_CONNECTED

RED_LINEBRK

STATUS_SLAVE_ERROR

SLAVE_ERROR_STATUS_INFO

SLAVE_NOT_ADDRESSABLE

SOE_MBXSEND_WKC_ERROR

SOE_WRITE_ERROR

MBSLAVE_COE_SDO_ABORT

CLIENTREGISTRATION_DROPPED

RED_LINEFIXED

FOE_MBSLAVE_ERROR

MBXRCV_INVALID_DATA

PDIWATCHDOG

SLAVE_NOTSUPPORTED

SLAVE_UNEXPECTED_STATE

ALL_DEVICES_OPERATIONAL

VOE_MBXSEND_WKC_ERROR

AOE_MBXSEND_WKC_ERROR

```

EEPROM_CHECKSUM_ERROR

JUNCTION_RED_CHANGE

SLAVES_UNEXPECTED_STATE

LINE_CROSSED

SLAVES_ERROR_STATUS

FRAMELOSS_AFTER_SLAVE

S2SMBX_ERROR

BAD_CONNECTION

COMMUNICATION_TIMEOUT

TAP_LINK_STATUS

MBOXRCV

SB_MISMATCH

SB_DUPLICATE_HC_NODE

HC_DETECTADDGROUPS

HC_PROBEALLGROUPS

HC_TOPOCHGDONE

```

3.3 Error Codes

```
enum EcWrapperDotNet.ECError
    summary> Specifies EtherCAT errors codes.
```

Values:

```

EC_NOERROR
    0x00000000: No Error

EC_ERROR
    0x98110000: Unspecific Error

```

EMRAS_ERROR

0x98110180: Unspecific RAS Error

EC_NOTSUPPORTED

0x98110001: APP: Feature not supported (e.g. function or property not available)

EC_INVALIDINDEX

0x98110002: APP: Invalid index (e.g. CoE: invalid SDO index)

EC_INVALIDOFFSET

0x98110003: ISW: Invalid offset (e.g. invalid offset while accessing Process Data Image)

EC_CANCEL

0x98110004: APP: Cancel (e.g. EtherCAT stack should abort current mailbox transfer)

EC_INVALIDSIZE

0x98110005: APP: Invalid size

EC_INVALIDDATA

0x98110006: ISW: Invalid data (multiple error sources)

EC_NOTREADY

0x98110007: ISW: Not ready (multiple error sources)

EC_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_ACYC_FRM_FREEQ_EMPTY

0x98110009: ISW: Cannot queue acyclic EtherCAT command (Acyclic command queue is full. Possible solution: Increase of configuration value dwMaxQueuedEthFrames)

EC_NOMEMORY

0x9811000A: CFG: No memory left (e.g. memory full / fragmented))

EC_INVALIDPARM

0x9811000B: APP: Invalid parameter (e.g. API function called with erroneous parameter set)

EC_NOTFOUND

0x9811000C: APP: Not found (e.g. Network Information File ENI not found or API called with invalid slave ID)

EC_DUPLICATE

0x9811000D: ISW: Duplicated fixed address detected (handled internally)

EC_INVALIDSTATE

0x9811000E: ISW: Invalid state (EtherCAT stack not initialized or not configured)

EC_TIMER_LIST_FULL

0x9811000F: ISW: Cannot add slave to timer list (slave timer list full)

EC_TIMEOUT

0x98110010: Timeout

EC_OPENFAILED

0x98110011: ISW: Open failed

EC_SENDFAILED

0x98110012: LLA: Frame send failed

EC_INSERTMAILBOX

0x98110013: CFG: Insert Mailbox error (internal limit MAX_QUEUED_COE_CMDS: 20)

EC_INVALIDCMD

0x98110014: ISW: Invalid Command (Unknown mailbox command code)

EC_UNKNOWN_MBX_PROTOCOL

0x98110015: ISW: Unknown Mailbox Protocol Command (Unknown Mailbox protocol or mailbox command with unknown protocol association)

EC_ACCESSDENIED

0x98110016: ISW: Access Denied (e.g. master internal software error)

EC_IDENTIFICATIONFAILED

0x98110017: ENI: Identification failed (e.g. identification command failed)

EC_LOCK_CREATE_FAILED

0x98110018: SYS: Create lock failed (e.g. OsCreateLockTyped failed)

EC_VERSION_MISMATCH

0x98110019: APP: Version mismatch for loaded library

EC_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_WRONG_FORMAT

0x9811001B: ENI: Wrong configuration format (e.g. Network information file empty or malformed),
SLV: Malformed EEPROM content

EC_FEATURE_DISABLED

0x9811001C: APP: Feature disabled (e.g. Application tried to perform a missing or disabled API function)

EC_SHADOW_MEMORY

0x9811001D: Shadow memory requested in wrong mode

EC_BUSCONFIG_MISMATCH

0x9811001E: ENI: Bus configuration mismatch (e.g. Network information file and currently connected bus topology does not match)

EC_CONFIGDATAREAD

0x9811001F: ENI: Error reading configuration file (e.g. Network information file could not be read)

EC_ENI_NO_SAFEOP_OP_SUPPORT

0x98110020: Configuration doesn't support SAFEOP and OP requested state

EC_XML_CYCCMDS_MISSING

0x98110021: ENI: Cyclic commands are missing (e.g. Network information file does not contain cyclic commands)

EC_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_MCSM_FATAL_ERROR

0x98110023: ISW: Fatal internal McSm (master control state machine is in an undefined state)

EC_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_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_CMD_MISSING

0x98110026: SLV: At least one EtherCAT command is missing in the received frame (e.g. received EtherCAT frame incomplete)

EC_CYCCMD_WKC_ERROR

0x98110027: Cyclic command WKC error

EC_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_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_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_SLAVE_NOT_ADDRESSABLE

0x9811002B: SLV: Station address lost (or slave missing) - FPRD to AL_STATUS failed (e.g. Slave had a power cycle)

EC_CYC_CMDS_OVERFLOW

0x9811002C: ENI: Too many cyclic commands in XML configuration file (e.g. EC_T_INIT_MASTER_PARMs.dwMaxAcycFramesQueued too small)

EC_LINK_DISCONNECTED

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

EC_MASTERCORE_INACCESSIBLE

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

EC_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_COE_MBXRCV_WKC_ERROR

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

EC_NO_MBX_SUPPORT

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

EC_NO_COE_SUPPORT

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

EC_NO_EOE_SUPPORT

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

EC_NO_FOE_SUPPORT

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

EC_NO_SOE_SUPPORT

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

EC_NO_VOE_SUPPORT

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

EC_EVAL_VIOLATION

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

EC_EVAL_EXPIRED

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

EC_LICENSE_MISSING

0x98110039: License key invalid or missing

EC_SDO_ABORTCODE_FIRST

EC_SDO_ABORTCODE_FIRST.

EC_SDO_ABORTCODE_TOGGLE

0x98110040: SLV: SDO: Toggle bit not alternated (CoE abort code 0x05030000 of slave)

EC_SDO_ABORTCODE_TIMEOUT

0x98110041: SLV: SDO: Protocol timed out (CoE abort code 0x05040000 of slave)

EC_SDO_ABORTCODE_CCS_SCS

0x98110042: SLV: SDO: Client/server command specifier not valid or unknown (CoE abort code 0x05040001 of slave)

EC_SDO_ABORTCODE_BLK_SIZE

0x98110043: SLV: SDO: Invalid block size (block mode only) (CoE abort code 0x05040002 of slave)

EC_SDO_ABORTCODE_SEQNO

0x98110044: SLV: SDO: Invalid sequence number (block mode only) (CoE abort code 0x05040003 of slave)

EC_SDO_ABORTCODE_CRC

0x98110045: SLV: SDO: CRC error (block mode only) (CoE abort code 0x05040004 of slave)

EC_SDO_ABORTCODE_MEMORY

0x98110046: SLV: SDO: Out of memory (CoE abort code 0x05040005 of slave)

EC_SDO_ABORTCODE_ACCESS

0x98110047: SLV: SDO: Unsupported access to an object (CoE abort code 0x06010000 of slave)

EC_SDO_ABORTCODE_WRITEONLY

0x98110048: SLV: SDO: Attempt to read a write only object (CoE abort code 0x06010001 of slave)

EC_SDO_ABORTCODE_READONLY

0x98110049: SLV: SDO: Attempt to write a read only object (CoE abort code 0x06010002 of slave)

EC_SDO_ABORTCODE_INDEX

0x9811004A: SLV: SDO: Object does not exist in the object dictionary (CoE abort code 0x06020000 of slave)

EC_SDO_ABORTCODE_PDO_MAP

0x9811004B: SLV: SDO: Object cannot be mapped to the PDO (CoE abort code 0x06040041 of slave)

EC_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_SDO_ABORTCODE_P_INCOMP

0x9811004D: SLV: SDO: General parameter incompatibility reason (CoE abort code 0x06040043 of slave)

EC_SDO_ABORTCODE_I_INCOMP

0x9811004E: SLV: SDO: General internal incompatibility in the device (CoE abort code 0x06040047 of slave)

EC_SDO_ABORTCODE_HARDWARE

0x9811004F: SLV: SDO: Access failed due to a hardware error (CoE abort code 0x06060000 of slave)

EC_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_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_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_SDO_ABORTCODE_OFFSET

0x98110053: SLV: SDO: Sub-index does not exist (CoE abort code 0x06090011 of slave)

EC_SDO_ABORTCODE_VALUE_RANGE

0x98110054: SLV: SDO: Value range of parameter exceeded (only for write access) (CoE abort code 0x06090030 of slave)

EC_SDO_ABORTCODE_VALUE_TOO_HIGH

0x98110055: SLV: SDO: Value of parameter written too high (CoE abort code 0x06090031 of slave)

EC_SDO_ABORTCODE_VALUE_TOO_LOW

0x98110056: SLV: SDO: Value of parameter written too low (CoE abort code 0x06090032 of slave)

EC_SDO_ABORTCODE_MINMAX

0x98110057: SLV: SDO: Maximum value is less than minimum value (CoE abort code 0x06090036 of slave)

EC_SDO_ABORTCODE_GENERAL

0x98110058: SLV: SDO: General error (CoE abort code 0x08000000 of slave)

EC_SDO_ABORTCODE_TRANSFER

0x98110059: SLV: SDO: Data cannot be transferred or stored to the application (CoE abort code 0x08000020 of slave)

EC_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_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_SDO_ABORTCODE_DICTIONARY

0x9811005C: SLV: SDO: Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error) (CoE abort code 0x08000023 of slave)

EC_SDO_ABORTCODE_UNKNOWN

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

EC_SDO_ABORTCODE_MODULE_ID_LIST_NOT_MATCH

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

EC_SDO_ABORTCODE_LAST

EC_SDO_ABORTCODE_LAST.

EC_FOE_ERRCODE_NOTDEFINED

0x98110060: SLV: ERROR FoE: not defined (FoE Error Code 0 (0x8000) of slave)

EC_FOE_ERRCODE_NOTFOUND

0x98110061: SLV: ERROR FoE: not found (FoE Error Code 1 (0x8001) of slave)

EC_FOE_ERRCODE_ACCESS

0x98110062: SLV: ERROR FoE: access denied (FoE Error Code 2 (0x8002) of slave)

EC_FOE_ERRCODE_DISKFULL

0x98110063: SLV: ERROR FoE: disk full (FoE Error Code 3 (0x8003) of slave)

EC_FOE_ERRCODE_ILLEGAL

0x98110064: SLV: ERROR FoE: illegal (FoE Error Code 4 (0x8004) of slave)

EC_FOE_ERRCODE_PACKENO

0x98110065: SLV: ERROR FoE: packet number wrong (FoE Error Code 5 (0x8005) of slave)

EC_FOE_ERRCODE_EXISTS

0x98110066: SLV: ERROR FoE: already exists (FoE Error Code 6 (0x8006) of slave)

EC_FOE_ERRCODE_NOUSER

0x98110067: SLV: ERROR FoE: no user (FoE Error Code 7 (0x8007) of slave)

EC_FOE_ERRCODE_BOOTSTRAPONLY

0x98110068: SLV: ERROR FoE: bootstrap only (FoE Error Code 8 (0x8008) of slave)

EC_FOE_ERRCODE_NOTINBOOTSTRAP

0x98110069: SLV: ERROR FoE: Downloaded file name is not valid in Bootstrap state (FoE Error Code 9 (0x8009) of slave)

EC_FOE_ERRCODE_INVALIDPASSWORD

0x9811006A: SLV: ERROR FoE: no rights (FoE Error Code 10 (0x800A) of slave)

EC_FOE_ERRCODE_PROGERROR

0x9811006B: SLV: ERROR FoE: program error (FoE Error Code 11 (0x800B) of slave)

EC_FOE_ERRCODE_INVALID_CHECKSUM

0x9811006C: FoE: Wrong checksum

EC_FOE_ERRCODE_INVALID_FIRMWARE

0x9811006D: SLV: ERROR FoE: Firmware does not fit for Hardware (FoE Error Code 13 (0x800D) of slave)

EC_FOE_ERRCODE_NO_FILE

0x9811006F: SLV: ERROR FoE: No file to read (FoE Error Code 15 (0x800F) of slave)

EC_CFGFILENOTFOUND

0x98110070: CFG: Network configuration file not found (e.g. path to configuration file (XML) was wrong or the file is not available)

EC_EEPROMREADERROR

0x98110071: SLV: Command error while EEPROM upload (read slave EEPROM)

EC_EEPROMWRITEERROR

0x98110072: SLV: Command error while EEPROM download (write slave EEPROM)

EC_XML_CYCCMDS_SIZE_MISMATCH

0x98110073: ENI: Cyclic command wrong size (too long) (size in network configuration file (XML) does not match size of process data)

EC_XML_INVALID_INP_OFF

0x98110074: ENI: Invalid input offset in cyclic command, please check InputOffs

EC_XML_INVALID_OUT_OFF

0x98110075: ENI: Invalid output offset in cyclic command, please check OutputOffs

EC_PORTCLOSE

0x98110076: Port close failed

EC_PORTOPEN

0x98110077: Port open failed

EC_SOE_ERRORCODE_INVALID_ACCESS

0x98110078: ERROR SoE: Invalid access to element 0

EC_SOE_ERRORCODE_NOT_EXIST

0x98110079: ERROR SoE: Does not exist

EC_SOE_ERRORCODE_INVL_ACC_ELEM1

0x9811007A: ERROR SoE: Invalid access to element 1

EC_SOE_ERRORCODE_NAME_NOT_EXIST

0x9811007B: ERROR SoE: Name does not exist

EC_SOE_ERRORCODE_NAME_UNDERSIZE

0x9811007C: ERROR SoE: Name undersize in transmission

EC_SOE_ERRORCODE_NAME_OVERSIZE

0x9811007D: ERROR SoE: Name oversize in transmission

EC_SOE_ERRORCODE_NAME_UNCHANGE

0x9811007E: ERROR SoE: Name unchangeable

EC_SOE_ERRORCODE_NAME_WR_PROT

0x9811007F: ERROR SoE: Name currently write-protected

EC_SOE_ERRORCODE_UNDEERS_TRANS

0x98110080: ERROR SoE: Attribute undersize in transmission

EC_SOE_ERRORCODE_OVERS_TRANS

0x98110081: ERROR SoE: Attribute oversize in transmission

EC_SOE_ERRORCODE_ATTR_UNCHANGE

0x98110082: ERROR SoE: Attribute unchangeable

EC_SOE_ERRORCODE_ATTR_WR_PROT

0x98110083: ERROR SoE: Attribute currently write-protected

EC_SOE_ERRORCODE_UNIT_NOT_EXIST

0x98110084: ERROR SoE: Unit does not exist

EC_SOE_ERRORCODE_UNIT_UNDERSIZE

0x98110085: ERROR SoE: Unit undersize in transmission

EC_SOE_ERRORCODE_UNIT_OVERSIZE

0x98110086: ERROR SoE: Unit oversize in transmission

EC_SOE_ERRORCODE_UNIT_UNCHANGE

0x98110087: ERROR SoE: Unit unchangeable

EC_SOE_ERRORCODE_UNIT_WR_PROT

0x98110088: ERROR SoE: Unit currently write-protected

EC_SOE_ERRORCODE_MIN_NOT_EXIST

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

EC_SOE_ERRORCODE_MIN_UNDERSIZE

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

EC_SOE_ERRORCODE_MIN_OVERSIZE

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

EC_SOE_ERRORCODE_MIN_UNCHANGE

0x9811008C: ERROR SoE: Minimum input value unchangeable

EC_SOE_ERRORCODE_MIN_WR_PROT

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

EC_SOE_ERRORCODE_MAX_NOT_EXIST

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

EC_SOE_ERRORCODE_MAX_UNDERSIZE

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

EC_SOE_ERRORCODE_MAX_OVERSIZE

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

EC_SOE_ERRORCODE_MAX_UNCHANGE

0x98110091: ERROR SoE: Maximum input value unchangeable

EC_SOE_ERRORCODE_MAX_WR_PROT

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

EC_SOE_ERRORCODE_DATA_NOT_EXIST

0x98110093: ERROR SoE: Data item does not exist

EC_SOE_ERRORCODE_DATA_UNDERSIZE

0x98110094: ERROR SoE: Data item undersize in transmission

EC_SOE_ERRORCODE_DATA_OVERSIZE

0x98110095: ERROR SoE: Data item oversize in transmission

EC_SOE_ERRORCODE_DATA_UNCHANGE

0x98110096: ERROR SoE: Data item unchangeable

EC_SOE_ERRORCODE_DATA_WR_PROT

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

EC_SOE_ERRORCODE_DATA_MIN_LIMIT

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

EC_SOE_ERRORCODE_DATA_MAX_LIMIT

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

EC_SOE_ERRORCODE_DATA_INCOR

0x9811009A: ERROR SoE: Data item incorrect

EC_SOE_ERRORCODE_PASWD_PROT

0x9811009B: ERROR SoE: Data item protected by password

EC_SOE_ERRORCODE_TEMP_UNCHANGE

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

EC_SOE_ERRORCODE_INVL_INDIRECT

0x9811009D: ERROR SoE: Invalid indirect

EC_SOE_ERRORCODE_TEMP_UNCHANGE1

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

EC_SOE_ERRORCODE_ALREADY_ACTIVE

0x9811009F: ERROR SoE: Command already active

EC_SOE_ERRORCODE_NOT_INTERRUPT

0x98110100: ERROR SoE: Command not interruptible

EC_SOE_ERRORCODE_CMD_NOT_AVAIL

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

EC_SOE_ERRORCODE_CMD_NOT_AVAIL1

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

EC_SOE_ERRORCODE_DRIVE_NO

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

EC_SOE_ERRORCODE_IDN

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

EC_SOE_ERRORCODE_FRAGMENT_LOST

0x98110105: ERROR SoE: At least one fragment lost

EC_SOE_ERRORCODE_BUFFER_FULL

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

EC_SOE_ERRORCODE_NO_DATA

0x98110107: ERROR SoE: No data state

EC_SOE_ERRORCODE_NO_DEFAULT_VALUE

0x98110108: ERROR SoE: No default value

EC_SOE_ERRORCODE_DEFAULT_LONG

0x98110109: ERROR SoE: Default value transmission too long

EC_SOE_ERRORCODE_DEFAULT_WP

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

EC_SOE_ERRORCODE_INVL_DRIVE_NO

0x9811010B: ERROR SoE: Invalid drive number

EC_SOE_ERRORCODE_GENERAL_ERROR

0x9811010C: ERROR SoE: General error

EC_SOE_ERRCODE_NO_ELEM_ADR

0x9811010D: ERROR SoE: No element addressed

EC_SLAVE_NOT_PRESENT

0x9811010E: APP / SLV: command not executed (slave not present on bus) (e.g. slave disappeared or was never present)

EC_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_EEPROMRELOADERROR

0x98110110: Command error while EEPROM reload

EC_SLAVECTRLRESETERERROR

0x98110111: Command error while Reset Slave Controller

EC_SYSDRIVERMISSING

0x98110112: SYS: Cannot open system driver (e.g. system driver was not loaded)

EC_BUSCONFIG_TOPOCHANGE

0x9811011E: Bus configuration not detected, Topology changed (e.g. Topology changed while scanning bus)

EC_EOE_MBX_WKC_ERROR

0x9811011F: EoE: Mailbox receive: working counter

EC_FOE_MBX_WKC_ERROR

0x98110120: FoE: Mailbox receive: working counter

EC_SOE_MBX_WKC_ERROR

0x98110121: SoE: mailbox receive: working counter

EC_AOE_MBX_WKC_ERROR

0x98110122: AoE: Mailbox receive: working counter

EC_VOE_MBX_WKC_ERROR

0x98110123: SLV: VoE mailbox send: working counter (VoE mailbox couldn't be written)

EC_EEPROMASSIGNERROR

0x98110124: SLV: EEPROM assignment failed

EC_MBX_ERROR_TYPE

0x98110125: SLV: Unknown mailbox error code received in mailbox

EC_REDLINEBREAK

0x98110126: SLV: Redundancy line break (e.g. cable break between slaves or between EtherCAT network adapter and first slave)

EC_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_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_XML_DC_CYCCMDS_MISSING

0x98110129: DC enabled and DC cyclic commands missing (e.g. access to 0x0900)

EC_DLSTATUS_IRQ_TOPOCHANGED

0x98110130: SLV: Data link (DL) status interrupt because of changed topology (automatically handled by master)

EC_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_PTS_IS_RUNNING

0x98110132: PTS: Pass Through Server is running (obsolete, replaced by EC_E_ADS_IS_RUNNING)

EC_ADS_IS_RUNNING

0x98110132: PTS: ADS adapter (Pass Through Server) is running (API call conflicts with ADS state (running))

EC_PTS_THREAD_CREATE_FAILED

0x98110133: PTS: Could not start the Pass Through Server

EC_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_PTS_NOT_ENABLED

0x98110135: PTS: The Pass Through Server is running but not enabled

EC_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_VOE_NO_MBX_RECEIVED

0x98110137: SLV: No VoE mailbox received yet from specific slave

EC_DC_REF_CLOCK_SYNC_OUT_UNIT_DISABLED

0x98110138: DC (time loop control) unit of reference clock disabled

EC_DC_REF_CLOCK_NOT_FOUND

0x98110139: SLV: Reference clock not found! May happen if reference clock is removed from network.

EC_MBX_CMD_WKC_ERROR

0x9811013B: SLV: Mailbox command working counter error (e.g. Mailbox init command Retry Count exceeded)

EC_NO_AOE_SUPPORT

0x9811013C: APP / SLV: AoE: Protocol not supported (e.g. Application calls AoE-API although not implemented at slave)

EC_AOE_INV_RESPONSE_SIZE

0x9811013D: AoE: Invalid AoE response received

EC_AOE_ERROR

0x9811013E: AoE: Common AoE device error

EC_AOE_SRVNOTSUPP

0x9811013F: AoE: Service not supported by server

EC_AOE_INVALIDGRP

0x98110140: AoE: Invalid index group

EC_AOE_INVALIDOFFSET

0x98110141: AoE: Invalid index offset

EC_AOE_INVALIDACCESS

0x98110142: AoE: Reading/writing not permitted

EC_AOE_INVALIDSIZE

0x98110143: AoE: Parameter size not correct

EC_AOE_INVALIDDATA

0x98110144: AoE: Invalid parameter value(s)

EC_AOE_NOTREADY

0x98110145: AoE: Device not in a ready state

EC_AOE_BUSY

0x98110146: AoE: Device busy

EC_AOE_INVALIDCONTEXT

0x98110147: AoE: Invalid context

EC_AOE_NOMEMORY
0x98110148: AoE: Out of memory

EC_AOE_INVALIDPARM
0x98110149: AoE: Invalid parameter value(s)

EC_AOE_NOTFOUND
0x9811014A: AoE: Not found

EC_AOE_SYNTAX
0x9811014B: AoE: Syntax error in command or file

EC_AOE_INCOMPATIBLE
0x9811014C: AoE: Objects do not match

EC_AOE_EXISTS
0x9811014D: AoE: Object already exists

EC_AOE_SYMBOLNOTFOUND
0x9811014E: AoE: Symbol not found

EC_AOE_SYMBOLVERSIONINVALID
0x9811014F: AoE: Symbol version invalid

EC_AOE_INVALIDSTATE
0x98110150: AoE: Server in invalid state

EC_AOE_TRANSMODENOTSUPP
0x98110151: AoE: AdsTransMode not supported

EC_AOE_NOTIFYHNDINVALID
0x98110152: AoE: Notification handle invalid

EC_AOE_CLIENTUNKNOWN
0x98110153: AoE: Notification client not registered

EC_AOE_NOMOREHDLS
0x98110154: AoE: No more notification handles

EC_AOE_INVALIDWATCHSIZE
0x98110155: AoE: Size for watch to big

EC_AOE_NOTINIT
0x98110156: AoE: Device not initialized

EC_AOE_TIMEOUT
0x98110157: AoE: Device has a timeout

EC_AOE_NOINTERFACE

0x98110158: AoE: Query interface failed

EC_AOE_INVALIDINTERFACE

0x98110159: AoE: Wrong interface required

EC_AOE_INVALIDCLSID

0x9811015A: AoE: Class ID invalid

EC_AOE_INVALIDOBJID

0x9811015B: AoE: Object ID invalid

EC_AOE_PENDING

0x9811015C: AoE: Request pending

EC_AOE_ABORTED

0x9811015D: AoE: Request aborted

EC_AOE_WARNING

0x9811015E: AoE: Signal warning

EC_AOE_INVALIDARRAYIDX

0x9811015F: AoE: Invalid array index

EC_AOE_SYMBOLNOTACTIVE

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

EC_AOE_ACCESSDENIED

0x98110161: AoE: Access denied

EC_AOE_INTERNAL

0x98110162: AoE: Internal error

EC_AOE_TARGET_PORT_NOT_FOUND

0x98110163: AoE: Target port not found

EC_AOE_TARGET_MACHINE_NOT_FOUND

0x98110164: AoE: Target machine not found

EC_AOE_UNKNOWN_CMD_ID

0x98110165: AoE: Unknown command ID

EC_AOE_PORT_NOT_CONNECTED

0x98110166: AoE: Port not connected

EC_AOE_INVALID_AMS_LENGTH

0x98110167: AoE: Invalid AMS length

EC_AOE_INVALID_AMS_ID

0x98110168: AoE: invalid AMS Net ID

EC_AOE_PORT_DISABLED

0x98110169: AoE: Port disabled

EC_AOE_PORT_CONNECTED

0x9811016A: AoE: Port already connected

EC_AOE_INVALID_AMS_PORT

0x9811016B: AoE: Invalid AMS port

EC_AOE_NO_MEMORY

0x9811016C: AoE: No memory

EC_AOE_VENDOR_SPECIFIC

0x9811016D: AoE: Vendor specific AoE device error

EC_XML_AOE_NETID_INVALID

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

EC_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_MBXERR_SYNTAX

0x98110170: SLV: Mailbox error: Syntax of 6 octet Mailbox header is wrong (Slave error mailbox return value: 0x01)

EC_MBXERR_UNSUPPORTEDPROTOCOL

0x98110171: SLV: Mailbox error: The Mailbox protocol is not supported (Slave error mailbox return value: 0x02)

EC_MBXERR_INVALIDCHANNEL

0x98110172: SLV: Mailbox error: Field contains wrong value (Slave error mailbox return value: 0x03)

EC_MBXERR_SERVICENOTSUPPORTED

0x98110173: SLV: Mailbox error: The addressed service in the mailbox protocol is not supported (Slave error mailbox return value: 0x04)

EC_MBXERR_INVALIDHEADER

0x98110174: SLV: Mailbox error: The mailbox protocol header of the mailbox protocol is wrong (Slave error mailbox return value: 0x05)

EC_MBXERR_SIZETOOSHORT

0x98110175: SLV: Mailbox error: Length of received mailbox data is too short (Slave error mailbox return value: 0x06)

EC_MBXERR_NOMOREMEMORY

0x98110176: SLV: Mailbox error: Mailbox protocol can not be processed because of limited resources (Slave error mailbox return value: 0x07)

EC_MBXERR_INVALIDSIZE

0x98110177: SLV: Mailbox error: The length of data is inconsistent (Slave error mailbox return value: 0x08)

EC_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_DATA_TYPE_CONVERSION_FAILED

0x98110179: Data type conversion failed

EC_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_LINE_CROSSED

0x9811017B: Line crossed (cabling wrong)

EC_LINE_CROSSED_SLAVE_INFO

0x9811017C: Line crossed at slave (obsolete)

EC_SOCKET_DISCONNECTED

0x9811017D: RAS: Socket disconnected (e.g. IP connection terminated or lost)

EC_ADO_NOT_SUPPORTED

0x9811017E: SLV: ADO for slave identification not supported (e.g. Request ID mechanism (ADO 0x134) not supported by slave)

EC_FRAMELOSS_AFTER_SLAVE

0x9811017F: Frameloss after Slave (opening port destroys communication)

EC_ERROR_LAST

EC_ERROR_LAST.

EC_ERROR2

EC_ERROR2.

EC_FOE_ERRCODE_FILE_HEAD_MISSING

0x98130001: SLV: ERROR FoE: File header does not exist (FoE Error Code 16 (0x8010) of slave)

EC_FOE_ERRCODE_FLASH_PROBLEM

0x98130002: SLV: ERROR FoE: Flash problem (FoE Error Code 17 (0x8011) of slave)

EC_FOE_ERRCODE_FILE_INCOMPATIBLE

0x98130003: SLV: ERROR FoE: File incompatible (FoE Error Code 18 (0x8012) of slave)

EC_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_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_SDO_ABORTCODE_OBJ_TOO_BIG

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

EC_SDO_ABORTCODE_PDO_MAPPED

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

EC_OEM_SIGNATURE_MISMATCH

0x98130008: ENI, OEM: Manufacturer signature mismatch

EC_ENI_ENCRYPTION_WRONG_VERSION

0x98130009: ENI, OEM: ENI encryption algorithm version not supported

EC_ENI_ENCRYPTED

0x9813000A: OEM: Loading encrypted ENI needs OEM key

EC_OEM_KEY_MISMATCH

0x9813000B: RAS, APP: OEM key mismatch

EC_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_AOE_NO_RUNTIME

0x9813000D: AoE: No Rtime

EC_AOE_LOCKED_MEMORY

0x9813000E: AoE: Allocation locked memory

EC_AOE_MAILBOX

0x9813000F: AoE: Insert mailbox error

EC_AOE_WRONG_HMSG

0x98130010: AoE: Wrong receive HMSG

EC_AOE_BAD_TASK_ID

0x98130011: AoE: Bad task ID

EC_AOE_NO_IO

0x98130012: AoE: No IO

EC_AOE_UNKNOWN_AMS_COMMAND
 0x98130013: AoE: Unknown ADS command

EC_AOE_WIN32
 0x98130014: AoE: Win 32 error

EC_AOE_LOW_INSTALL_LEVEL
 0x98130015: AoE: Low installation level

EC_AOE_NO_DEBUG
 0x98130016: AoE: No debug available

EC_AOE_AMS_SYNC_WIN32
 0x98130017: AoE: Sync Win 32 error

EC_AOE_AMS_SYNC_TIMEOUT
 0x98130018: AoE: Sync Timeout

EC_AOE_AMS_SYNC_AMS
 0x98130019: AoE: Sync AMS error

EC_AOE_AMS_SYNC_NO_INDEX_MAP
 0x9813001A: AoE: Sync no index map

EC_AOE_TCP_SEND
 0x9813001B: AoE: TCP send error

EC_AOE_HOST_UNREACHABLE
 0x9813001C: AoE: Host unreachable

EC_AOE_INVALIDAMSFRAGMENT
 0x9813001D: AoE: Invalid AMS fragment

EC_AOE_NO_LOCKED_MEMORY
 0x9813001E: AoE: No allocation locked memory

EC_AOE_MAILBOX_FULL
 0x9813001F: AoE: Mailbox full

EC_S2SMBX_NOT_CONFIGURED
 0x98130020: S2S: Not Configured

EC_S2SMBX_NO_MEMORY
 0x98130021: S2S: No Memory

EC_S2SMBX_NO_DESCRIPTOR
 0x98130022: S2S: No Descriptor

EC_S2SMBX_DEST_SLAVE_NOT_FOUND

0x98130023: S2S: Destination Slave not found

EC_MASTER_RED_STATE_INACTIVE

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

EC_MASTER_RED_STATE_ACTIVE

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

EC_JUNCTION_RED_LINE_BREAK

0x98130026: Junction redundancy line break

EC_VALIDATION_ERROR

0x98130027: Validation error (validation data mismatch)

EC_TIMEOUT_WAITING_FOR_DC

0x98130028: Timeout waiting for DC

EC_TIMEOUT_WAITING_FOR_DCM

0x98130029: Timeout waiting for DCM

EC_SIGNATURE_MISMATCH

0x98130030: Signature mismatch

EC_PDIWATCHDOG

0x98130031: PDI watchdog expired

EC_BAD_CONNECTION

0x98130032: Bad connection

EC_XML_INCONSISTENT

0x98130033: ENI: Inconsistent content

EC_ERROR_LAST2

EC_ERROR_LAST2.

EMRAS_INVALIDCOOKIE

0x98110181: RAS: Invalid Cookie (e.g.obsolete)

EMRAS_MULSRVDISMULCON

0x98110183: RAS: Connect 2nd server denied because Multi Server support is disabled (obsolete)

EMRAS_LOGONCANCELLED

0x98110184: RAS: Logon canceled (Server-side connection reject while opening a client connection.)

EMRAS_INVALIDVERSION

0x98110186: RAS: Invalid Version (Connection reject because of using mismatching protocol versions on client and server side)

EMRAS_INVALIDACCESSCONFIG

0x98110187: RAS: Access configuration is invalid (e.g. SPoC access configuration invalid)

EMRAS_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_INVALIDDATARECEIVED

0x98110189: RAS: Invalid data received (communication corrupted)

EMRAS_SERVERSTOPPED

0x98110191: RAS: Server stopped (e.g. connection dropped because of Remote API Server stop)

EMRAS_WDEXPIRED

0x98110192: RAS: Watchdog expired (e.g. connection dropped because of missing keep-alive messages)

EMRAS_RECONEXPIRED

0x98110193: RAS: Reconnect expired (obsolete)

EMRAS_CLIENTLOGON

0x98110194: RAS Server: Client logged on

EMRAS_RECONNECT

0x98110195: RAS: obsolete

EMRAS_SOCKCHANGE

0x98110196: RAS: Socket exchanged after reconnect (obsolete)

EMRAS_CLNTDISC

0x98110197: RAS: Client disconnect

EMRAS_ACCESS_NOT_FOUND

0x98110198: RAS: Access not configured for this call (e.g. SPoC access configuration missing)

EMRAS_TOKEN_MISSING

0x98110199: RAS: Token missing

EMRAS_TOKEN_INVALID

0x9811019A: RAS: Token invalid

EMRAS_TOKEN_DENIED

0x9811019B: RAS: Token denied

EMRAS_TLS_CERTIFICATE_ERROR

0x9811019C: RAS: TLS certificate error

EMRAS_TLS_PRIVATEKEY_ERROR

0x9811019D: RAS: TLS private key error

EMRAS_TLS_HANDSHAKE_FAILED

0x9811019E: RAS: TLS handshake failed

EMRAS_ERROR_LAST

EMRAS_ERROR_LAST.

4 Examples

4.1 CoeSdoUpload

```
public void CoeSdoUploadExample()
{
    ushort wSlaveAddress = 1012; // slave with station address 1012
    ushort wObjIndex = 0x1018; // object index 0x1018
    byte byObjSubIndex = 1; // subindex 1 (Vendor ID, which is from type UDINT with 4
    ↪ bytes)
    uint dwDataLen = 4; // 4 bytes
    byte[] pbyData = new byte[dwDataLen];
    uint dwOutDataLen = 0;
    uint dwTimeout = 5000; // timeout 5s
    uint dwFlags = (uint)EMailBoxFlags.NONE; // no complete access

    uint dwSlaveId = m_oEcWrapper.GetSlaveId(wSlaveAddress);
    ECErrors eRes = m_oEcWrapper.CoeSdoUpload(dwSlaveId, wObjIndex, byObjSubIndex,
    ↪ pbyData, dwDataLen, out dwOutDataLen, dwTimeout, dwFlags);
    if (eRes != ECErrors.EC_NOERROR)
    {
        LogMasterError("CoeSdoUpload failed: ", eRes);
        return;
    }

    string szValue =
    ↪ CECWrapperDotNet.ConvValueToString(DN_EC_T_DEFTYPE.ARRAY_OF_BYTE, pbyData);
    LogInfo("CoeSdoUpload success: " + szValue); // 02 00 00 00
}
```

4.2 CoeSdoDownload

```
public void CoeSdoDownloadExample()
{
    ushort wSlaveAddress = 1012; // slave with station address 1012
    ushort wObjIndex = 0x40A2; // object index 0x40A2
    byte byObjSubIndex = 2; // subindex 2
    uint dwValue = 123; // value 123
    uint dwTimeout = 5000; // timeout 5s
    uint dwFlags = (uint)EMailBoxFlags.NONE; // no complete access

    // convert UNSIGNED32 value to 4 bytes
    byte[] pbyData = null;
    ECErrors eRes = CECWrapperDotNet.ConvValueToBytes(DN_EC_T_DEFTYPE.UNSIGNED32,
    ↪ dwValue, out pbyData);
    if (eRes != ECErrors.EC_NOERROR)
    {
        LogMasterError("ConvValueToBytes failed: ", eRes);
        return;
    }

    uint dwSlaveId = m_oEcWrapper.GetSlaveId(wSlaveAddress);
    eRes = m_oEcWrapper.CoeSdoDownload(dwSlaveId, wObjIndex, byObjSubIndex, pbyData,
    ↪
```

(continues on next page)

(continued from previous page)

```

↪(uint)pbyData.Length, dwTimeout, dwFlags);
if (eRes != EError.EC_NOERROR)
{
    LogMasterError("CoeSdoDownload failed: ", eRes);
    return;
}

LogInfo("CoeSdoDownload success");
}

```

4.3 ReadSlaveEEPROM

```

public void ReadSlaveEEPROMExample()
{
    bool bFixedAddressing = true; // enable fixed addressing
    ushort wSlaveAddress = 1006; // slave with station address 1006
    ushort wEEPROMStartOffset = 0x0; // offset 0
    uint dwReadLen = 10; // 10 bytes
    byte[] pwReadData = new byte[dwReadLen];
    uint dwNumOutData = 0;
    uint dwTimeout = 2000; // timeout 2s

    EError eRes = m_oEcWrapper.ReadSlaveEEPROM(bFixedAddressing, wSlaveAddress, ↪
    ↪wEEPROMStartOffset, pwReadData, dwReadLen, out dwNumOutData, dwTimeout);
    if (eRes != EError.EC_NOERROR)
    {
        LogMasterError("ReadSlaveEEPROM failed: ", eRes);
        return;
    }

    string szValue = ↪
    ↪CECWrapperDotNet.ConvValueToString(DN_EC_T_DEFTYPE.ARRAY_OF_BYTE, pwReadData);
    LogInfo("ReadSlaveEEPROM success: " + szValue);
}

```

4.4 FoeFileDownload

```

public void FoeFileDownloadExample()
{
    ushort wSlaveAddress = 1012; // slave with station address 1012
    string szFileName = "foe.txt"; // filename of file in slave
    byte[] pbyData = new byte[] { 1, 2, 3, 4 }; // 4 bytes
    uint dwPassword = 0; // password is 0
    uint dwTimeout = 2000; // timeout 2s

    uint dwSlaveId = m_oEcWrapper.GetSlaveId(wSlaveAddress);
    EError eRes = m_oEcWrapper.FoeFileDownload(dwSlaveId, szFileName, ↪
    ↪(uint)szFileName.Length, pbyData, (uint)pbyData.Length, dwPassword, dwTimeout);
    if (eRes != EError.EC_NOERROR)
    {
        LogMasterError("FoeFileDownload failed: ", eRes);
        return;
    }
}

```

(continues on next page)

(continued from previous page)

```
LogInfo("FoeFileDownload success");  
}
```

5 FAQ

I installed .NET and the demo crashes with strange errors. What can I do?

This might be a problem of mixing x86 with x64 binaries. Verify that if you have installed the runtime for x64 bit, please install also EC-Master for x64 bit.