



acontis technologies GmbH

SOFTWARE

EC-Master

EtherCAT® Master Stack Class A

Version 3.1

Edition: 2020-10-22

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.

Content - compact

1	Synchronization with Distributed Clocks (DC)	6
1.1	Introduction	6
1.2	Configuration with ET9000	8
1.3	Configuration with EC-Engineer	11
1.4	Programmer's Guide	13
2	Master synchronization (DCM):	20
2.1	Introduction	20
3	File access over EtherCAT (FoE)	42
3.1	Specification	42
3.2	Programmer's Guide	46
4	Vendor specific protocol over EtherCAT (VoE)	54
4.1	Specification	54
4.2	Programmer's Guide	54
5	Automation Device Specification over EtherCAT (AoE)	58
5.1	Specification	58
5.2	Programmer's Guide	58

Content

1	Synchronization with Distributed Clocks (DC)	6
1.1	Introduction	6
1.1.1	Specification	6
1.1.2	DCM Modes	6
1.1.3	Support slaves and topologies	6
1.1.4	Typical Bus Timing	7
1.1.5	Implementation Details	7
1.1.6	Slaves in sync	8
1.1.7	Sync Window Monitoring	8
1.2	Configuration with ET9000	8
1.2.1	Enable "DC Mode" for slave	8
1.2.2	Enable "Sync Window Monitoring" for master	10
1.3	Configuration with EC-Engineer	11
1.3.1	Distributed Clocks Master settings (Expert)	11
1.3.2	Distributed Clocks Slave settings (Expert)	12
1.4	Programmer's Guide	13
1.4.1	ecatDcConfigure	13
1.4.2	ecatDclsEnabled	14
1.4.3	ecatGetBusTime	14
1.4.4	ecatDcContDelayCompEnable	15
1.4.5	ecatDcContDelayCompDisable	15
1.4.6	ecatIoControl – EC_IOCTL_DC_SLV_SYNC_STATUS_GET	15
1.4.7	ecatIoControl – EC_IOCTL_DC_FIRST_DC_SLV_AS_REF_CLOCK	16
1.4.8	ecatNotify – EC_NOTIFY_REFCLOCK_PRESENCE	16
1.4.9	ecatNotify – EC_NOTIFY_DC_STATUS	17
1.4.10	ecatNotify – EC_NOTIFY_DC_SLV_SYNC	18

1.4.11	ecatFindInpVarByName – “Inputs.BusTime”	19
1.4.12	ecatIoControl – EC_IOCTL_DC_ENABLE_ALL_DC_SLV	19
2	Master synchronization (DCM):	20
2.1	Introduction	20
2.1.1	DCM in sync	20
2.1.2	Configuration with ET9000	21
2.1.3	Controller adjustment	22
2.1.4	DCM Master Shift mode	25
2.1.5	DCM Master Ref Clock mode	25
2.1.6	DCM Linklayer Ref Clock mode	25
2.1.7	Programmer's Guide	26
2.1.7.1	ecatDcmConfigure	26
2.1.7.2	ecatDcmGetStatus	30
2.1.7.3	ecatDcmResetStatus	31
2.1.7.4	ecatDcmGetBusShiftConfigured	31
2.1.7.5	ecatDcmGetLog	31
2.1.7.6	ecatDcmShowStatus	31
2.1.7.7	ecatDcmGetAdjust	32
2.1.7.8	API returned status values	32
2.1.7.9	Notifications	32
2.1.7.10	Sync signal activation	33
2.1.7.11	ecatNotify – EC_NOTIFY_DCM_SYNC	33
2.1.8	Example code	34
2.1.9	DCM on Windows	35
2.1.9.1	Purpose of the ECAT Driver	35
2.1.9.2	Prerequisites of the ECAT Driver	35
2.1.9.3	Installing and configuring the ECAT Driver on Windows	36
2.1.9.4	Enable Realtime Priority Class	41
3	File access over EtherCAT (FoE)	42
3.1	Specification	42
3.1.1	FoE Protocol	42
3.1.1.1	FoE file download	43
3.1.1.2	FoE file upload	45
3.1.2	Boot State	46
3.2	Programmer's Guide	46
3.2.1	ecatFoeFileDownload	46
3.2.2	ecatFoeFileUpload	48
3.2.3	ecatFoeDownloadReq	49
3.2.4	ecatFoeSegmentedDownloadReq	49
3.2.5	ecatFoeUploadReq	51
3.2.6	ecatFoeSegmentedUploadReq	52
3.2.7	Notifications	53
3.2.7.1	EC_NOTIFY_FOE_MBXSEND_WKC_ERROR	53
3.2.7.2	EC_NOTIFY_FOE_MBSLAVE_ERROR	53
3.2.7.3	Extending EC_T_MBX_DATA	53
4	Vendor specific protocol over EtherCAT (VoE)	54
4.1	Specification	54
4.2	Programmer's Guide	54

4.2.1	ecatVoeWrite.....	54
4.2.2	ecatVoeWriteReq.....	55
4.2.3	ecatNotify – eMbxTferType_VOE_WRITE	55
4.2.4	ecatVoeRead	56
4.2.5	ecatNotify – eMbxTferType_VOE_READ	57
5	Automation Device Specification over EtherCAT (AoE)	58
5.1	Specification	58
5.2	Programmer's Guide	58
5.2.1	Current limitations	58
5.2.2	ecatAoeGetSlaveNetId	58
5.2.3	ecatAoeRead	59
5.2.4	ecatAoeReadReq.....	60
5.2.5	ecatAoeWrite.....	61
5.2.6	ecatAoeWriteReq.....	62
5.2.7	ecatAoeReadWrite.....	63
5.2.8	ecatAoeWriteControl.....	64
5.2.9	ecatNotify – eMbxTferType_AOE_READ	65
5.2.10	ecatNotify – eMbxTferType_AOE_WRITE	65

1 Synchronization with Distributed Clocks (DC)

1.1 Introduction

DC clock synchronization enables all EtherCAT devices (master and slaves) to share the same EtherCAT System Time (see document “ESC Datasheet Section I”).

1.1.1 Specification

“DC-slave” is defined as slave who shall be synchronized by means of distributed clocks.

During network start-up several steps have to be performed by the EC-Master to set-up a consistent time base in all DC-slaves:

- Initial propagation delay measurement and compensation (ETG.8000)
- Offset compensation (ETG.8000)
- Set start time (ETG.8000)
- After network start-up: continuous drift compensation (ETG.8000)
- The Master must synchronize itself on the reference clock (ETG.1020) → DCM

References

ETG.1000.3 and ETG.1000.4
ETG.1020 → Synchronization
ETG.8000 → Distributed Clocks

1.1.2 DCM Modes

The following DCM Modes are available:

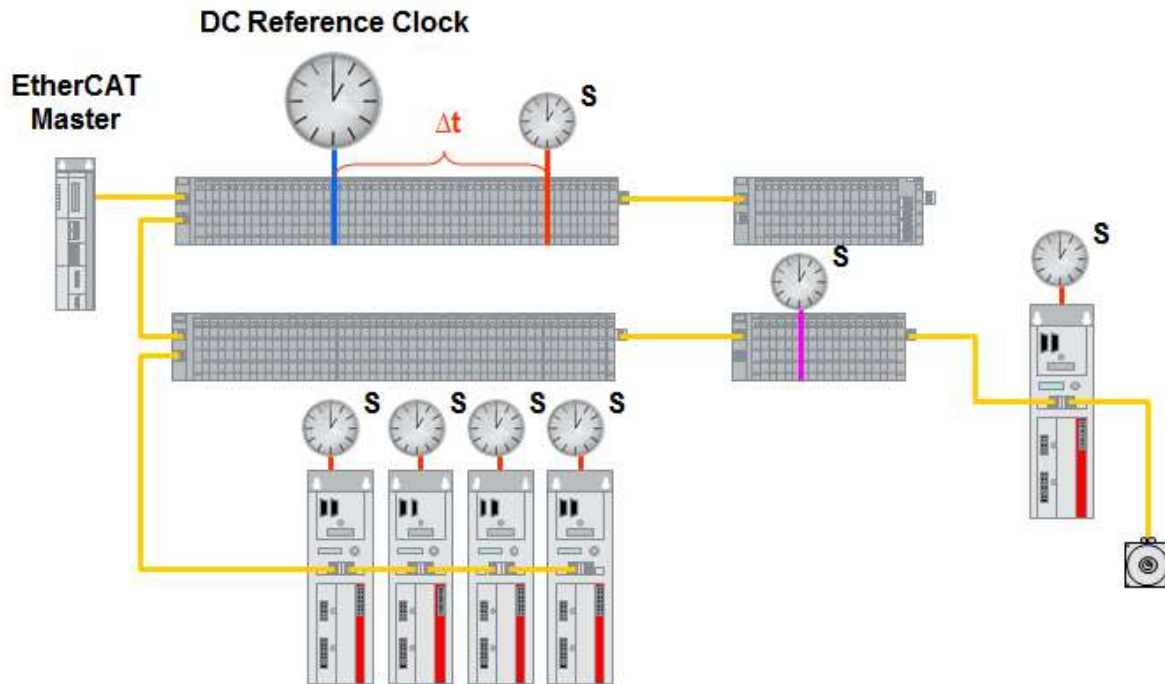
Name	Purpose
BusShift	Synchronize slaves to master timer (Default)
MasterShift	Synchronize master timer to slave. Feasibility depending on target HW and SW.
LinkLayerRefClock	Bus Shift using Link Layer clock. Special HW needed.
MasterRefClock	Bus Shift excluding reference clock controlling. Lowers CPU usages, but very high timer accuracy needed.
DCX	Synchronization of two or more EtherCAT segments by a bridge device. Only available with Feature Pack External Synchronization.

See also 2.1.7.1 “ecatDcmConfigure”.

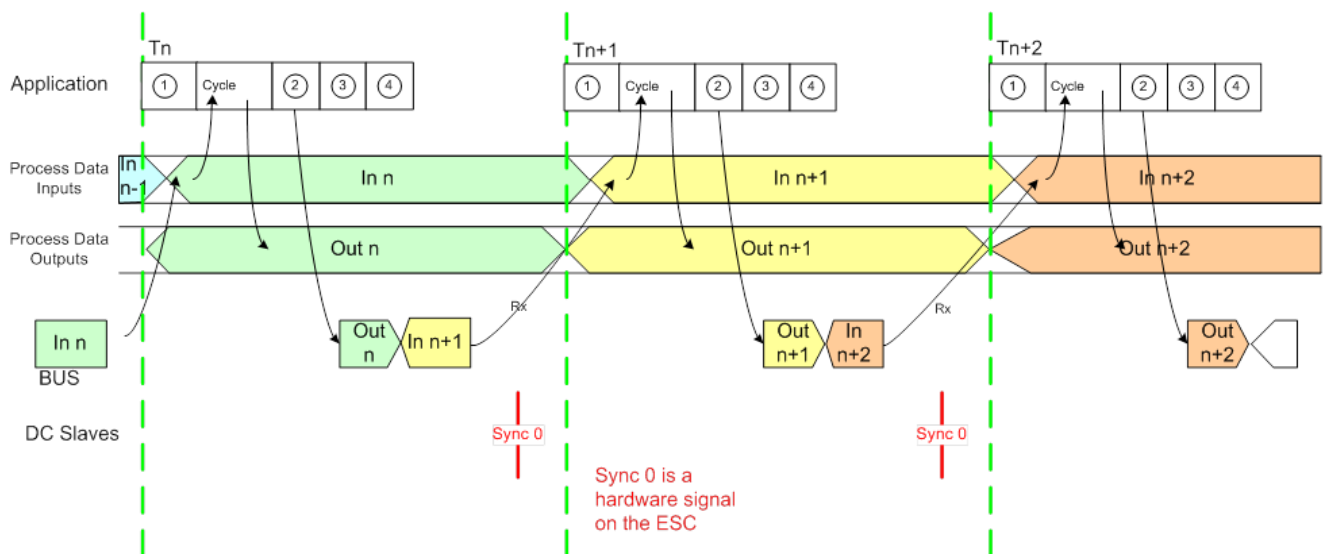
1.1.3 Support slaves and topologies

EC-Master supports all currently existing slave types and possible topologies:

- Slaves with 32 bit or 64 bit system time register (0x0910)
- Reference Clock with 32 bit or 64 bit system time register (0x0910)
- Reference Clock as first slave (auto increment address 0) or in the middle. Only slaves behind the reference clock could be synchronized.
- Drift compensation with 32 bit or 64 bit ARMW command in the cyclic frame
- Topologies: Line, Daisy chain, Daisy chain with drop lines, tree structure, star topology with EK1122 junctions



1.1.4 Typical Bus Timing



1.1.5 Implementation Details

The generation of Sync impulses is started after initialization and sending of 8000 FRMW frames which are required to bring slaves initially into sync. After this FRMW generation a grace period of 50 ms under real-time OSES is used and 500 ms using Windows is configured before start of cyclic operation of slaves, which causes the Sync impulse generation delayed by this grace period. Initial propagation delay measurement and offset compensation commands are not part of the ENI file. The ARMW command for drift compensation is part of the cyclic frame.

1.1.6 Slaves in sync

Slaves in sync means that the system time difference of all DC slaves do not exceed a configured limit. Out of sync is detected individually and immediately for each slaves.

The master awaits that the slaves are in sync in Master state transition INIT->PREOP. Therefore the master state transition may timeout if the slaves do not get in sync.

Due to technology the slaves are always getting in sync as long as there is no error in setup. In order to detect system time difference exceeding, **Sync Window Monitoring** is used, see below.

1.1.7 Sync Window Monitoring

Sync Window Monitoring must be explicitly enabled in configuration.

The system time difference exceeding detection in **Sync Window Monitoring** uses a deviation limit and a settle time and is issued continuously with configured commands (Ado 0x092C) in cyclic frames.

In sync is assumed if there is no violation of the system time difference limit (for all DC slaves!) detected within the settle time.

The deviation limit (`dwDevLimit`) and settle time (`dwSettleTime`) can be configured using `ecatDcConfigure()`.

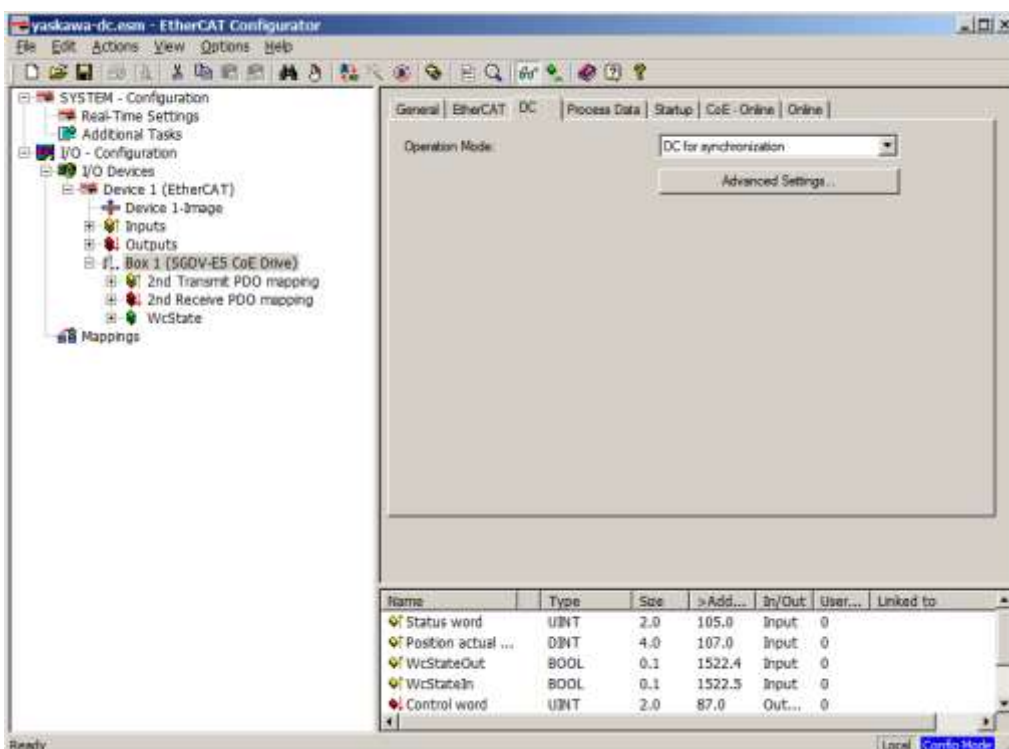
If the configuration only contains the cyclic commands for SAFE-OP or OP (e.g. ET9000) the master queues acyclic datagrams (Ado 0x092C) for system time difference measurement.

If there are at less than two DC slaves on bus (e.g. if the reference clock is the only DC slave on bus), **Sync Window Monitoring** is skipped. If it is skipped, because it is not enabled in configuration or there are less than two DC slaves on bus, slaves are immediately considered in sync.

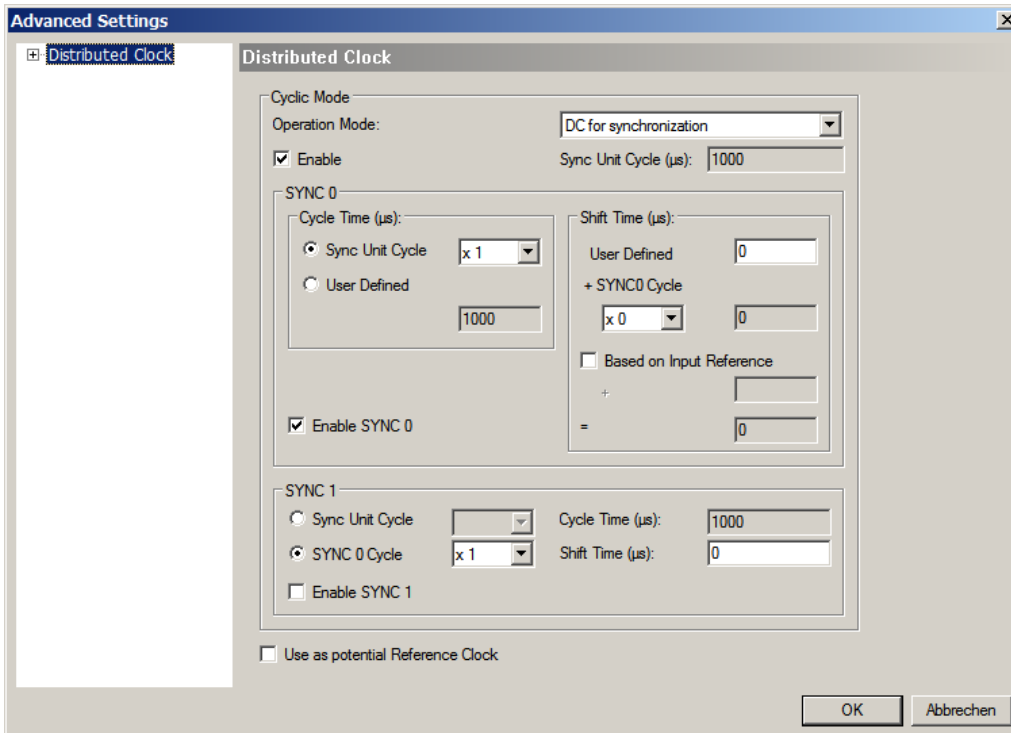
1.2 Configuration with ET9000

1.2.1 Enable “DC Mode” for slave

If a slave supports DC, an additional tab in ET9000 appears.



In the “Advanced Settings” additional slave specific parameters may be set. By default the cycle time for “SYNC 0” is equal to the bus cycle time (Sync Unit Cycle).

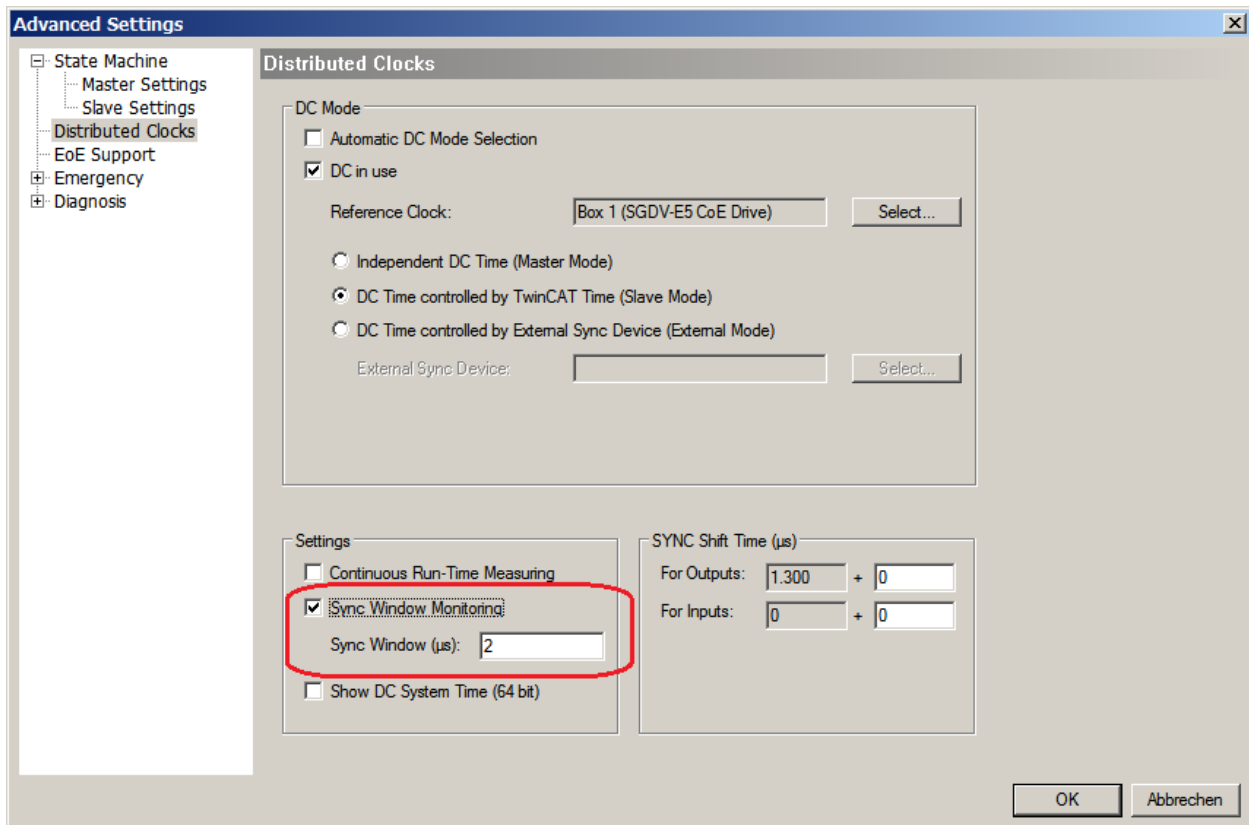


The screenshot shows the 'Advanced Settings' dialog box with the 'Distributed Clock' tab selected. The 'Cyclic Mode' section has 'Operation Mode' set to 'DC for synchronization' and 'Enable' checked. The 'Sync Unit Cycle (µs)' is set to 1000. The 'SYNC 0' section has 'Cycle Time (µs)' set to 'Sync Unit Cycle' (x 1) and 'User Defined' (1000). The 'Shift Time (µs)' section has 'User Defined' (0) and '+ SYNC0 Cycle' (x 0, 0). The 'Based on Input Reference' checkbox is unchecked. The 'SYNC 1' section has 'Cycle Time (µs)' set to 'Sync Unit Cycle' (x 1) and 'User Defined' (1000). The 'Shift Time (µs)' is set to 0. The 'Enable SYNC 1' checkbox is unchecked. The 'Use as potential Reference Clock' checkbox is unchecked. The 'OK' and 'Abbrechen' buttons are at the bottom right.

Section	Parameter	Value
Cyclic Mode	Operation Mode	DC for synchronization
	Enable	Checked
Sync Unit Cycle	Sync Unit Cycle (µs)	1000
	Enable SYNC 0	Checked
SYNC 0	Cycle Time (µs)	Sync Unit Cycle (x 1)
	User Defined	1000
Shift Time (µs)	User Defined	0
	+ SYNC0 Cycle	x 0, 0
Based on Input Reference	Based on Input Reference	Unchecked
	+	
SYNC 1	Cycle Time (µs)	Sync Unit Cycle (x 1)
	User Defined	1000
Shift Time (µs)	Shift Time (µs)	0
	Enable SYNC 1	Unchecked
Use as potential Reference Clock	Use as potential Reference Clock	Unchecked
	OK / Abbrechen	Buttons

1.2.2 Enable “Sync Window Monitoring” for master

By enabling the option “Sync Window Monitoring” in the “Advanced Settings” of the master, the EtherCAT configurator will insert a command (datagram) in the cyclic frame to read the ESC registers 0x092C. If this is selected the master will throw the notification EC_NOTIFY_DC_SLV_SYNC.



1.3 Configuration with EC-Engineer

The EC-Engineer automatically chooses the DC settings for slaves as proposed by the device's vendor and sets DCM mode to bus shift.

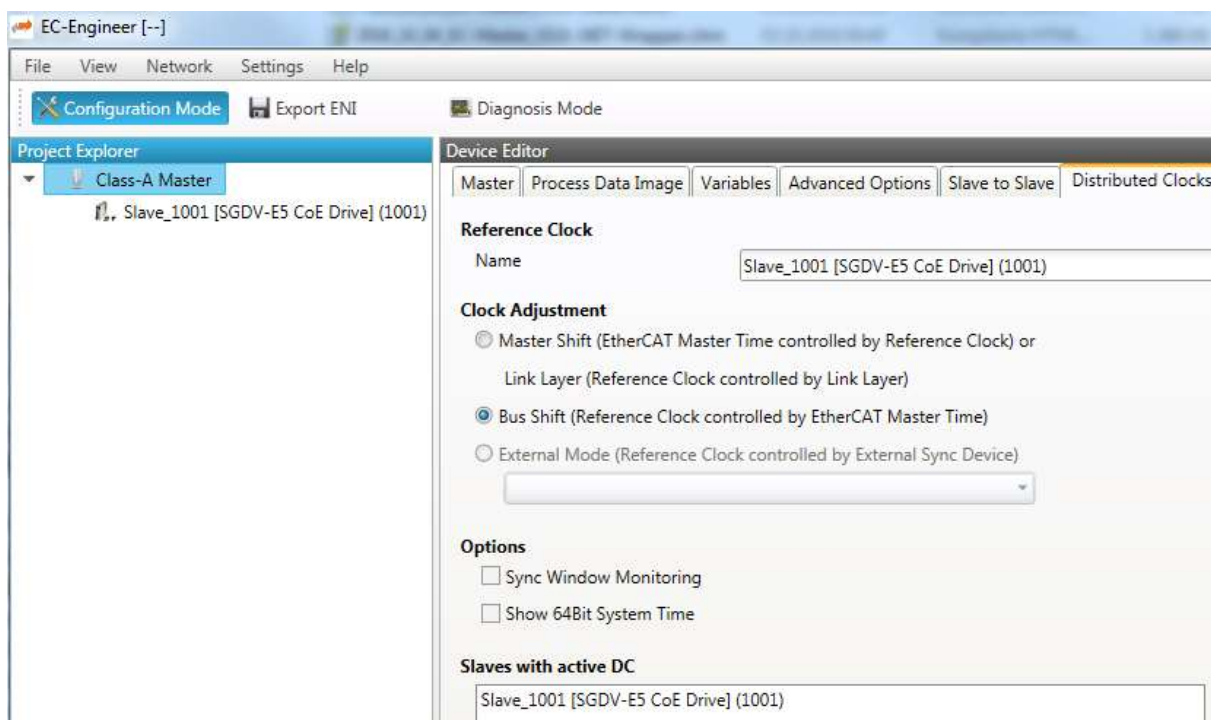
The settings can be changed according to the project's needs. **DC options are part of the EC-Engineer Expert Mode.** The *Expert Mode* can be activated from the menu:



Please refer to the EC-Engineer Manual for detailed description.

1.3.1 Distributed Clocks Master settings (Expert)

In this tab, the user can change distributed clock related settings. The tab is only available if the configuration contains slaves.



Reference Clock Name: Name of the reference clock. By default, this is the first slave with DC support.

Master Shift: The reference clock controls the Master time

Bus Shift: The Master time controls the reference clock

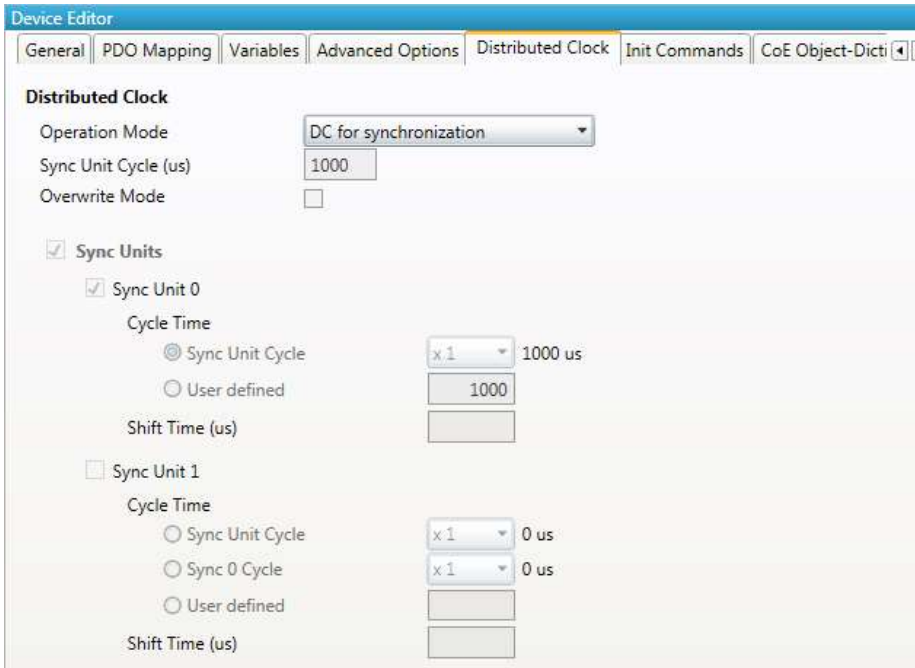
Continuous Propagation Compensation: A command (datagram) will be inserted in the Cyclic frame which allows the EtherCAT master to measure and compensate the propagation delay time by time.

Sync Window Monitoring: A command (datagram) will be inserted in the cyclic frame to read the ESC registers 0x092C. If this is selected the master will throw a notification.

Show 64Bit System Time: Master supports slaves with 32bit and 64bit system time register (0x0910). If this is selected he will interpret it as 64bit system time.

1.3.2 Distributed Clocks Slave settings (Expert)

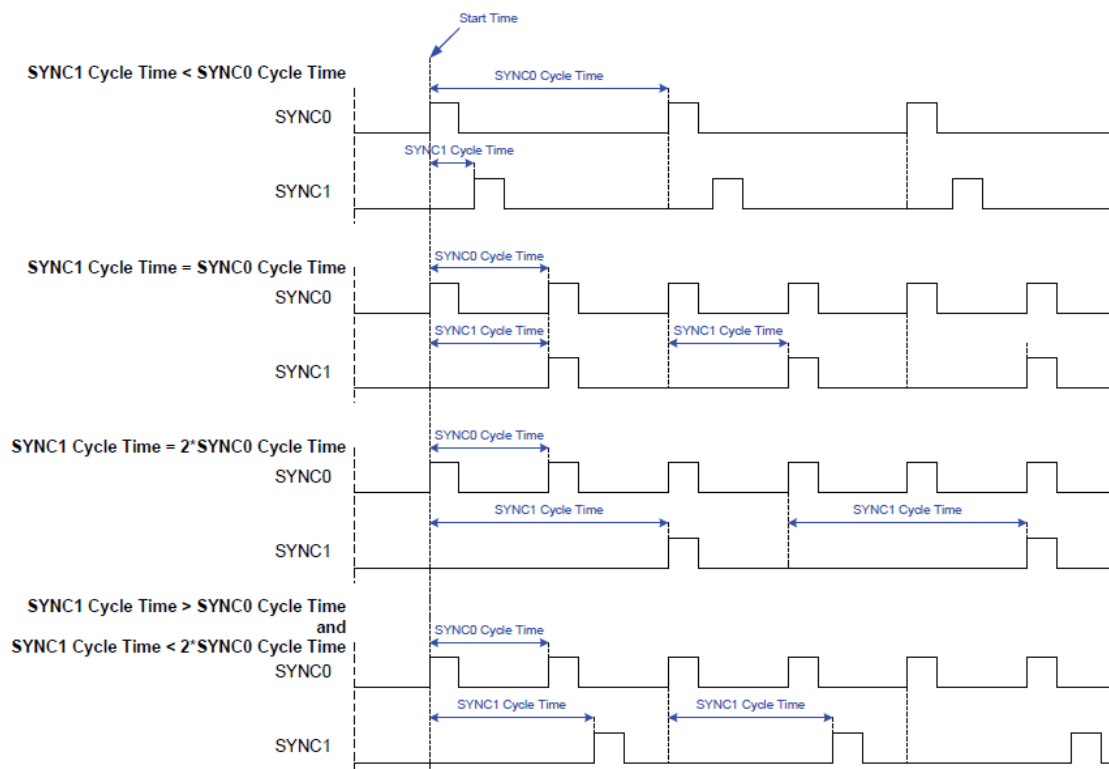
In this tab, the user can change distributed clock related settings. The tab is only available if the device's vendor specified the DC usage. **Sync signal generation** or **DC latching** is selected automatically according to **Operation Mode**.



Operation Mode: Selectable DC operation modes. The modes cannot be edited.

Sync Unit Cycle: Base interval in microseconds which will be used from master (see also chapter 5.2.1 in the EC-Engineer User Manual).

The Sync Units can be activated and configured to generate signals. The timing is described in chapter 9.2.3 **SyncSignal Generation** in the **ET1100 Datasheet**:



1.4 Programmer's Guide

1.4.1 ecatDcConfigure

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.
(See Chapter "Drift Compensation" of the ETG Document "ESC Datasheet Section 1 – Technology")

This function can be called any time after [ecatConfigureMaster\(\)](#). Usually it is issued before [ecatStart](#).

```
EC_T_DWORD ecatDcConfigure(
    EC_T_DC_CONFIGURE* pDcConfigure
);
```

Parameters

pDcConfigure

[in] Configuration parameter a pointer to a structure of type EC_T_DC_CONFIGURE.

Return

EC_E_NOERROR or error code

Comment

```
typedef struct _EC_T_DC_CONFIGURE
{
    EC_T_DWORD          dwCIntId;
    EC_T_DWORD          dwTimeout;
    EC_T_DWORD          dwDevLimit;
    EC_T_DWORD          dwSettleTime;
    EC_T_DWORD          dwTotalBurstLength;
    EC_T_DWORD          dwBurstBulk;
    EC_T_BOOL           bBulkInLinkLayer;
    EC_T_BOOL           bAcycDistributionDisabled;
    EC_T_DWORD          dwDcStartTimeGrid;
    EC_BOOL             bDcInitBeforeSlaveStateChange;
    EC_T_DWORD          dwReserved[4];
} EC_T_DC_CONFIGURE, *EC_PT_DC_CONFIGURE;
```

Description

dwCIntId

[in] Reserved.

dwTimeout ([ms])

[in] Is used for timeout in DC initial synchronization approach where Time Offsets and Propagation Delays are evaluated.

dwDevLimit

[in] Maximum allowed deviation of the individual slave clock and the DC reference clock. The check against the limit is only active if the configuration tool (ET9000) the option "Sync Window Monitoring" is active. By setting this option the tool configures a BRD command to read slave register 0x092C in each cycle. Using this parameter the maximum value of the wire or'ed slave deviation can be set to determine whether the slave clocks are within this range.:

dwDevLimit	Limit
0	disable no "Sync Window Monitoring"
1	1 nsec
2	3 nsec

3	7 nsec
4	15 nsec
5	31 nsec
6	63 nsec
7	127 nsec
8	254 nsec
9	511 nsec
10	1023 nsec
n	$2^n - 1$ nsec

dwSettleTime ([ms])
[in] When the master starts synchronization of the slave clocks there is some time were the slaves will be oscillating around the final optimal point. This would lead to multiple notifications of the clocks being in sync and being out of sync until finally being in sync. To avoid multiple notifications the client may specify a settle time. The master will only notify the client if within this settle time no clock will run out-of-sync.

dwTotalBurstLength
[in] Overall amount of burst frames sent. Default 10000.

dwBurstBulk
[in] Amount of burst frames to send before waiting for responses. Default 12.

bBulkInLinkLayer
[in] If EC_TRUE, bulk is realized by link layer (link layer must support it), otherwise by master. The MAC needs to support the frame repeating function. In this case the link layer will repeat the DC burst frames itself, reducing the hardware accesses of the master to the MAC.

bAcycDistributionDisabled
[in] If EC_TRUE, acyclic distribution of DC time is disabled

dwDcStartTimeGrid
[in] Time grid in nsec to align DC start time. Setting this value help by synchronizing several EtherCAT networks together to eliminate the random shift value between the SYNC signals.

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

1.4.2 *ecatDclsEnabled*

This function checks if DC is enabled and used. This function can be called any time after [ecatConfigureMaster\(\)](#).

EC_T_DWORD *ecatDclsEnabled* (
 EC_T_BOOL* **pbDclsEnabled**);

Parameters

pbDclsEnabled
[out] Set to EC_TRUE if DC is enabled and used.

Return

EC_E_NOERROR or error code.

1.4.3 *ecatGetBusTime*

This function returns the actual bus time in nano seconds.

EC_T_DWORD *ecatGetBusTime* (
 EC_T_UINT*64 **pqwBusTime**);

Parameters

pqwBusTime
[out] Bus time [ns]

Return

EC_E_NOERROR or error code

1.4.4 *ecatDcContDelayCompEnable*

Enable the continuous propagation delay compensation. This function can be called any time after [ecatConfigureMaster\(\)](#).

EC_T_DWORD ecatDcContDelayCompEnable (EC_T_VOID);

Parameters

-

Return

EC_E_NOERROR or error code

Comment

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

1.4.5 *ecatDcContDelayCompDisable*

Disable the continuous propagation delay compensation previously enabled by **ecatDcContDelayCompEnable**.

EC_T_DWORD ecatDcContDelayCompDisable (EC_T_VOID);

Parameters

-

Return

EC_E_NOERROR or error code

1.4.6 *ecatIoControl – EC_IOCTL_DC_SLV_SYNC_STATUS_GET*

Get the last generated *EC_NOTIFY_DC_SLV_SYNC* notification.

Parameters

pbyInBuf

[] Should be set to *EC_NULL*.

dwInBufSize

[] Should be set to 0.

pbyOutBuf

[out] pointer to *EC_T_DC_SYNC_NOTIFY_DESC* data type.

dwOutBufSize

[in] Size of the output buffer in bytes.

pdwNumOutData

[out] Pointer to *EC_T_DWORD*. Amount of bytes written to the output buffer *pbyOutBuf*.

Comment

Chapter 1.4.10 “*EC_NOTIFY_DC_SLV_SYNC*” describes *EC_T_DC_SYNC_NOTIFY_DESC*.

Return

EC_E_NOERROR or error code

1.4.7 *ecatIoctl* – *EC_IOCTL_DC_FIRST_DC_SLV_AS_REF_CLOCK*

Enable or disable the usage of the first DC slave on bus overriding the configured reference clock.

Parameters

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

Return

EC_E_NOERROR or error code

1.4.8 *ecatNotify* – *EC_NOTIFY_REFCLOCK_PRESENCE*

Distributed clocks reference clock presence notification. It will be received before EC_NOTIFY_DC_SLV_SYNC as soon as reference clock was found on bus or removed from bus.

This notification is disabled by default. See EC_IOCTL_SET_NOTIFICATION_ENABLED in document EC-Master_ClassB for how to control the activation.

Parameters

pbyInBuf
[in] pointer to notification descriptor EC_T_REFCLOCK_PRESENCE_NOTIFY_DESC
dwInBufSize
[in] sizeof(EC_T_REFCLOCK_PRESENCE_NOTIFY_DESC)
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[in] EC_NULL (not used).

Comment

EC_T_REFCLOCK_PRESENCE_NOTIFY_DESC

```
typedef struct _EC_T_REFCLOCK_PRESENCE_NOTIFY_DESC {  
    EC_T_BOOL          bPresent;  
    EC_T_SLAVE_PROP    SlaveProp;  
} EC_T_REFCLOCK_PRESENCE_NOTIFY_DESC;
```

Description

bPresent
[in] Reference clock present
SlaveProp
[in] slave properties if reference clock present

1.4.9 *ecatNotify* – **EC_NOTIFY_DC_STATUS**

Distributed clocks status notification. It will be received after EC_NOTIFY_DC_SLV_SYNC as soon as DC is initialized or topology change was done . After topology was changed it may be received without EC_NOTIFY_DC_SLV_SYNC if slaves did not get out of sync.

This notification is enabled by default. See EC_IOCTL_SET_NOTIFICATION_ENABLED in document EC-Master_ClassB for how to control the deactivation.

Parameters

pbyInBuf

[in] Pointer to EC_T_DWORD (EC_E_NOERROR on success, Error code otherwise)

dwInBufSize

[in] sizeof(EC_T_DWORD)

pbyOutBuf

[in] NULL (not used)

dwOutBufSize

[in] 0 (not used)

pdwNumOutData

[in] EC_NULL (not used)

Comment

If EC_E_NOERROR is returned and window monitoring is enabled, all slaves are in SYNC

1.4.10 ecatNotify – EC_NOTIFY_DC_SLV_SYNC

DC slave synchronization notification. Every time the slaves are coming in sync or getting out of sync the clients will be notified here.

This notification is enabled by default. See EC_IOCTL_SET_NOTIFICATION_ENABLED in document EC-Master_ClassB for how to control the deactivation.

Parameters

pbyInBuf
[in] pointer to notification descriptor EC_T_DC_SYNC_NTIFY_DESC
dwInBufSize
[in] sizeof(EC_T_DC_SYNC_NTIFY_DESC)
pbyOutBuf
[in] NULL (not used)
dwOutBufSize
[in] 0 (not used)
pdwNumOutData
[in] EC_NULL (not used)

Comment

The notification is raised in any case if any DC slaves are configured. Slaves can only be out of sync if **Sync Window Monitoring** is enabled otherwise they are considered in sync.

EC_T_DC_SYNC_NTIFY_DESC

```
typedef struct _EC_T_DC_SYNC_NTIFY_DESC{  
    EC_T_DWORD      IsInSync;  
    EC_T_DWORD      IsNegative;  
    EC_T_DWORD      dwDeviation;  
    EC_T_SLAVE_PROP  SlaveProp;  
} EC_T_DC_SYNC_NTIFY_DESC;
```

Description

IsInSync
[in] EC_TRUE : Wire or'ed deviation value meets limit requirements.
EC_FALSE: Wire or'ed deviation value does not meet limit requirements. The limit is set by ecatDcConfigure().

IsNegative
[in] EC_TRUE : deviation value is negative
EC_FALSE: deviation value is positive

dwDeviation
[in] wire or'ed deviation value in nanoseconds in case of "in sync"

SlaveProp
[in] slave properties in case of "not in sync"

1.4.11 *ecatFindInpVarByName* – “Inputs.BusTime”

The *DC system time* (written to ESC register 0x0910) is part of the process data with name “Inputs.BusTime”. See “2.2.6.18 *ecatFindInpVarByName*” in the EC-Master Class B documentation.

```
EC_T_DWORD ecatFindInpVarByName(  
    EC_T_CHAR*          szVariableName,  
    EC_T_PROCESS_VAR_INFO* pProcessVarInfoEntry) ;
```

Parameters

szVariableName
 [in] Variable name
pProcessVarInfoEntry
 [out] Process variable information entry

Return

EC_E_NOERROR or error code.

1.4.12 *ecatIoControl* – *EC_IOCTL_DC_ENABLE_ALL_DC_SLV*

Enable or disable the usage of DC at all supporting slaves on bus overriding the configured settings. Perhaps *EC_IOCTL_DC_FIRST_DC_SLV_AS_REF_CLOCK* is necessary to set the reference clock at an allowed position.

Parameters

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

Return

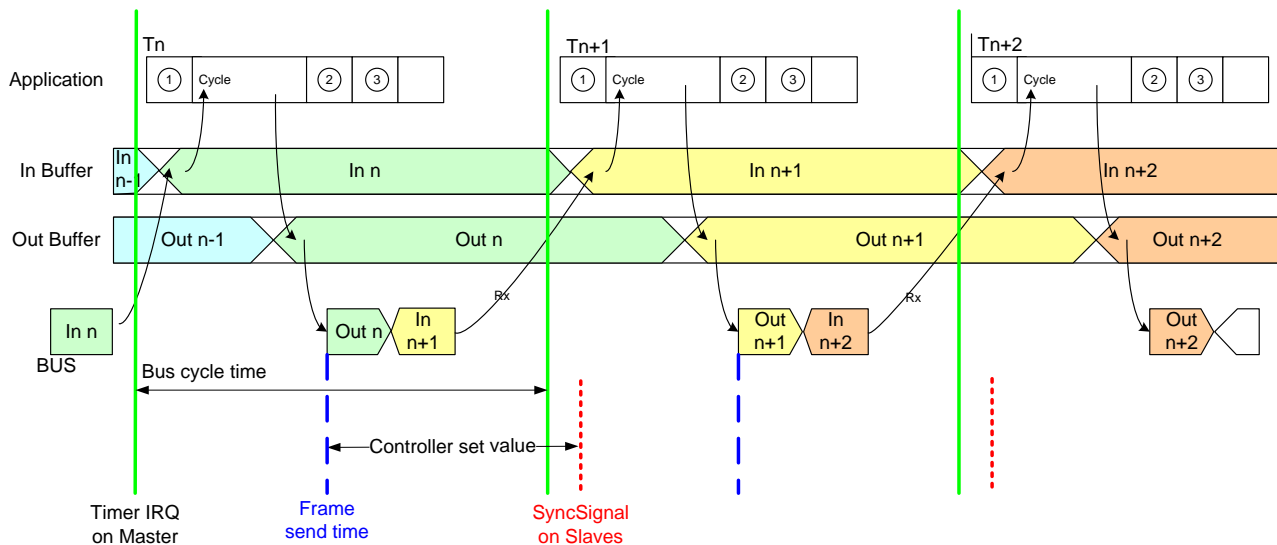
EC_E_NOERROR or error code

2 Master synchronization (DCM):

2.1 Introduction

In applications like motion control it is necessary that process data update and the slave SYNC pulses are correlated in timely behavior, because otherwise the SYNC Interrupt (on slave) used to apply new process data to the application would use new data on some slaves and old data on some other in case the current cyclic datagram (which updates process data) is on the flow during the SYNC Interrupt is raised on all slaves at the same time.

The Distributed Clock Master Synchronization (DCM) provides a controller mechanism to synchronize the process data update and the SYNC pulse in slaves.



The DC Master Synchronisation (DCM) in BusShift mode adjusts the bus time register of the DC reference clock. All the DC slaves converge to this time. This mode is useful to synchronize multiple EtherCAT busses or if adjustment of Master timer is not possible.

Features:

- PI drift controller
- Automatic timer adjustment error determination (I controller)

2.1.1 DCM in sync

DCM in sync means that the synchronization between the send time of the cyclic frames and the system time of the reference clock was successful.

The master awaits that DCM is in sync in Master state transition PREOP->SAFEOP. Therefore the master state transition may timeout if DCM does not get in sync.

Due to the Master's DC implementation, DCM may get in sync in transition INIT->PREOP.

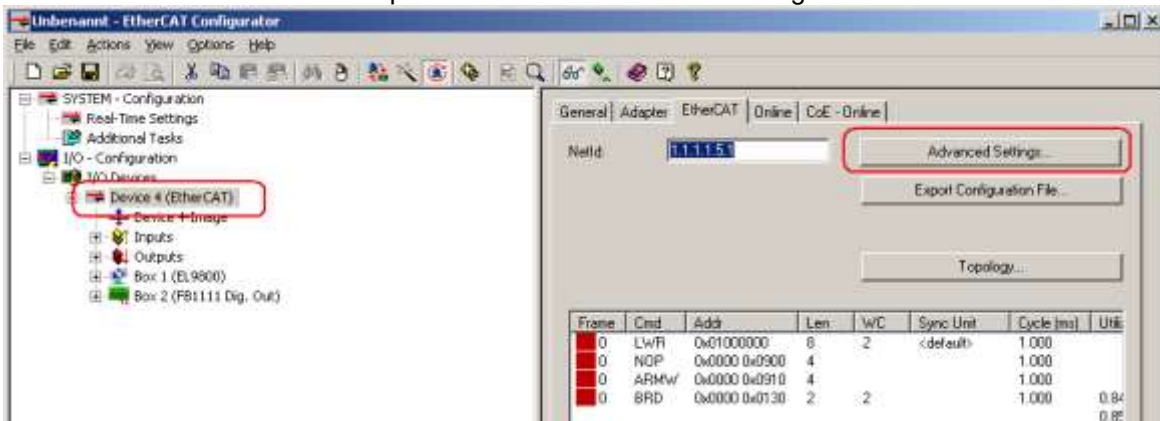
In sync is assumed if there is no error reported from the DCM controller within the settle time or if there is no DC slave connected.

2.1.2 Configuration with ET9000

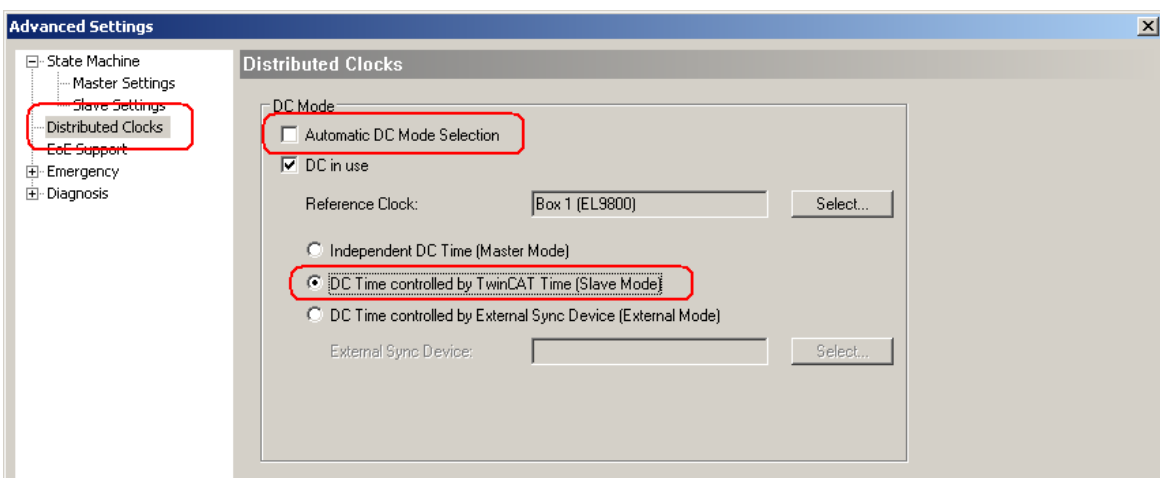
Since version 2.11.0 of the EtherCAT Configurator from Beckhoff explicitly setup whether the DC time shall be controlled by the EtherCAT master or not.

To create a DCM capable configuration, please accomplish the following steps.

1. Step: Scan the EtherCAT Bus
2. Select the EtherCAT device and press the button “Advanced Settings...”



3. In the open dialog please select “Distributed Clocks” on the left column. Then de-select “Automatic DC Mode Selection” and select the option “DC Time controlled by TwinCAT Time (Slave Mode)”.



4. Now the DC time can be controlled by the EtherCAT master.

Note: Don't forget to enable DC for the slaves.

2.1.3 Controller adjustment

To adjust the controller parameters the diagnostic values in file dcmlog0.0.csv can be used.
The generation of logging information can be enabled setting bLogEnabled to EC_TRUE with the function **ecatDcmConfigure()**, see 2.1.7.1 “ecatDcmConfigure”.

Controller log file description:

Column name	Description
Time[ms]	Controller execution timestamp
SyncSetVal [ns]	Controller set value (distance between frame send time and SYNC0)
BusTime [ns]	System Time
BusTimeOff	System Time modulo Sync Cycle Time
CtlAdj [ns]	Controller adjust value
CtlErr [ns]	Controller error (EC_T_DCM_SYNC_NTFY_DESC.nCtlErrorNsecCur)
CtlErrFilt	Filtered controller error
Drift [ppm]	Drift between local clock and DC reference clock
CtlErr [1/10 pmil]	
CtlOutSum	
CtlOutTot	
DCStartTime	DC start time send to the slaves to activate their SYNC signals
DCMErrorCode	Current error code of the DCM controller (same value as returned by emDcmGetStatus)
DCMInSync	Current InSync state of the DCM controller
DCInSync	Current InSync state of the DC slaves
SystemTimeDifference [ns]	Current system time difference of the DC slave if monitoring is enabled

Log file analyze:

To understand how the controller values correlates the following table can helps.

Controller error	Drift	Reference-Clock	Output
Positive	Must decrease	Must run faster	Double cycle time
Negative	Must increase	Must run slower	Half cycle time

The DCM Controller reacts if the controller error is positive or negative. E.g. on a positive controller error (CtlErr) the drift is too high and has to be decreased. Therefore the controller will speed up the reference clock.

In some case the drift is too high (*EcMasterDemoDc* shows error messages) and cannot be balanced by the controller. This holds if the drift is higher than about 400ppm.

Diagram 1: Bus offset in nanoseconds

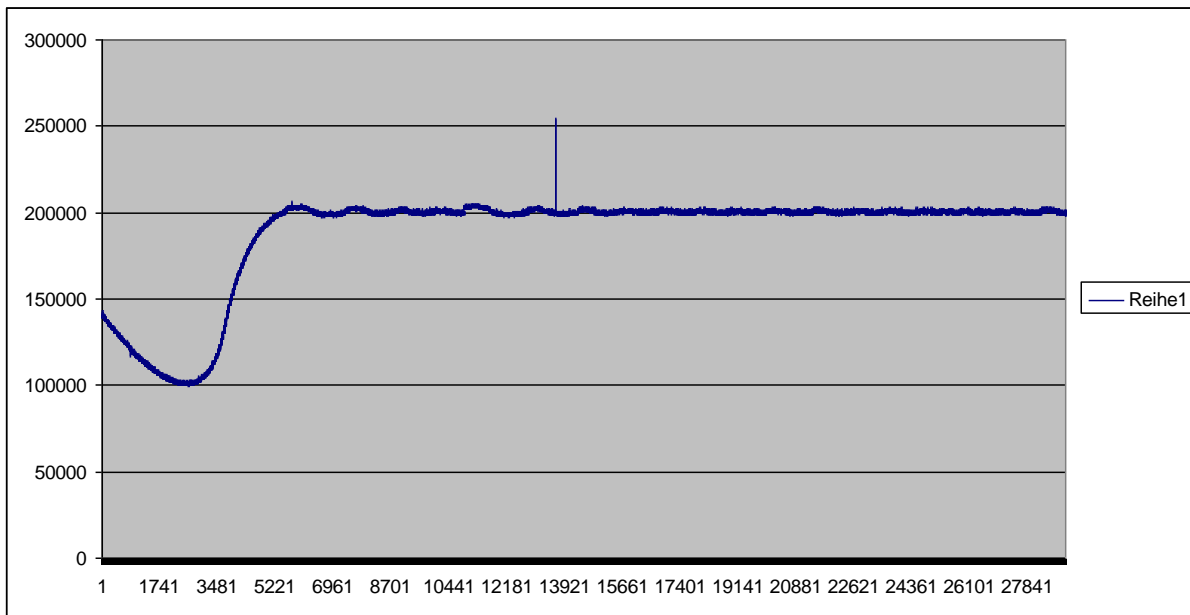


Diagram 2: Drift in ppm (part per million)

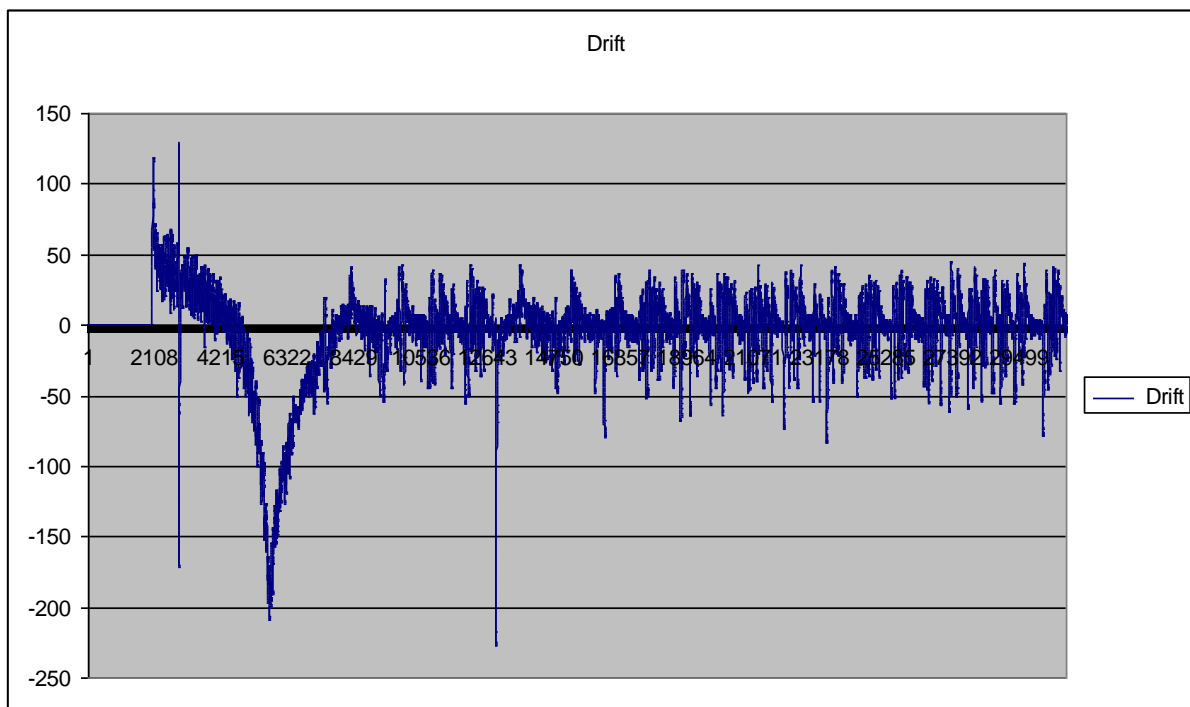


Diagram 3: Controller error in nanoseconds

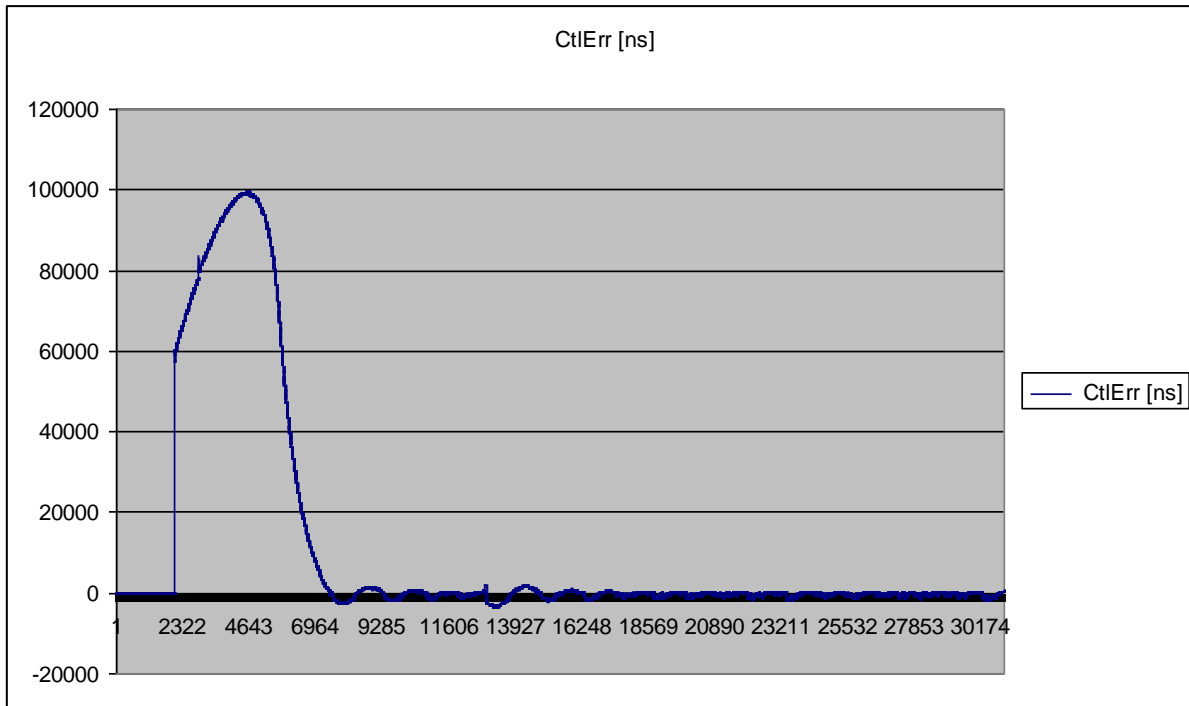
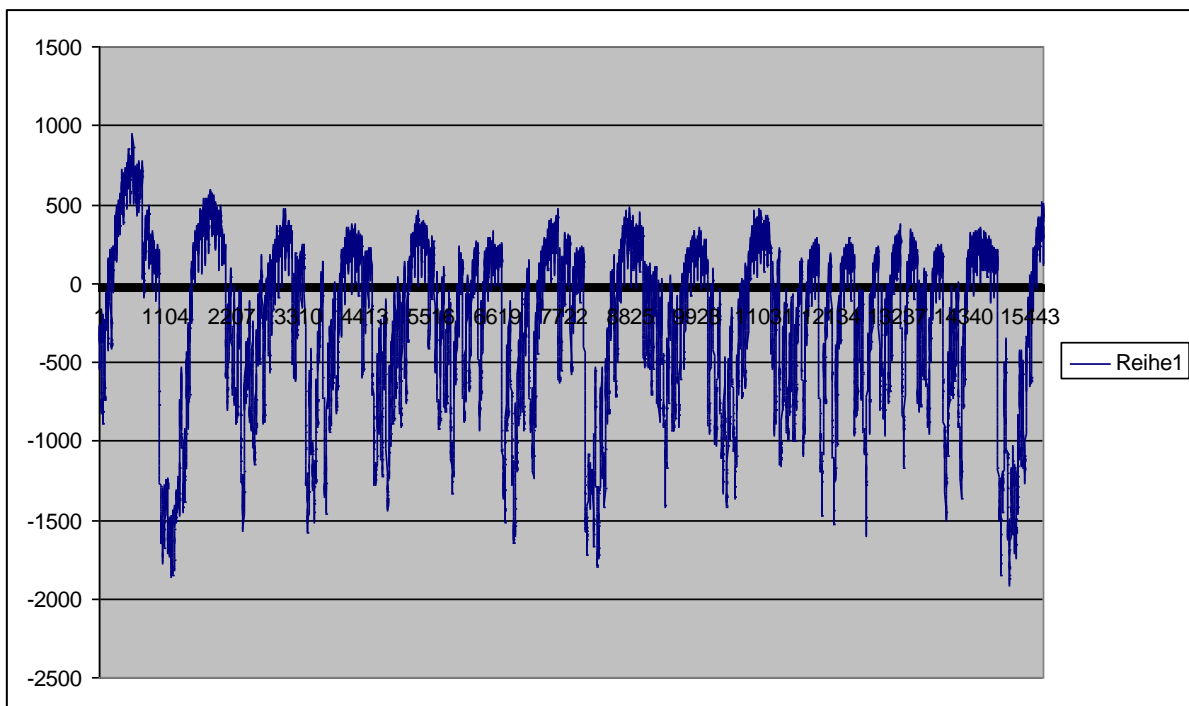


Diagram 4: Controller error in steady state in nanoseconds



Troubleshooting:

DCM BusShift needs a very deterministic and accurate time base.

The following statements have to be true:

- The timer input frequency must be determined with an accuracy greater than 600ppm (333333Hz vs 333000Hz. E.g. at 1ms, the cycle time must be between 999.400us and 1000.600us)
- The timer frequency must never change after the application start

On a PC platform the following settings have to be **disabled** in the BIOS. Be sure that these settings are really applied.

- System management interrupt
- Legacy USB support
- Intel C-STATE tech
- Intel Speedstep tech
- Spread Spectrum

2.1.4 DCM Master Shift mode

In this mode, the local time base will be adjusted to synchronize it with the network “bus time”. The following function pointers have to be implemented to enable the adjustment:

EC_T_OS_PARMS::pfHwTimerGetInputFrequency

EC_T_OS_PARMS::pfHwTimerModifyInitialCount

The master shift must not be enabled in the ENI file because it doesn't need any cyclic command.

This mode can be activate using `ecatDcmConfigure()`, see 2.1.7.1 “ecatDcmConfigure”.

2.1.5 DCM Master Ref Clock mode

The DCM Master Ref Clock mode is similar to the bus shift mode, without its control loop. This reduces the CPU load and makes it a good alternative for low performance CPU. Because of the missing control loop, the reaction time on disturbance is longer and the cycle must be very accurate.

The bus shift time must be enabled in the ENI file because it use the same cyclic command to synchronize the EtherCAT network with master system

This mode can be activate using `ecatDcmConfigure()`, see 2.1.7.1 “ecatDcmConfigure”.

2.1.6 DCM Linklayer Ref Clock mode

In this mode the link layer should provide the time base for the cyclic frames. **EC_LINKIOCTL_GETTIME** will be called during the DC initialization to initialize the DC related registers of the DC slaves and during the slave transition PREOP to SAFEOP to start the DC SYNC signals if needed.

EC_LINKIOCTL_GETTIME should return the current 64 bits value in nanosecond of a time counter running continuously.

During the call to **EcLinkSendFrame**, the link layer should insert the send time of the frame following the instruction given by **EC_T_LINK_FRAMEDESC::wTimestampOffset** and

EC_T_LINK_FRAMEDESC::wTimestampSize of the parameter **pLinkFrameDesc**.

A value of 0 means that no time stamp should be inserted.

2.1.7 Programmer's Guide

2.1.7.1 ecatDcmConfigure

DC Master Sync Controller configuration.

```
EC_T_DWORD ecatDcmConfigure (
    EC_T_DCM_CONFIG*      pDcmConfig
    EC_T_DWORD             dwInSyncTimeout
);
```

Parameters

pDcmConfig
[in] Configuration information, a pointer to a structure of type **EC_T_DCM_CONFIG**.

dwInSyncTimeout
[in] Currently not implemented.

Return

EC_E_NOERROR or error code

Comment

EC_T_DCM_CONFIG

This structure contains the configuration information for the DCM.

```
typedef struct _EC_T_DCM_CONFIG {
    EC_T_DCM_MODE eMode;
    union {
        EC_T_DCM_CONFIG_BUSSHIFT      BusShift;
        EC_T_DCM_CONFIG_MASTERSHIFT    MasterShift;
        EC_T_DCM_CONFIG_LINKLAYERREFCLOCK LinkLayerRefClock;
        EC_T_DCM_CONFIG_MASTERREFCLOCK MasterRefClock;
        EC_T_DCM_CONFIG_DCX            Dcx;
        EC_T_DWORD                     adwReserved[16];
    }
} EC_T_DCM_CONFIG;
```

EC_T_DCM_MODE

Enumerator for different DCM modes.

```
typedef enum _EC_T_DCM_MODE
{
    eDcmMode_Off                = 0,
    eDcmMode_BusShift           = 1,
    eDcmMode_MasterShift        = 2,
    eDcmMode_LinkLayerRefClock  = 3,
    eDcmMode_MasterRefClock     = 4,
    eDcmMode_Dcx                = 5,
    eDcmMode_MasterShiftByApp   = 6
} EC_T_DCM_MODE;
```

EC_T_DCM_CONFIG_BUSSHIFT

Contains the configuration information for the DCM bus shift mode. Valid if eMode is set to eDcmMode_BusShift

```
typedef struct _EC_T_DCM_CONFIG_BUSSHIFT {  
    EC_T_INT      nCtlSetVal;  
    EC_T_INT      nCtlGain;  
    EC_T_INT      nCtlDriftErrorGain;  
    EC_T_INT      nMaxValidVal;  
    EC_T_BOOL     bLogEnabled;  
    EC_T_DWORD    dwInSyncLimit;  
    EC_T_DWORD    dwInSyncSettleTime;  
    EC_T_BOOL     bCtlOff;  
    EC_T_BOOL     bUseDcLoopCtlStdValues;  
    EC_T_DWORD    dwInSyncStartDelayCycle;  
} EC_T_DCM_CONFIG_BUSSHIFT;
```

Description

nCtlSetVal

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

nCtlGain

[in] Proportional gain in ppt (part per thousand). Default is value 2. A value of 0 let the current setting unmodified.

nCtlDriftErrorGain

[in] Multiplier for drift error. Default value is 3. A value of 0 let the current setting unmodified.

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 let the current setting unmodified.

bLogEnabled

[in] If set to EC_TRUE, logging information are generated and can be get calling **ecatDcmGetLog**

dwInSyncLimit

[in] Limit in nsec for InSync monitoring. Default value is 4000. A value of 0 let the current setting unmodified.

dwInSyncSettleTime

[in] Settle time [ms] for InSync monitoring. Default value is 1500. A value of 0 let the current setting unmodified.

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.

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 a problems with the reference clock to get InSync.

dwInSyncStartDelayCycle

[in] Delay time in msec before InSync monitoring start

EC_T_DCM_CONFIG_MASTERSHIFT

Contains the configuration information for the DCM master shift mode. Valid if eMode is set to eDcmMode_MasterShift

```
typedef struct _EC_T_DCM_CONFIG_MASTERSHIFT
{
    EC_T_INT      nCtlSetVal;
    EC_T_INT      nCtlGain;
    EC_T_INT      nCtlDriftErrorGain;
    EC_T_INT      nMaxValidVal;
    EC_T_BOOL     bLogEnabled;
    EC_T_DWORD    dwInSyncLimit;
    EC_T_DWORD    dwInSyncSettleTime;
    EC_T_BOOL     bCtlOff;
    EC_T_DC_STARTTIME_CB_DESC DcStartTimeCallbackDesc;
    EC_T_DWORD    dwInSyncStartDelayCycle;
} EC_T_DCM_CONFIG_MASTERSHIFT;
```

Description

nCtlSetVal

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

nCtlGain

[in] Proportional gain in ppt (part per thousand). Default is value 2. A value of 0 let the current setting unmodified.

nCtlDriftErrorGain

[in] Multiplier for drift error. Default value is 3. A value of 0 let the current setting unmodified.

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 let the current setting unmodified.

bLogEnabled

[in] If set to EC_TRUE, logging information are generated and can be get calling **ecatDcmGetLog**

dwInSyncLimit

[in] Limit in nsec for InSync monitoring. Default value is 4000. A value of 0 let the current setting unmodified.

dwInSyncSettleTime

[in] Settle time [ms] for InSync monitoring. Default value is 1500. A value of 0 let the current setting unmodified.

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 ecatDcmGetAdjust function.

DcStartTimeCallbackDesc

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

dwInSyncStartDelayCycle

[in] Delay time in msec before InSync monitoring start

EC T DCM CONFIG LINKLAYERREFCLOCK

Contains the configuration information for the DCM LinkLayerRefClock mode. Valid if eMode is set to eDcmMode_LinkLayerRefClock

```
typedef struct _EC_T_DCM_CONFIG_LINKLAYERREFCLOCK
{
    EC_T_INT      nCtlSetVal;
    EC_T_BOOL     bLogEnabled;
    EC_T_DC_STARTTIME_CB_DESC DcStartTimeCallbackDesc;
} EC_T_DCM_CONFIG_LINKLAYERREFCLOCK;
```

Description

nCtlSetVal

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

bLogEnabled

[in] If set to EC_TRUE, logging information are generated and can be get calling **ecatDcmGetLog**

DcStartTimeCallbackDesc

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

EC T DCM CONFIG MASTERREFCLOCK

Contains the configuration information for the DCM MasterRefClock mode. Valid if eMode is set to eDcmMode_MasterRefClock

```
typedef struct _EC_T_DCM_CONFIG_MASTERREFCLOCK {
    EC_T_INT      nCtlSetVal;
    EC_T_BOOL     bLogEnabled;
    EC_T_DWORD    dwInSyncLimit;
    EC_T_DWORD    dwInSyncSettleTime;
    EC_T_DWORD    dwInSyncStartDelayCycle;
} EC_T_DCM_CONFIG_MASTERREFCLOCK;
```

Description

nCtlSetVal

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

bLogEnabled

[in] If set to EC_TRUE, logging information are generated and can be get calling **ecatDcmGetLog**

dwInSyncLimit

[in] Limit in nsec for InSync monitoring. Default value is 4000. A value of 0 let the current setting unmodified.

dwInSyncSettleTime

[in] Settle time [ms] for InSync monitoring. Default value is 1500. A value of 0 let the current setting unmodified.

dwInSyncStartDelayCycle

[in] Delay time in msec before InSync monitoring start

EC T DCM CONFIG DCX

Contains the configuration information for the DCX external synchronization mode. Valid if eMode is set to eDcmMode_Dcx. For further details see Feature Pack External Synchronization documentation.

EC_T_DC_STARTTIME_CB_DESC

EC-Master requests DC start time for every single slave from a given callback DcStartTimeCallbackDesc with slave station address as input parameter. The slave specific DC start time value will be passed directly to the slave without modifications by master. This means no other values like nCtlSetVal will be added. Shift value configured in ENI will still be applied.

```
typedef EC_T_DWORD (*EC_PF_DC_STARTTIME_CB)(EC_T_VOID*      pvContext,
                                              EC_T_WORD      wSlaveFixedAddr,
                                              EC_T_UINT64*     pqwDcStartTime);
```

Parameters

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

wSlaveFixedAddr
[in] Slave fixed address.

pqwDcStartTime
[out] DC start time for specific slave.

Return

EC_E_NOERROR if successful. If unsuccessful the master will use the default DC start time calculation algorithm as fallback.

```
typedef struct _EC_T_DC_STARTTIME_CB_DESC
{
    EC_T_VOID*      pvContext;
    EC_PF_DC_STARTTIME_CB pfnCallback;
} EC_T_DC_STARTTIME_CB_DESC;
```

Parameters

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

pfnCallback
[in] Dc start time callback function pointer. If not null, DC start time calculated by application, otherwise by master

2.1.7.2 ecatDcmGetStatus

Returns the current controller status.

```
EC_T_DWORD ecatDcmGetStatus (
    EC_T_DWORD*      pdwErrorCode,
    EC_T_INT*        pnDiffCur,
    EC_T_INT*        pnDiffAvg,
    EC_T_INT*        pnDiffMax
);
```

Parameters

pdwErrorCode
[out] Pointer to current error code of the DCM controller (see below).

pnDiffCur
[out] Pointer to current difference between set value and actual value of controller in nanosec.

pnDiffAvg
[out] Pointer to average difference between set value and actual value of controller in nanosec.

pnDiffMax
[out] Pointer to maximum difference between set value and actual value of controller in nanosec.

Possible values for the DCM controller error code

EC_E_NOTREADY

DCM control loop is not running.

EC_E_BUSY

DCM control loop is running and try to get InSync according the DCM configuration

DCM_E_DRIFT_TO_HIGH

The ESC can handle drifts of at most 600ppm. Cause to this state could be integration errors, e.g. wrong or changing timer frequency.

DCM_E_MAX_CTL_ERROR_EXCEED

Set if the controller error exceeds the InSyncLimit, see 2.1.7.1 “ecatDcmConfigure”.

EC_E_NOERROR

DCM control loop is InSync.

Return

EC_E_NOERROR or error code

2.1.7.3 ecatDcmResetStatus

Reset the current controller status, average and maximum difference between set value and actual value.

EC_T_DWORD ecatDcmResetStatus (EC_T_VOID);

Return

EC_E_NOERROR or error code

2.1.7.4 ecatDcmGetBusShiftConfigured

Return information if DCM bus shift mode is possible due to configuration (ENI file). This function can be called any time after [ecatConfigureMaster\(\)](#).

**EC_T_DWORD ecatDcmGetBusShiftConfigured(
EC_T_BOOL* pbBusShiftConfigured
);**

Parameters

pbBusShiftConfigured

[out] Set to EC_TRUE if DCM bus shift mode is possible.

Return

EC_E_NOERROR or error code

2.1.7.5 ecatDcmGetLog

Get logging information from the DCM controller.

**EC_T_DWORD ecatDcmGetLog (
EC_T_CHAR** pszLog
);**

Parameters

pszLog

[out] Pointer to a string containing the current logging information

Comment

This function returns non-zero pointer only if *bLogEnabled* was set to EC_TRUE calling **ecatDcmConfigure()**, see 2.1.7.1 “ecatDcmConfigure”. The logging information is typically saved in the dcmlog0.0.csv file. See “Controller log file description” in chapter 2.1.3 for content description of *pszLog*.

Return

EC_E_NOERROR or error code

2.1.7.6 ecatDcmShowStatus

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

EC_T_DWORD ecatDcmShowStatus(EC_T_VOID);

Return

EC_E_NOERROR or error code

2.1.7.7 ecatDcmGetAdjust

Returns the current adjustment value for the timer. bCtlOff must be set to EC_TRUE to enable external adjustment.

EC_T_DWORD ecatDcmGetAdjust (
 EC_T_INT* **pnAdjustPermil**
);

Parameters

pnAdjustPermil

[out] Current adjustment value of the timer.

Possible values for the DCM controller error code

EC_E_INVALIDSTATE

DCM config bCtlOff is set to EC_FALSE

EC_E_NOERROR

Adjustment value is valid and must be applied.

Return

EC_E_NOERROR or error code

2.1.7.8 API returned status values

EC_E_NOERROR	The function was successfully executed.
DCM_E_NOTINITIALIZED	Init function not called or not successful.
DCM_E_MAX_CTL_ERROR_EXCEED	DCM Controller error: Synchronisation out of limit.
DCM_E_NOMEMORY	Out of memory.
DCM_E_INVALID_HWLAYER	Error at HW access, e.g. BSP invalid.
DCM_E_TIMER_MODIFY_ERROR	Error modifying the timer.
DCM_E_TIMER_NOT_RUNNING	Timer is not running.
DCM_E_WRONG_CPU	Function is called on wrong CPU.
DCM_E_INVALID_SYNC_PERIOD	Invalid DC sync period length (invalid clock master?)
DCM_E_INVALID_SETVAL	DCM Controller: Set value invalid, e. g. to small.
DCM_E_DRIFT_TO_HIGH	DCM Controller: Drift between local timer and ref clock to high.
DCM_E_BUS_CYCLE_WRONG	DCM Controller: dwBusCycleTimeUsec doesn't match real cycle.

2.1.7.9 Notifications

At startup the master raises the notifications EC_NOTIFY_DC_SLV_SYNC, EC_NOTIFY_DC_STATUS and EC_NOTIFY_DCM_SYNC at master state transition from INIT to PREOP.

The order is typically as follows (EC_NOTIFY_DCM_SYNC may be before or after reaching PREOP):

```
EC_NOTIFY_STATECHANGED (INIT)
[...]
```

EC_NOTIFY_DC_SLV_SYNC

```
[...]
```

EC_NOTIFY_DC_STATUS

```
[...]
```

[EC_NOTIFY_DCM_SYNC]

```
[...]
```

EC_NOTIFY_STATECHANGED (PREOP)

```
[...]
```

[EC_NOTIFY_DCM_SYNC]

```
[...]
```


EC_NOTIFY_STATECHANGED (SAFEOP)
[...]
EC_NOTIFY_STATECHANGED (OP)

2.1.7.10 Sync signal activation

The sync signals are activated during transition PREOP – SAFEOP according to Init Command (Ado 0x0980, 0x0990, 0x09A0, 0x09A8).

2.1.7.11 ecatNotify – EC_NOTIFY_DCM_SYNC

DCM InSync notification. See chapter 2.1.1 “DCM in sync”.

This notification is enabled by default. See EC_IOCTL_SET_NOTIFICATION_ENABLED in document EC-Master_ClassB for how to control the deactivation.

Parameters

pbyInBuf
[in] pointer to notification descriptor EC_T_DCM_SYNC_NTIFY_DESC
dwInBufSize
[in] sizeof(EC_T_DCM_SYNC_NTIFY_DESC).
pbyOutBuf
[in] NULL (not used).
dwOutBufSize
[in] 0 (not used).
pdwNumOutData
[in] EC_NULL (not used).

Comment

EC_T_DCM_SYNC_NTIFY_DESC

```
typedef struct _EC_T_DCM_SYNC_NTIFY_DESC
{
    EC_T_DWORD      IsInSync;
    EC_T_INT         nCtlErrorNsecCur;
    EC_T_INT         nCtlErrorNsecAvg;
    EC_T_INT         nCtlErrorNsecMax;
} EC_T_DCM_SYNC_NTIFY_DESC;
```

Description

IsInSync
[in] EC_TRUE as long as time of master and reference clock are in sync. False if the InSyncLimit from the bus shift configuration is exceeded.
nCtlErrorNsecCur
[in] Current difference between set value and actual value of controller in nanosec.
nCtlErrorNsecAvg
[in] Average difference between set value and actual value of controller in nanosec.
nCtlErrorNsecMax
[in] Maximum difference between set value and actual value of controller in nanosec.

2.1.8 Example code

A master application which includes DCM API needs to call following steps:

Part 1 Main Thread: Master initialization and controller configuration, main loop, shutdown.

```
#include <AtEthercat.h>

/* initialize the master */
dwRes = ecatInitMaster(&oInitParms);

/* configure the master */
dwRes = ecatConfigureMaster(eCnfType_Filename, (EC_T_PBYTE)szCfgFile, OsStrlen(szCfgFile));
.
/* register client */
dwRes = ecatRegisterClient(ecatNotifyCallback, pNotification, &oRegisterResults);

/* configure DCM bus shift */
OsMemset(&oDcmConfigure, 0, sizeof(EC_T_DCM_CONFIG_BUSSHIFT));
oDcmConfigure.nCtlSetVal = DCM_CONTROLLER_SETVAL_NANOSEC;
oDcmConfigure.bLogEnabled = EC_FALSE;
dwRes = ecatDcmConfigure(&oDcmConfigure, 0);

/* set master and bus state to OP */
dwRes = ecatSetMasterState(dwStartTimeout+dwScanBustimeout, eEcatState_OP);

/* application loop */
while(bRun)
{
    dwRes = ecatDcmGetStatus(&dwStatus, &nDiffCur, &nDiffAvg, &nDiffMax);
}

/* stop master operation */
dwTmpRes = ecatStop(dwStartTimeout);
```

Part 2 - Cyclic (Job) Thread: master jobs and dcm logging.

```
EC_T_VOID tEcJobTask(EC_T_VOID* pvAppThreadParamDesc)
{
    while (!pDemoThreadParam->bJobThreadShutdown)
    {
        dwRes = ecatExecJob(eUsrJob_ProcessAllRxFrames, &bPrevCycProcessed);

        /* get logging information */
        dwRes = ecatDcmGetLog(&pszLog);

        /* send all cyclic frames */
        dwRes = ecatExecJob(eUsrJob_SendAllCycFrames, EC_NULL);

        /* run the master timer handler */
        dwRes = ecatExecJob(eUsrJob_MasterTimer, EC_NULL);

        /* send all queued acyclic EtherCAT frames */
        dwRes = ecatExecJob(eUsrJob_SendAcycFrames, EC_NULL);

        OsSleep(CYCLE_TIME);
    }
}
```

For closer details find a DCM example in project *EcMasterDemoDc* in the folder *Examples*.

2.1.9 DCM on Windows

2.1.9.1 Purpose of the ECAT Driver

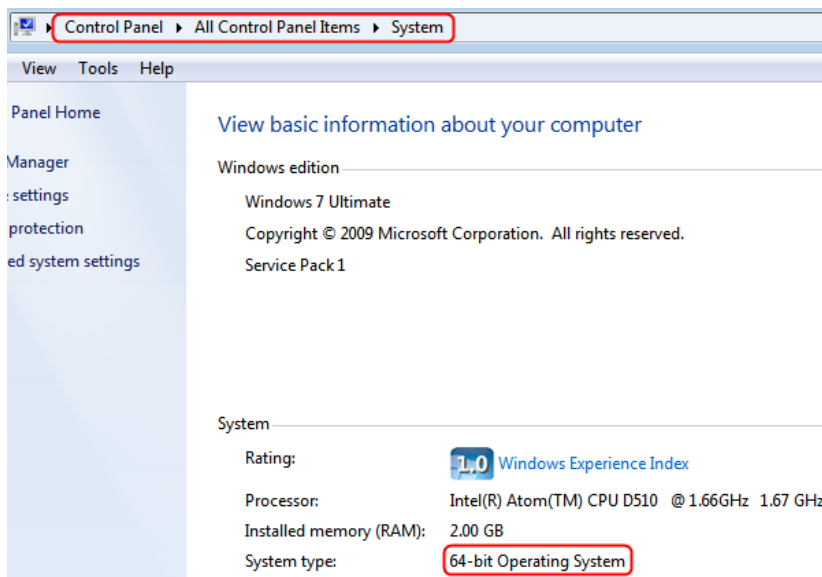
Because of the non-realtime behavior of Windows, it is not possible to get DCM in sync when using the standard timer API.

The ECAT driver implements an auxiliary clock that is accurate enough to get DCM in sync. The ECAT driver **does not** guarantee response to the cycle's deadline.

2.1.9.2 Prerequisites of the ECAT Driver

Microsoft Windows: 32 bit or 64 bit?

On Windows the System Properties dialog contains the information about the PC's architecture. The System type states if it is a 64-bit Operating System or 32-Bit.



Disable CPU Power Management Driver

To prevent influences from the Intel-PPM-driver, the driver can be disabled. The registry file "intelppmOFF.reg" to disable the Intel-PPM-driver can be merged into the registry and is located in the "Files\Windows"-folder of the EC-Master installation directory, e.g. "C:\Program Files\aconis_technologies\EC-Master\Files\Windows". Merging "intelppmON.reg" would enable it again.

Disable Local APIC usage of Windows

To make sure that there is no collision between Windows and the ECAT driver, disable the use of the Local APIC timer for Windows. This can be done by changing the boot configuration with the following command in the Windows console as an administrator:

bcdedit.exe /set {current} useplatformtick yes

No Local APIC support since Windows 10

Since Windows 10, direct access to the local APIC timer is no longer allowed and can cause an BSOD. In this case the ECAT driver can use the Windows Kernel Timer as clock source.

To use the Windows Kernel Timer an additional OS layer parameter

`EC_T_OS_PARMS::PlatformParams.bUseKernelTimerForAuxClk` must be set before `AuxClkInit()`:

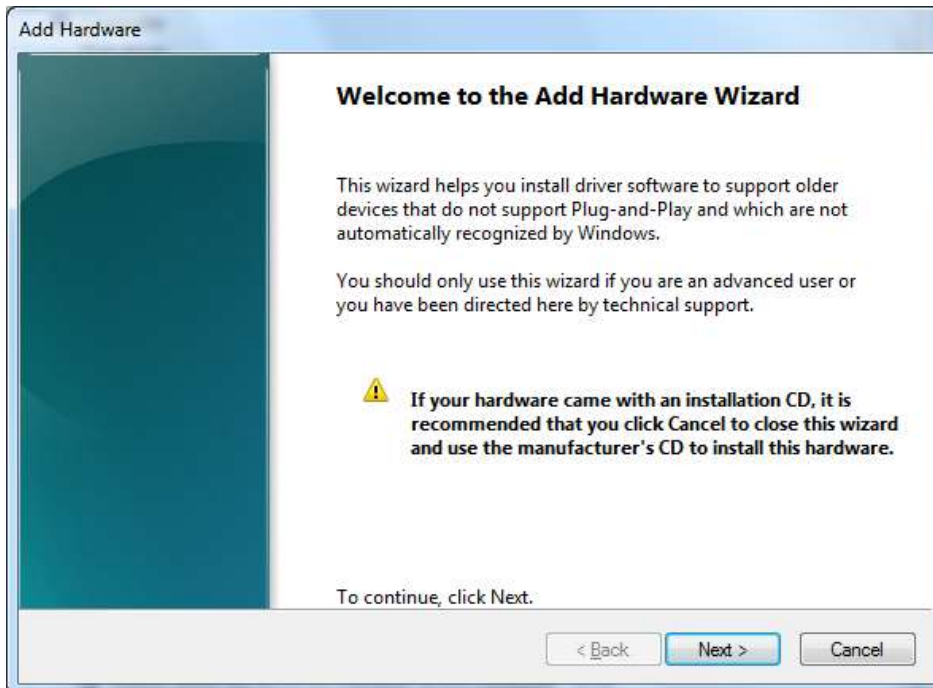
```
EC_T_OS_PARMS oOsParams;  
OsMemset(&oOsParams, 0, sizeof(EC_T_OS_PARMS));  
oOsParams.dwSignature = EC_OS_PARAMS_SIGNATURE;  
oOsParams.dwSize = sizeof(EC_T_OS_PARMS);  
oOsParams.PlatformParams.bUseKernelTimerForAuxClk = EC_TRUE;  
  
OsInit(&oOsParams);
```

2.1.9.3 Installing and configuring the ECAT Driver on Windows

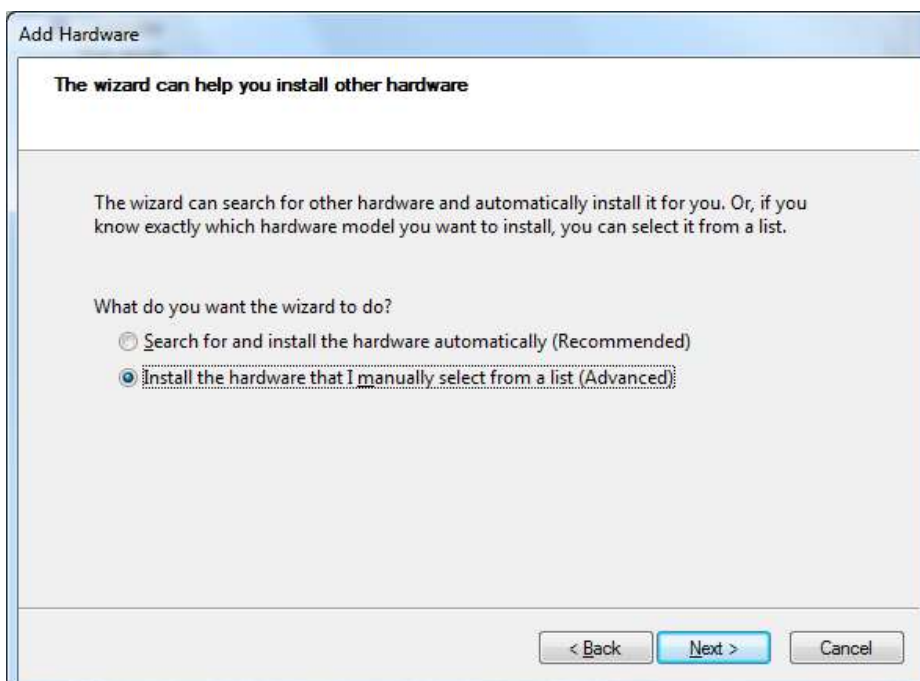
Step 1: Start the “Add Hardware Wizard” and click “Next”



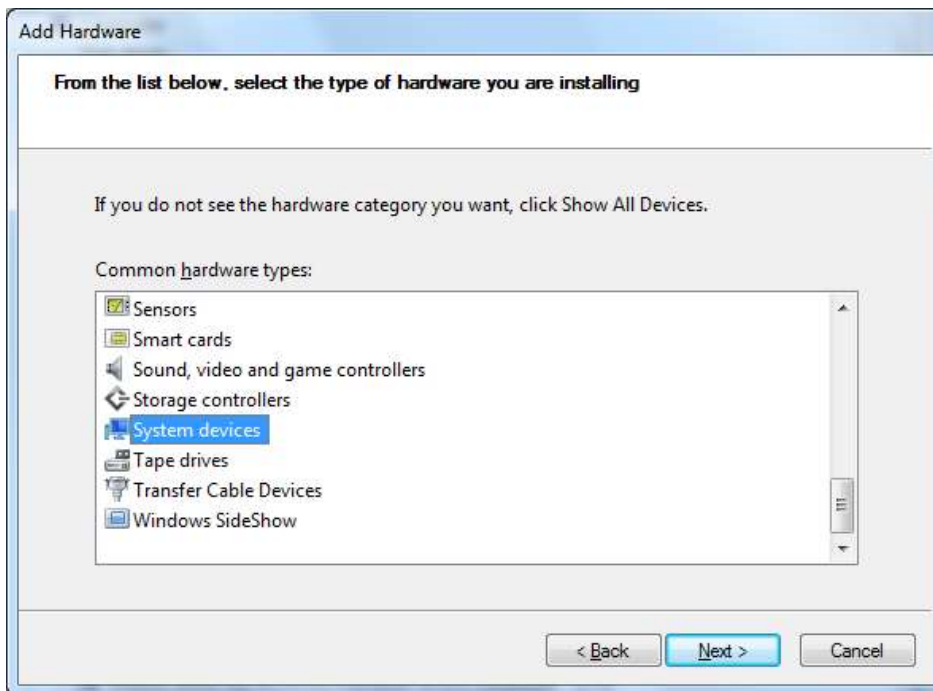
The Add Hardware Wizard shows up:



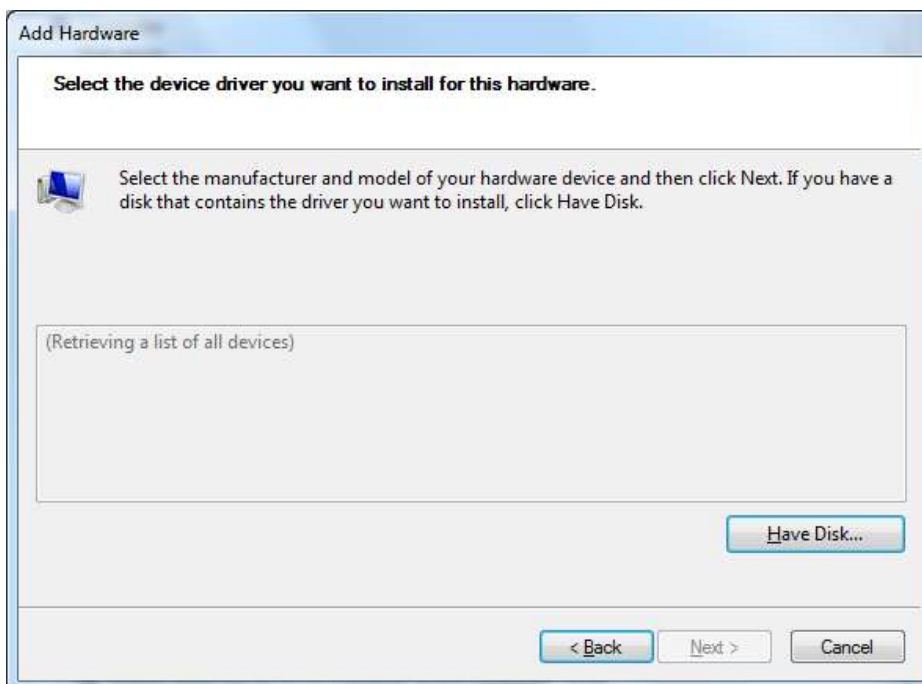
Step 2: Select “Install the hardware...” and proceed



Step 3: Select “System devices” and proceed



Step 4: Click to “Have Disk...” and proceed



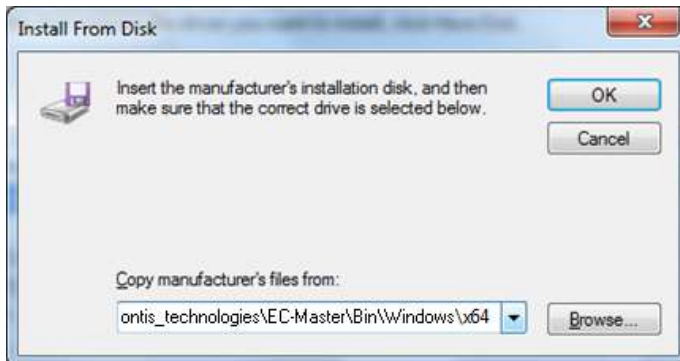
Step 5: Enter the directory to the correct driver version (32 bit or 64 bit)

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



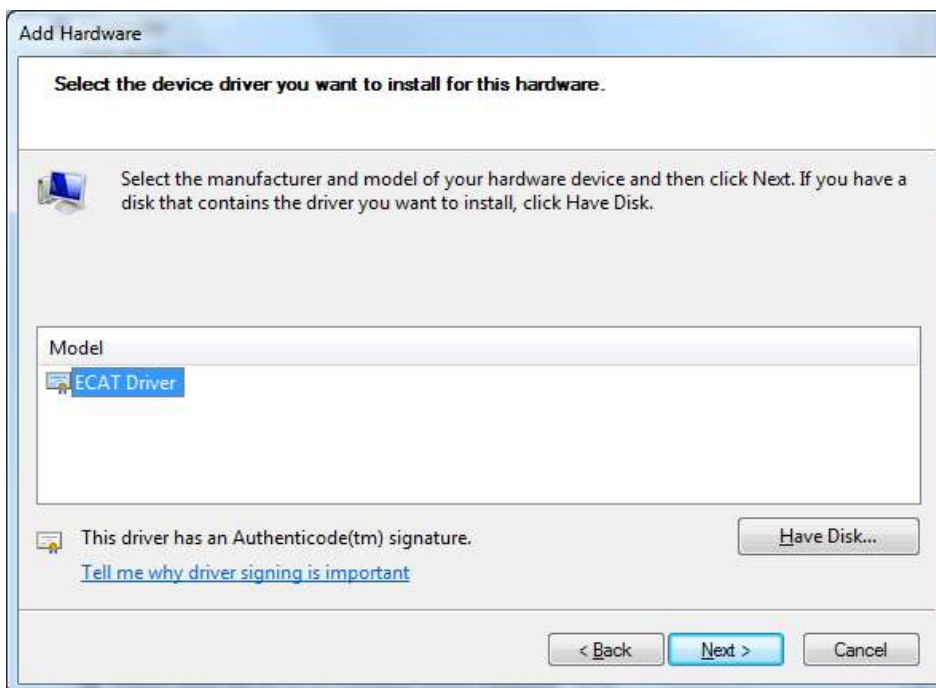
There are two different drivers available: 32 bit and 64 bit.
The subfolder **x86** contains the **32 bit** driver files.
The subfolder **x64** contains the **64 bit** driver files.

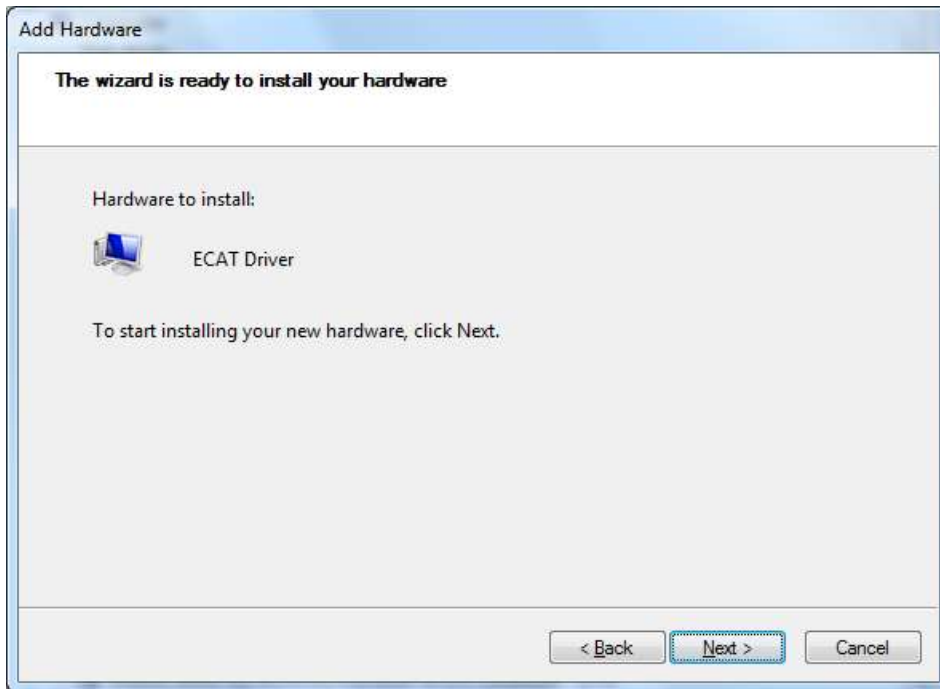
Enter the correct directory at the input box:



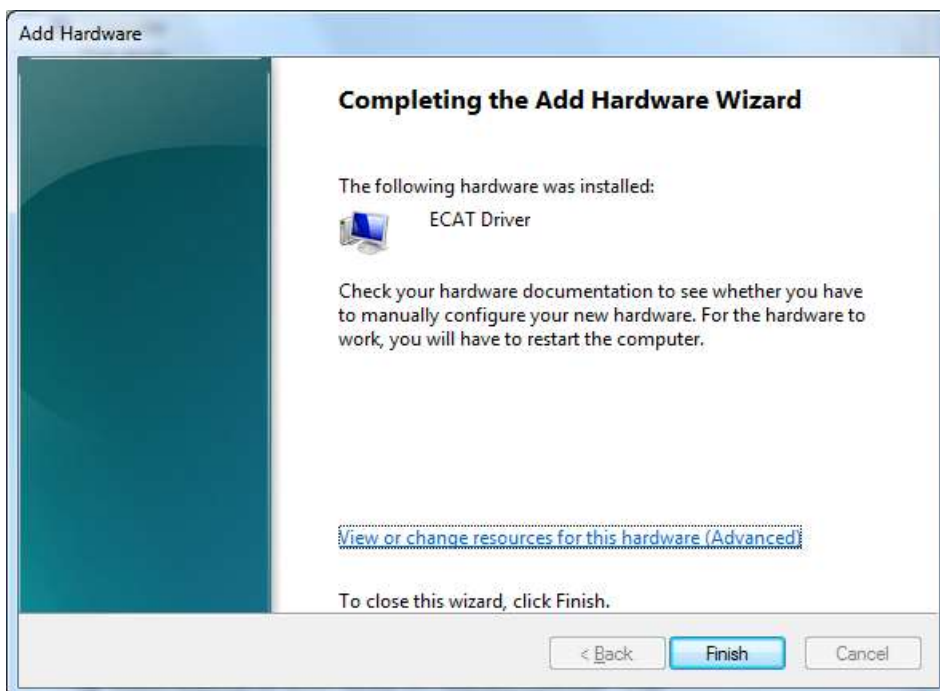
Press OK to proceed.

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



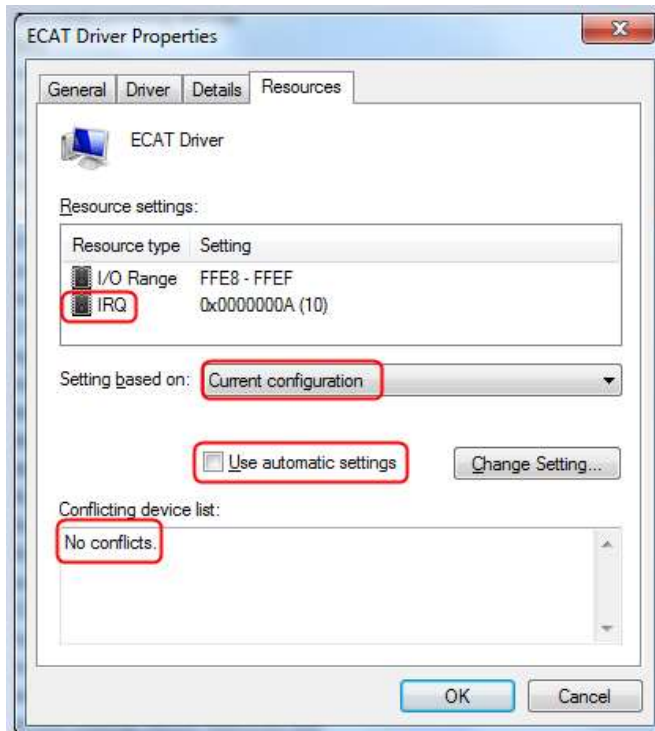


Step 7: Continue on any warning about unsigned driver



Step 8: Configure the driver for local APIC usage

- **!This is not required for Windows 10!**
- Open the Device Manager and open the properties of the ECAT Driver
- Uncheck "Use automatic settings"
- Change "Current Configuration" to "Basic configuration 0001"
- Double-click the "IRQ"-label and adjust the Interrupt Request Value until "No conflicts" is shown.



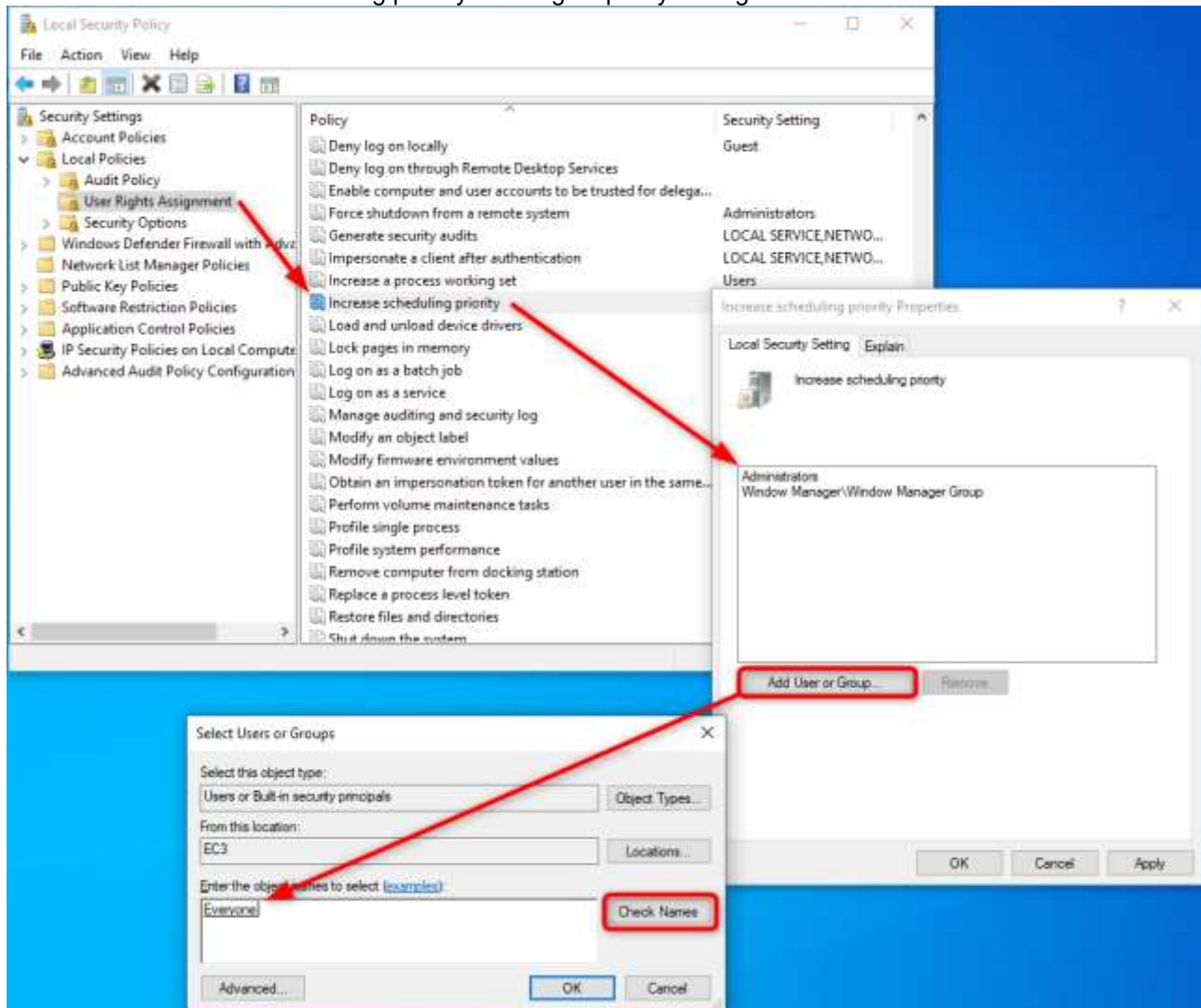
Step 9: Press "OK" to close and apply the settings

- Confirm the manual settings

Step 10: Restart the PC

2.1.9.4 Enable Realtime Priority Class

In order to comply with the timing as best as possible, it is necessary that the application / EcMasterDemoDc runs with realtime priority. By default, an application on Windows can not be run with real-time priority. To allow this the "Increase scheduling priority" user rights policy setting must be set:



3 File access over EtherCAT (FoE)

3.1 Specification

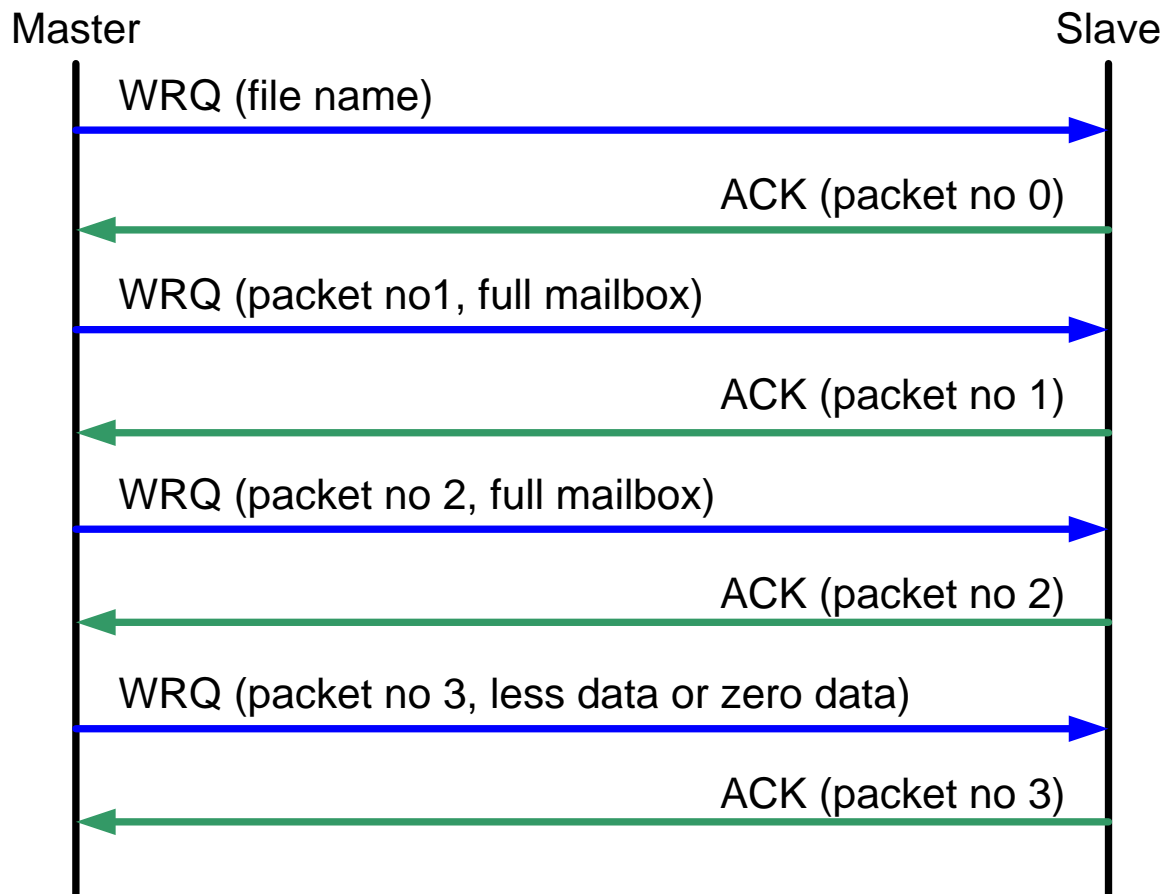
3.1.1 *FoE Protocol*

The File access over EtherCAT (FoE) mailbox command specifies a standard way to download a firmware or any other files from a client to a server or to upload a firmware or any other files from a server to a client.

Reference: ETG.1000.5 and ETG.1000.6

3.1.1.1 FoE file download

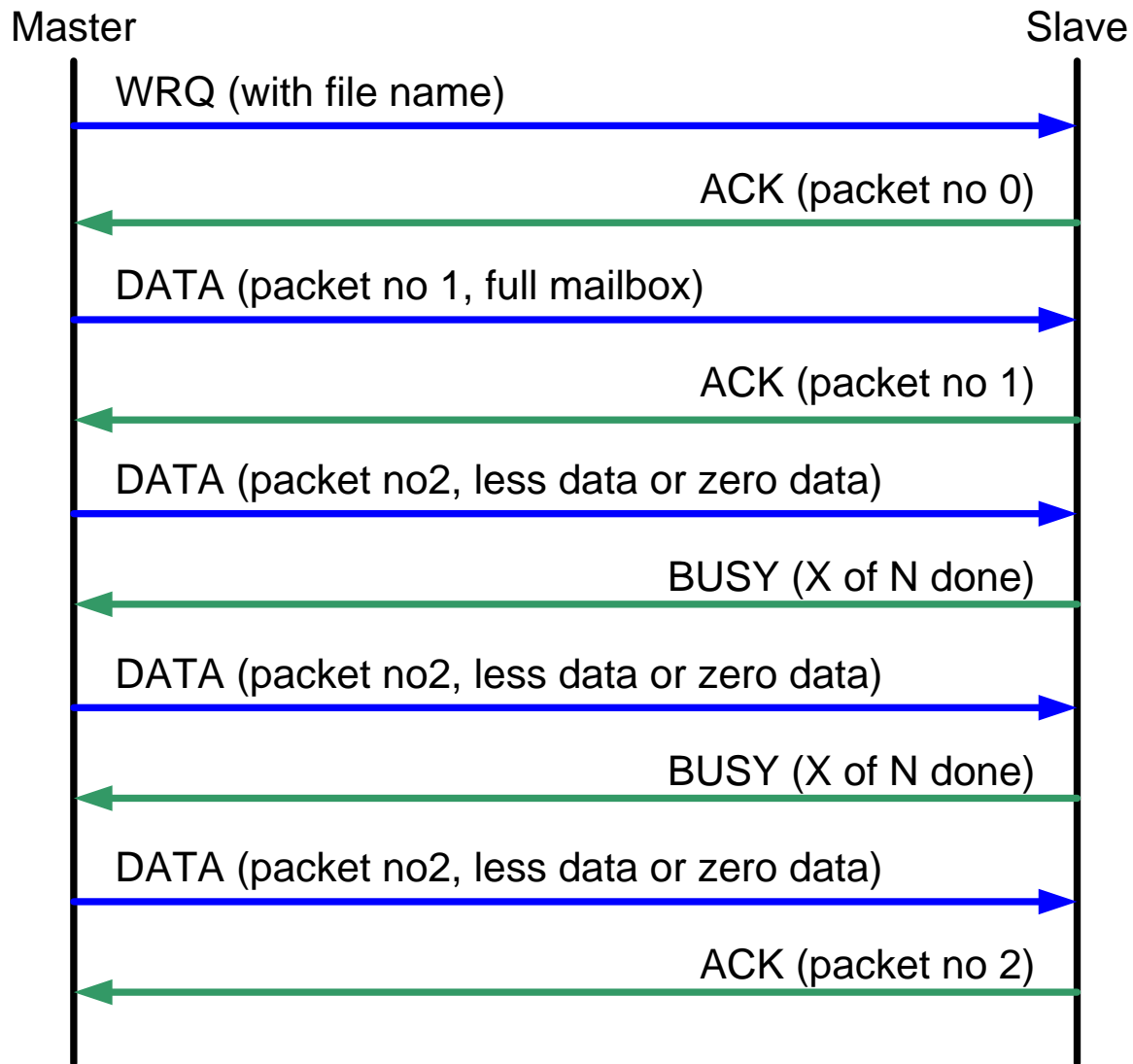
Regular download:



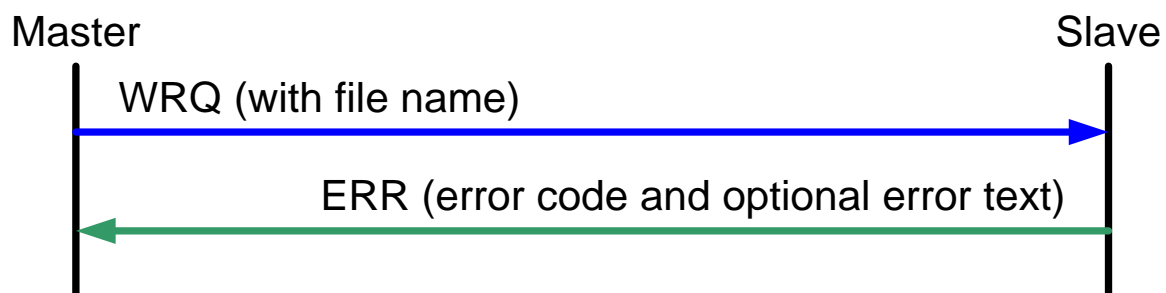
Segmented download:

In case of segmented download the EC-Master raises `EC_NOTIFY_MBOXRCV` to request more data from the application after each ACK from the slave, see below.

Download with busy:

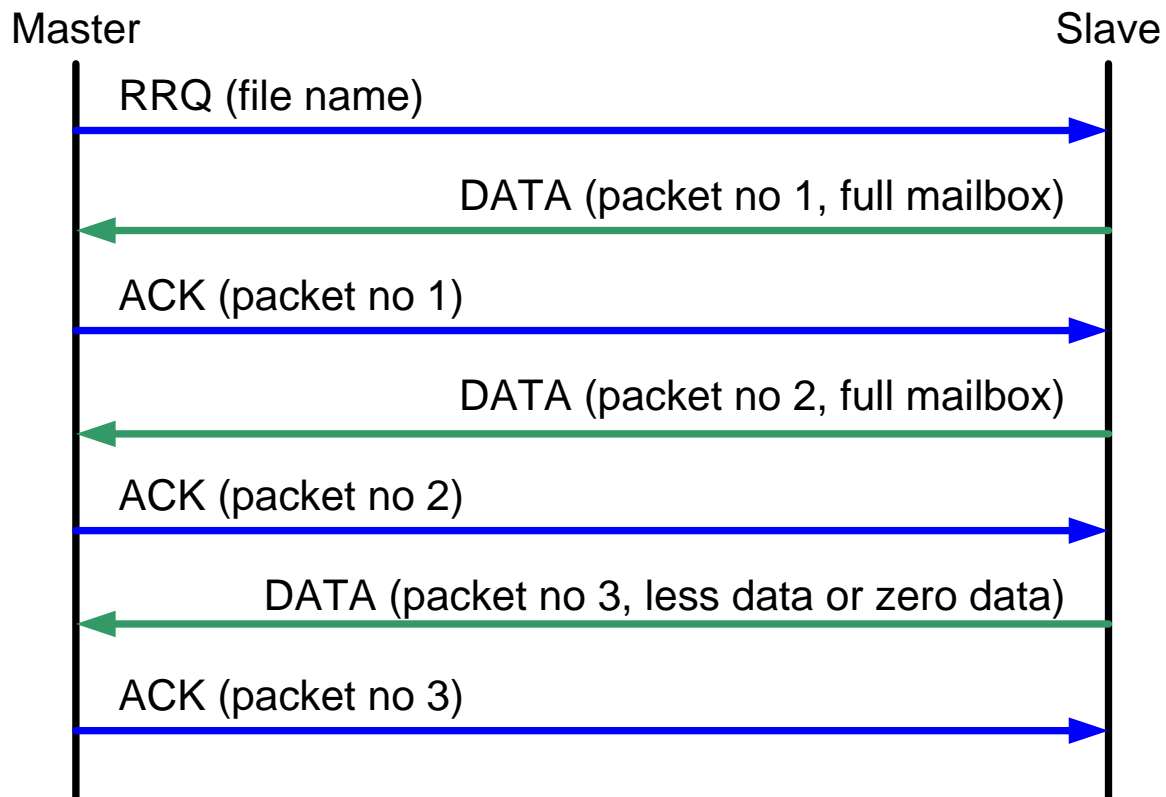


Download with error:



3.1.1.2 FoE file upload

Regular upload:



3.1.2 Boot State

For the download of firmware the BOOT state in the EtherCAT state machine is defined. In bootstrap mode only FoE Download is possible. A special Mailbox size can be supported by the slave for the Boot state (ETG.2000). This is part of the Init-Commands in the network configuration. See also *EC-Master Class B Manual*: `EC_T_CFG_SLAVE_INFO.dwMbxInSize2` and `EC_T_CFG_SLAVE_INFO.dwMbxOutSize2` at `ecatGetCfgSlaveInfo`.

Reference: ETG.1000.5 and ETG.1000.6

3.2 Programmer's Guide

3.2.1 *ecatFoeFileDownload*

Performs an FoE File download to an EtherCAT slave device. This function is used to download a complete file. The function returns after `dwTimeout` ([ms]) are expired or download is completed successfully.

```
EC_T_DWORD ecatFoeFileDownload (  
    EC_T_DWORD    dwSlaveld,  
    EC_T_CHAR*    szFileName,  
    EC_T_DWORD    dwFileNameLen,  
    EC_T_BYTE*    pbyData,  
    EC_T_DWORD    dwDataLen,  
    EC_T_DWORD    dwPassword,  
    EC_T_DWORD    dwTimeout  
);
```

Parameters

dwSlaveld
[in] EtherCAT slave ID. See `ecatGetSlaveld()` in the *EC-Master Class B Manual*.

szFileName
[in] File name of slave file to write.

dwFileNameLen
[in] Length of slave file name in bytes.

pbyData
[in] Data buffer to read downloaded file from.

dwDataLen
[in] Data Buffer length in bytes.

dwPassword
[in] Slave Password.

dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time.

Return

`EC_E_NOERROR` or error code
`EC_E_FOE_ERRCODE_NOTINBOOTSTRAP` if slave in BOOTSTRAP and filename not accepted by slave.
`EC_E_INVALIDPARAM` if maximum file name length of 64 bytes (`MAX_FILE_NAME_SIZE`) exceeded.

Comment

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

Comment

This example code shows how to completely download new firmware into an EtherCAT slave.

```
{
    EC_T_CHAR* szFileName      = "\\fmwr.bin";
    FILE*      pFile           = NULL;
    EC_T_DWORD dwDataLen       = 200*1024;
    EC_T_BYTE* pbyFileData     = EC_NULL;

    dwRes = ecatSetSlaveState(0, DEVICE_STATE_BOOTSTRAP, 5000);
    if (dwRes != EC_E_NOERROR)
    {
        goto Exit;
    }
    pbyFileData = (EC_T_BYTE*)OsMalloc(dwDataLen);
    pFile       = OsFopen(szFileName, "rb");
    dwDataLen   = OsFread(pbyFileData, 1, dwDataLen, pFile);
    OsFclose(pFile);

    LogMsg("FoE download started...");
    dwRes = ecatFoeFileDownload(0, "FMWR", strlen("FMWR"), pbyFileData, dwDataLen, 0, 30000);
    LogMsg("FoE download done (0x%08X)!", dwRes);

    SafeOsFree(pbyFileData);
    if (dwRes != EC_E_NOERROR)
    {
        goto Exit;
    }
}
```

3.2.2 *ecatFoeFileUpload*

Performs an FoE File upload from an EtherCAT slave device. This function is used to upload a complete file. The function returns after `dwTimeout` ([ms]) are expired or upload is completed successfully.

```
EC_T_DWORD ecatFoeFileUpload (  
    EC_T_DWORD    dwSlaveld,  
    EC_T_CHAR*    szFileName,  
    EC_T_DWORD    dwFileNameLen,  
    EC_T_BYTE*    pbyData,  
    EC_T_DWORD    dwDataLen,  
    EC_T_DWORD*   pdwOutDataLen,  
    EC_T_DWORD    dwPassword,  
    EC_T_DWORD    dwTimeout  
);
```

Parameters

dwSlaveld
[in] EtherCAT slave ID. See `ecatGetSlaveld()` in the *EC-Master Class B Manual*.

szFileName
[in] File name of slave file to read.

dwFileNameLen
[in] Length of slave file name in bytes.

pbyData
[out] Data buffer to store uploaded file to.

dwDataLen
[in] Data Buffer length in bytes.

pdwOutDataLen,
[out] Pointer returning size of data uploaded from slave.

dwPassword
[in] Slave Password.

dwTimeout
[in] Timeout in milliseconds. The function will block at most for this time.

Return

EC_E_NOERROR or error code (e.g. *EC_E_FOE_ERRCODE_MAX_FILE_SIZE* in case the file does not fit into the buffer provided (`*pdwOutDataLen > dwDataLen`)).
EC_E_INVALIDPARAM if maximum file name length of 64 bytes (`MAX_FILE_NAME_SIZE`) exceeded.

Comment

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

3.2.3 *ecatFoeDownloadReq*

Initiates an FoE File download to an EtherCAT slave device. This function is used to download a complete file. The function returns immediately. After `dwTimeout` ([ms]) are expired or download is completed successfully `EC_NOTIFY_MBOXRCV` is raised.

```
EC_T_DWORD ecatFoeDownloadReq (  
    EC_T_MBXTFER*  pMbxTfer,  
    EC_T_DWORD     dwSlaveld,  
    EC_T_CHAR*     szFileName,  
    EC_T_DWORD     dwFileNameLen,  
    EC_T_DWORD     dwPassword,  
    EC_T_DWORD     dwTimeout  
);
```

Parameters

pMbxTfer

[in] Pointer to the corresponding mailbox transfer object.
The complete file data to write must to be stored at `pMbxTfer->pbyMbxTferData`.

dwSlaveld

[in] EtherCAT slave ID. See `ecatGetSlaveld()` in the *EC-Master Class B Manual*.

szFileName

[in] File name of slave file to write.

dwFileNameLen

[in] Length of slave file name in bytes.

dwPassword

[in] Slave Password.

dwTimeout

[in] Timeout in milliseconds.

Return

`EC_E_NOERROR` or error code

`EC_E_INVALIDPARAM` if maximum file name length of 64 bytes (`MAX_FILE_NAME_SIZE`) exceeded.

Comment

`EC_NOWAIT` as time-out is still accepted for compatibility (deprecated) and will set the time-out to 10s.

The EC-Master notifies about progress or `MbxTfer` state change with `EC_NOTIFY_MBOXRCV`. See `ecatNotification.cpp` as a sample for how to evaluate the progress.

3.2.4 *ecatFoeSegmentedDownloadReq*

Initiates or continues a segmented FoE File download to an EtherCAT slave device. This function is used to download a file chunk-by-chunk. The function returns immediately. `dwTimeout` ([ms]) specify the overall timeout of the FoE transfer. The EC-Master raises `EC_NOTIFY_MBOXRCV` to request the next chunk from the application and for progress and `MbxTfer` state change information.

```
EC_T_DWORD ecatFoeSegmentedDownloadReq (  
    EC_T_MBXTFER*  pMbxTfer,  
    EC_T_DWORD     dwSlaveld,  
    EC_T_CHAR*     szFileName,  
    EC_T_DWORD     dwFileNameLen,  
    EC_T_DWORD     dwFileSize,  
    EC_T_DWORD     dwPassword,  
    EC_T_DWORD     dwTimeout  
);
```

Parameters

pMbxTfer

[in] Pointer to the corresponding mailbox transfer object.
pMbxTfer->pbyMbxTferData: next chunk, pMbxTfer->dwDataLen: next chunk size.

dwSlaveId

[in] EtherCAT slave ID. See ecatGetSlaveId() in the *EC-Master Class B Manual*.
Only evaluated when initiating the request.

szFileName

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

dwFileNameLen

[in] Length of slave file name in bytes.

dwFileSize

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

dwPassword

[in] Slave Password.
Only evaluated when initiating the request.

dwTimeout

[in] Timeout in milliseconds.
Only evaluated when initiating the request.

Return

EC_E_NOERROR or error code

EC_E_INVALIDPARAM if maximum file name length of 64 bytes (MAX_FILE_NAME_SIZE) exceeded.

Comment

EC_NOWAIT as time-out is still accepted for compatibility (deprecated) and will set the time-out to 10s.

The slave may have a different mailbox size for BOOTSTRAP than for PREOP, SAFEOP, OP.
See e.g. **EC_T_CFG_SLAVE_INFO.dwMbxOutSize2** at **ecatGetCfgSlaveInfo**.

The optimal 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.

The EC-Master notifies about progress or MbxTfer state change with *EC_NOTIFY_MBOXRCV*. See *ecatNotification.cpp* as a sample for how to evaluate the progress.

The EC-Master requests next chunk with *EC_NOTIFY_MBOXRCV*. See *ecatNotification.cpp* as a sample for how to provide data chunks.

3.2.5 *ecatFoeUploadReq*

Initiates an FoE File upload from an EtherCAT slave device. This function is used to upload a complete file. The function returns immediately. After `dwTimeout` ([ms]) are expired or upload is completed successfully `EC_NOTIFY_MBOXRCV` is raised.

```
EC_T_DWORD ecatFoeUploadReq (  
    EC_T_MBXTFER*  pMbxTfer,  
    EC_T_DWORD      dwSlaveld,  
    EC_T_CHAR*      szFileName,  
    EC_T_DWORD      dwFileNameLen,  
    EC_T_DWORD      dwPassword,  
    EC_T_DWORD      dwTimeout  
);
```

Parameters

pMbxTfer
[in] Pointer to the corresponding mailbox transfer object.
The complete file data to read will be stored at `pMbxTfer->pbyMbxTferData`.

dwSlaveld
[in] EtherCAT slave ID. See `ecatGetSlaveld()` in the *EC-Master Class B Manual*.

szFileName
[in] File name of slave file to read.

dwFileNameLen
[in] Length of slave file name in bytes.

dwPassword
[in] Slave Password.

dwTimeout
[in] Timeout in milliseconds.

Return

`EC_E_NOERROR` or error code
`EC_E_INVALIDPARAM` if maximum file name length of 64 bytes (`MAX_FILE_NAME_SIZE`) exceeded.

Comment

`EC_NOWAIT` as time-out is still accepted for compatibility (deprecated) and will set the time-out to 10s.

The EC-Master notifies about progress or `MbxTfer` state change with `EC_NOTIFY_MBOXRCV`. See `ecatNotification.cpp` as a sample for how to evaluate the progress.

3.2.6 *ecatFoeSegmentedUploadReq*

Initiates or continues a segmented FoE File upload from an EtherCAT slave device. This function is used to upload a file chunk-by-chunk. The function returns immediately. `dwTimeout` ([ms]) specify the overall timeout of the FoE transfer. The EC-Master raises `EC_NOTIFY_MBOXRCV` to provide the next chunk to the application and for progress and `MbxTfer` state change information.

```
EC_T_DWORD ecatFoeSegmentedUploadReq (
    EC_T_MBXTFER*  pMbxTfer,
    EC_T_DWORD     dwSlaveld,
    EC_T_CHAR*     szFileName,
    EC_T_DWORD     dwFileNameLen,
    EC_T_DWORD     dwFileSize,
    EC_T_DWORD     dwPassword,
    EC_T_DWORD     dwTimeout
);
```

Parameters

pMbxTfer

[in] Pointer to the corresponding mailbox transfer object.
pMbxTfer->pbyMbxTferData: next chunk, *pMbxTfer->dwDataLen*: next chunk size.

dwSlaveld

[in] EtherCAT slave ID. See `ecatGetSlaveld()` in the *EC-Master Class B Manual*.
 Only evaluated when initiating the request.

szFileName

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

dwFileNameLen

[in] Length of slave file name in bytes.

dwFileSize

[in] Used only for progress information.

dwPassword

[in] Slave Password.
 Only evaluated when initiating the request.

dwTimeout

[in] Timeout in milliseconds.
 Only evaluated when initiating the request.

Return

`EC_E_NOERROR` or error code

`EC_E_INVALIDPARM` if maximum file name length of 64 bytes (`MAX_FILE_NAME_SIZE`) exceeded.

Comment

`EC_NOWAIT` as time-out is still accepted for compatibility (deprecated) and will set the time-out to 10s.

The slave may have a different mailbox size for BOOTSTRAP than for PREOP, SAFEOP, OP.
 See e.g. `EC_T_CFG_SLAVE_INFO.dwMbxInSize2` at `ecatGetCfgSlaveInfo`.

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.

The EC-Master notifies about progress or `MbxTfer` state change with `EC_NOTIFY_MBOXRCV`. See `ecatNotification.cpp` as a sample for how to evaluate the progress.

The EC-Master provides each chunk with `EC_NOTIFY_MBOXRCV`. See `ecatNotification.cpp` as a sample for how to log sizes of uploaded data chunks.

3.2.7 Notifications

Following notifications are defined for FoE.

3.2.7.1 EC_NOTIFY_FOE_MBXSEND_WKC_ERROR

This error will be indicated in case the working counter of a FoE mailbox write command was not set to the expected value of 1.

Detailed error information is stored in structure EC_T_WKCERR_DESC of the union element WkcErrDesc. The structure member SlaveProp contains information about the corresponding slave device.

3.2.7.2 EC_NOTIFY_FOE_MBSLAVE_ERROR

This error will be indicated in case a FoE mailbox slave send an error message.

Detailed error information is stored in structure EC_T_MBOX_FOE_ABORT_DESC of the union element FoeErrorDesc. The structure member SlaveProp contains information about the corresponding slave device, dwErrorCode contains the FoE error code and achErrorString contains the error string.

3.2.7.3 Extending EC_T_MBX_DATA

EC_T_MBX_DATA_FOE

FoE transfer data, e.g. progress information in notification.

```
#define EC_FOE_BUSY_COMMENT_SIZE 32
typedef struct _EC_T_MBX_DATA_FOE {
    EC_T_DWORD    dwTransferredBytes;
    EC_T_DWORD    dwRequestedBytes;
    EC_T_DWORD    dwBusyDone;
    EC_T_DWORD    dwBusyEntire;
    EC_T_CHAR     szBusyComment[EC_FOE_BUSY_COMMENT_SIZE];
    EC_T_DWORD    dwFileSize;
} EC_T_MBX_DATA_FOE;
```

dwTransferredBytes
[out] amount of transferred bytes

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

dwBusyDone
[out] If slave is busy: 0 ... dwBusyEntire

dwBusyEntire
[out] If dwBusyEntire > 0: Slave is busy

szBusyComment
[out] Busy Comment from slave

dwFileSize
[out] File size

4 Vendor specific protocol over EtherCAT (VoE)

4.1 Specification

VoE is for vendor specific protocols. With VoE the vendor has access to a raw EtherCAT mailbox without a specific header or specific protocol mechanism.

4.2 Programmer's Guide

4.2.1 *ecatVoeWrite*

This performs a VoE mailbox write to an EtherCAT slave device. This function blocks until the VoE write was performed successfully or the timeout has elapsed (EC_E_TIMEOUT will be returned in this case). A non-blocking version of this function (*ecatVoeWriteReq*) is also available. Please have a look at section 4.2.2.

```
EC_T_DWORD ecatVoeWrite(  
    EC_T_DWORD dwSlaveId,  
    EC_T_BYTE* pbyData,  
    EC_T_DWORD dwDataLen,  
    EC_T_DWORD dwTimeout  
);
```

Parameters

dwSlaveId

[in] EtherCAT slave ID. See *ecatGetSlaveId()* in the *EC-Master Class B Manual*.

pbyData

[in] Data to be transferred.

dwDataLen

[in] Data size of *pbyData*. The maximum data length including 6 bytes for the mailbox header is given at **EC_T_CFG_SLAVE_INFO.dwMbxOutSize**, see *ecatGetCfgSlaveInfo()* in the *EC-Master Class B Manual*.

dwTimeout

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

Return

EC_E_NOERROR or error code

Comment

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

4.2.2 ecatVoeWriteReq

Initiates a VoE mailbox write to an EtherCAT slave device. If the timeout value was set to EC_NOWAIT this function returns immediately without blocking.

```
EC_T_DWORD ecatVoeWriteReq (  
    EC_T_MBXTFER* pMbxTfer,  
    EC_T_DWORD dwSlaveId,  
    EC_T_DWORD dwTimeout  
);
```

Parameters

pMbxTfer

[in] Pointer to the corresponding mailbox transfer object.
The data to write have to be stored at pMbxTfer->pbyMbxTferData.

dwSlaveId

[in] EtherCAT slave ID. See ecatGetSlaveId() in the *EC-Master Class B Manual*.

dwTimeout

[in] Timeout in milliseconds.

Return

EC_E_NOERROR or error code

Comment

The amount of data bytes to write has to be stored in pMbxTfer->dwDataLen. The maximum data length including 6 bytes for the mailbox header is given at EC_T_CFG_SLAVE_INFO.dwMbxOutSize, see ecatGetCfgSlaveInfo() in the *EC-Master Class B Manual*.

A unique transfer ID must be written into pMbxTfer->dwTferId. See "EtherCAT Mailbox Transfer" in the *EC-Master Class B Manual*.

4.2.3 ecatNotify – eMbxTferType_VOE_WRITE

VoE mailbox was successfully written to the VoE slave.

Parameters

pbyInBuf

[in] pMbxTfer – Pointer to a structure of type EC_T_MBXTFER, this structure contains the corresponding mailbox transfer object.

dwInBufSize

[in] Size of the transfer object.

pbyOutBuf

[in] NULL (not used).

dwOutBufSize

[in] 0 (not used).

pdwNumOutData

[in] EC_NULL (not used).

Comment

The corresponding transfer ID can be found in pMbxTfer->dwTferId. The transfer result is stored in pMbxTfer->dwErrorCode.

4.2.4 ecatVoeRead

Retrieves VoE mailbox, that was sent by an EtherCAT slave device and stored by the EtherCAT master. If a VoE mailbox was already received, further received VoE mailboxes will be discarded as long as ecatVoeRead was not called.

There are two ways to use this function.

1. Synchronously by passing a timeout value that is greater than 0:
The function returns with EC_E_NOERROR if the EtherCAT master has received (or receives) VoE mailbox data from an VoE slave within the indicated timeout. If no data was received, the function returns with EC_E_TIMEOUT.
2. Asynchronously by setting the timeout value to EC_NOWAIT:
If dwTimeout was set to EC_NOWAIT ecatVoeRead returns immediately. If the VoE slave has provided some VoE data the function returns EC_E_NOERROR and EC_E_VOE_NO_MBX_RECEIVED otherwise

```
EC_T_DWORD ecatVoeRead(  
    EC_T_DWORD dwSlaveId,  
    EC_T_BYTE* pbyData,  
    EC_T_DWORD dwDataLen,  
    EC_T_DWORD* pdwOutDataLen,  
    EC_T_DWORD dwTimeout  
);
```

Parameters

dwSlaveId

[in] EtherCAT slave ID. See ecatGetSlaveId() in the *EC-Master Class B Manual*.

pbyData

[out] Data buffer to uploaded data, see below.

dwDataLen

[in] Size of data buffer carried in pbyData, including the Mailbox Header, see below.

pdwOutDataLen

[out] Pointer returning size of data uploaded from slave.

dwTimeout

[in] Timeout in milliseconds. The function will block at most for this time. If the timeout value is set to EC_NOWAIT the function will return immediately.

Return

EC_E_NOERROR or error code

Comment

pbyData includes the Mailbox header of type **ETHERCAT_MBOX_HEADER** followed by the VoE payload.

ETHERCAT_MBOX_HEADER

```
typedef struct TETHERCAT_MBOX_HEADER  
{  
    EC_T_WORD   wLength;  
    EC_T_WORD   wAddress;  
    EC_T_BYTE   byChnPri;  
    EC_T_BYTE   byTypCntRsvd;  
} ETHERCAT_MBOX_HEADER;
```

Description

wLength

Following bytes (payload length)

wAddress

Station address of destination (READ) or source (WRITE). 0 = Master

byChnPri

Channel, Priority

byTypCntRsvd

wMbxType, Counter, Rsvd

4.2.5 ecatNotify – eMbxTferType_VOE_READ

This notification occurs when a EtherCAT slave device has provided new VoE mailbox data to the master, which are ready to retrieve. This data must be retrieved by calling ecatVoeRead.

Parameters

pbyInBuf

[in] pMbxTfer – Pointer to a structure of type EC_T_MBXTFER, this structure contains the used mailbox transfer object . To retrieve this VoE mailbox data ecatVoeRead has to be called.

dwInBufSize

[in] Size of the transfer object.

pbyOutBuf

[in] NULL (not used).

dwOutBufSize

[in] 0 (not used).

pdwNumOutData

[in] EC_NULL (not used).

Comment

The corresponding Slave ID can be found in pMbxTfer->dwTferId.

The MBX data stored in pMbxTfer->pbyMbxTferData may have to be buffered by the client. After ecatNotify returns the pointer and thus the data is invalid. Access to the memory area pointed to by pMbxTfer->pbyMbxTferData after returning from ecatNotify is illegal and the results are undefined.

5 Automation Device Specification over EtherCAT (AoE)

5.1 Specification

The AoE protocol is used to access the Object dictionary of slave devices of underlying fieldbuses, e.g. for a CAN application protocol Slave connected to a EtherCAT-CANopen gateway device. It is also used in relation with the EtherCAT Automation Protocol (EAP).

Reference: ETG.1020 →AoE

5.2 Programmer's Guide

5.2.1 Current limitations

- Fragmented AoE access is not yet implemented
- Only one AoE request can be executed at a time.

5.2.2 *ecatAoeGetSlaveNetId*

Retrieves the NetID of a specific EtherCAT device

```
EC_T_DWORD ecatAoeGetSlaveNetId (  
    EC_T_DWORD dwSlaveId,  
    EC_T_AOE_NETID* poAoeNetId);
```

Parameters

dwSlaveId

[in] EtherCAT slave ID. See *ecatGetSlaveId()* in the *EC-Master Class B Manual*.

poAoeNetId

[out] AoE NetID of the corresponding slave

Return

EC_E_NOERROR on success, error code otherwise.

Comment

The structure of **EC_T_AOE_NETID** is as follows:

```
typedef struct _EC_T_AOE_NETID {  
    EC_T_BYTE aby[6];  
} EC_T_AOE_NETID;
```

5.2.3 ecatAoeRead

Performs a AoE mailbox read request to an EtherCAT slave device.

A non-blocking version of this function (ecatAoeReadReq) is also available. Please have a look at section 5.2.4.

```
EC_T_DWORD ecatAoeRead(  
    EC_T_DWORD dwSlaveld,  
    EC_T_AOE_NETID* poTargetNetId,  
    EC_T_WORD wTargetPort,  
    EC_T_DWORD dwIndexGroup,  
    EC_T_DWORD dwIndexOffset,  
    EC_T_DWORD dwDataLength,  
    EC_T_BYTE* pbyData,  
    EC_T_DWORD* pdwDataOutLen,  
    EC_T_DWORD* pdwErrorCode,  
    EC_T_DWORD* pdwCmdResult,  
    EC_T_DWORD dwTimeout);
```

Parameters

dwSlaveld

[in] EtherCAT slave ID. See ecatGetSlaveld() in the *EC-Master Class B Manual*.

poTargetNetId

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

wTargetPort

[in] Target port.

dwIndexGroup

[in] AoE read command index group.

dwIndexOffset

[in] AoE read command index offset

dwDataLength

[in] Number of bytes to read from the target device. Must correspond with the size of pbyData.

pbyData

[out] Data buffer to store the read data

pdwDataOutLen

[out] Pointer to number of bytes read from the target device.

pdwErrorCode

[out] Pointer to AoE response error code.

pdwCmdResult

[out] Pointer to AoE read command result code.

dwTimeout

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

Return

- *EC_E_NOERROR*
- *EC_E_AOE_VENDOR_SPECIFIC*: will be returned in case the AoE device has reponed with an user defined error code.
- *Other error codes*

Comment

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

5.2.4 ecatAoeReadReq

Performs a non-blocking AoE mailbox read request to an EtherCAT slave device.

```
EC_T_DWORD ecatAoeReadReq(  
    EC_T_MBXTFER* pMbxTfer,  
    EC_T_DWORD dwSlaveId,  
    EC_T_AOE_NETID* poTargetNetId,  
    EC_T_WORD wTargetPort,  
    EC_T_DWORD dwIndexGroup,  
    EC_T_DWORD dwIndexOffset,  
    EC_T_DWORD dwTimeout);
```

Parameters

pMbxTfer

[in] Pointer to the corresponding mailbox transfer object. The data to write have to be stored at *pMbxTfer->pbyMbxTferData*.

dwSlaveId

[in] EtherCAT slave ID. See *ecatGetSlaveId()* in the *EC-Master Class B Manual*.

poTargetNetId

[in] Target NetID. The Target NetID of a AoE slave device can be retrieved by *ecatAoeGetSlaveNetId()*.

wTargetPort

[in] Target port.

dwIndexGroup

[in] AoE read command index group.

dwIndexOffset

[in] AoE read command index offset

dwTimeout

[in] Timeout in milliseconds.

Return

- *EC_E_NOERROR*
- *EC_E_AOE_VENDOR_SPECIFIC*: will be returned in case the AoE device has reponded with an user defined error code.
- Other error codes

Comment

If the functions *ecatAoeReadReq* or *ecatAoeWriteReq* return *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*.

EC_T_MBXTFER::MbxData::AoE_Response is defined as:

```
typedef struct _EC_T_AOE_CMD_RESPONSE{  
    EC_T_DWORD      dwErrorCode; /* AoE response error code */  
    EC_T_DWORD      dwCmdResult; /* AoE command result code */  
    EC_T_DWORD      dwRsvd;      /* Reserved for the future */  
} EC_T_AOE_CMD_RESPONSE;
```

A unique transfer ID must be written into *pMbxTfer->dwTferId*. See “EtherCAT Mailbox Transfer” in the *EC-Master Class B Manual*.

5.2.5 ecatAoeWrite

Performs a AoE mailbox write request to an EtherCAT slave device.

A non-blocking version of this function (ecatAoeWriteReq) is also available. Please have a look at section 5.2.6.

```
EC_T_DWORD ecatAoeWrite(  
    EC_T_DWORD dwSlaveld,  
    EC_T_AOE_NETID* poTargetNetId,  
    EC_T_WORD wTargetPort,  
    EC_T_DWORD dwIndexGroup,  
    EC_T_DWORD dwIndexOffset,  
    EC_T_DWORD dwDataLength,  
    EC_T_BYTE* pbyData,  
    EC_T_DWORD* pdwErrorCode,  
    EC_T_DWORD* pdwCmdResult,  
    EC_T_DWORD dwTimeout);
```

Parameters

dwSlaveld

[in] EtherCAT slave ID. See ecatGetSlaveld() in the *EC-Master Class B Manual*.

poTargetNetId

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

wTargetPort

[in] Target port.

dwIndexGroup

[in] AoE write command index group.

dwIndexOffset

[in] AoE write command index offset

dwDataLength

[in] Number of bytes to write to the target device. Must correspond with the size of pbyData.

pbyData

[in] Data buffer with data that shall be written

pdwErrorCode

[out] Pointer to AoE response error code.

pdwCmdResult

[out] Pointer to AoE write command result code.

dwTimeout

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

Return

- *EC_E_NOERROR*
- *EC_E_AOE_VENDOR_SPECIFIC*: will be returned in case the AoE device has reponded with an user defined error code.
- *Other error codes*

Comment

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

5.2.6 ecatAoeWriteReq

Performs a non-blocking AoE mailbox write request to an EtherCAT slave device.

```
EC_T_DWORD ecatAoeWriteReq(  
    EC_T_MBXTFER* pMbxTfer,  
    EC_T_DWORD dwSlaveId,  
    EC_T_AOE_NETID* poTargetNetId,  
    EC_T_WORD wTargetPort,  
    EC_T_DWORD dwIndexGroup,  
    EC_T_DWORD dwIndexOffset,  
    EC_T_DWORD dwTimeout);
```

Parameters

pMbxTfer

[in] Pointer to the corresponding mailbox transfer object. The data to write have to be stored at *pMbxTfer->pbyMbxTferData*.

dwSlaveId

[in] EtherCAT slave ID. See *ecatGetSlaveId()* in the *EC-Master Class B Manual*.

poTargetNetId

[in] Target NetID. The Target NetID of a AoE slave device can be retrieved by *ecatAoeGetSlaveNetId()*.

wTargetPort

[in] Target port.

dwIndexGroup

[in] AoE write command index group.

dwIndexOffset

[in] AoE write command index offset

dwTimeout

[in] Timeout in milliseconds.

Return

- *EC_E_NOERROR*
- *EC_E_AOE_VENDOR_SPECIFIC*: will be returned in case the AoE device has reponded with an user defined error code.
- Other error codes

Comment

See the "Comment" of the function description in section 5.2.4 *ecatAoeReadReq*.

5.2.7 ecatAoeReadWrite

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

```
EC_T_DWORD ecatAoeReadWrite(  
    EC_T_DWORD dwSlaveld,  
    EC_T_AOE_NETID* poTargetNetId,  
    EC_T_WORD wTargetPort,  
    EC_T_DWORD dwIndexGroup,  
    EC_T_DWORD dwIndexOffset,  
    EC_T_DWORD dwReadDataLength,  
    EC_T_DWORD dwWriteDataLength,  
    EC_T_BYTE* pbyData,  
    EC_T_DWORD* pdwErrorCode,  
    EC_T_DWORD* pdwCmdResult,  
    EC_T_DWORD dwTimeout);
```

Parameters

dwSlaveld

[in] EtherCAT slave ID. See `ecatGetSlaveld()` in the *EC-Master Class B Manual*.

poTargetNetId

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

wTargetPort

[in] Target port.

dwIndexGroup

[in] AoE write command index group.

dwIndexOffset

[in] AoE write command index offset

dwReadDataLength

[in] Number of bytes to read from the device

dwWriteDataLength

[in] Number of bytes to write to the device

pbyData

[in] Data buffer with data used for reading and writing

pdwErrorCode

[out] Pointer to AoE response error code.

pdwCmdResult

[out] Pointer to AoE write command result code.

dwTimeout

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

Return

- `EC_E_NOERROR`
- `EC_E_AOE_VENDOR_SPECIFIC`: will be returned in case the AoE device has reponded with an user defined error code.
- Other error codes

Comment

-

5.2.8 ecatAoeWriteControl

Performs a AoE mailbox write control request to an EtherCAT slave device.

```
EC_T_DWORD ecatAoeWriteControl(  
    EC_T_DWORD dwSlaveld,  
    EC_T_AOE_NETID* poTargetNetId,  
    EC_T_WORD wTargetPort,  
    EC_T_WORD wAoEState,  
    EC_T_WORD wDeviceState,  
    EC_T_DWORD dwDataLen,  
    EC_T_BYTE* pbyData,  
    EC_T_DWORD* pdwErrorCode,  
    EC_T_DWORD* pdwCmdResult,  
    EC_T_DWORD dwTimeout);
```

Parameters

dwSlaveld
[in] EtherCAT slave ID. See ecatGetSlaveld() in the *EC-Master Class B Manual*.

poTargetNetId
[in] Target NetID. The Target NetID of a AoE slave device can be retrieved by ecatAoeGetSlaveNetId().

wTargetPort
[in] Target port

wAoEState
[in] AoE state

wDeviceState
[in] Device specific state

dwDataLen
[in] Number of bytes to write to the target device. Must correspond with the size of pbyData.

pbyData
[in] Data buffer with data that shall be written

pdwErrorCode
[out] Pointer to AoE response error code.

pdwCmdResult
[out] Pointer to AoE write command result code.

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

Return

- *EC_E_NOERROR*
- *EC_E_AOE_VENDOR_SPECIFIC*: will be returned in case the AoE device has reponded with an user defined error code.
- *Other error codes*

Comment

-

5.2.9 *ecatNotify – eMbxTferType_AOE_READ*

AoE read request was completet.

Parameters

pbyInBuf

[] pMbxTfer – Pointer to a structure of type EC_T_MBXTFER, this structure contains the used mailbox transfer object. This mailbox transfer object also contains AoE device error codes in case of an AoE access error. See section 5.2.4 *ecatAoeReadReq()*.

dwInBufSize

[] Size of the transfer object.

pbyOutBuf

[] NULL (not used).

dwOutBufSize

[] 0 (not used).

pdwNumOutData

[] EC_NULL (not used).

Comment

The MBX data stored in pMbxTfer->pbyMbxTferData may have to be buffered by the client. After *ecatNotify* returns the pointer and thus the data is invalid. Access to the memory area pointed to by pMbxTfer->pbyMbxTferData after returning from *ecatNotify* is illegal and the results are undefined.

5.2.10 *ecatNotify – eMbxTferType_AOE_WRITE*

AoE write request was completet.

Parameters

pbyInBuf

[] pMbxTfer – Pointer to a structure of type EC_T_MBXTFER, this structure contains the used mailbox transfer object. This mailbox transfer object also contains AoE device error codes in case of an AoE access error. See section 5.2.4 *ecatAoeReadReq()*.

dwInBufSize

[] Size of the transfer object.

pbyOutBuf

[] NULL (not used).

dwOutBufSize

[] 0 (not used).

pdwNumOutData

[] EC_NULL (not used).

Comment

The MBX data stored in pMbxTfer->pbyMbxTferData may have to be buffered by the client. After *ecatNotify* returns the pointer and thus the data is invalid. Access to the memory area pointed to by pMbxTfer->pbyMbxTferData after returning from *ecatNotify* is illegal and the results are undefined.