



**acontis technologies GmbH**

**SOFTWARE**

**EC-Master**

**EtherCAT® Master Stack Class A**

**V3.3**

**AT3301**

**Edition: February 6, 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

<b>1</b>	<b>Synchronization with Distributed Clocks (DC)</b>	<b>4</b>
1.1	Technical overview . . . . .	4
1.1.1	Support SubDevices and topologies . . . . .	4
1.1.2	Typical Bus Timing . . . . .	5
1.1.3	Implementation Details . . . . .	6
1.1.4	SubDevices in sync . . . . .	6
1.1.5	Sync Window Monitoring . . . . .	6
1.2	Configuration with EC-Engineer . . . . .	6
1.2.1	Distributed Clocks MainDevice settings (Expert) . . . . .	7
1.2.2	Distributed Clocks SubDevice settings (Expert) . . . . .	8
1.3	Programmer's Guide . . . . .	9
1.3.1	emDcConfigure . . . . .	9
1.3.2	emDcIsEnabled . . . . .	11
1.3.3	emGetBusTime . . . . .	11
1.3.4	emDcContDelayCompEnable . . . . .	11
1.3.5	emDcContDelayCompDisable . . . . .	12
1.3.6	emIoCtl - EC_IOCTL_DC_SLV_SYNC_STATUS_GET . . . . .	12
1.3.7	emIoCtl - EC_IOCTL_DC_SETSYNCSTARTOFFSET . . . . .	12
1.3.8	emIoCtl - EC_IOCTL_DC_FIRST_DC_SLV_AS_REF_CLOCK . . . . .	13
1.3.9	emFindInpVarByName - "Inputs.BusTime" . . . . .	13
1.3.10	emIoCtl - EC_IOCTL_DC_ENABLE_ALL_DC_SLV . . . . .	14
1.3.11	emNotify - EC_NOTIFY_REFCLOCK_PRESENCE . . . . .	14
1.3.12	emNotify - EC_NOTIFY_DC_STATUS . . . . .	15
1.3.13	emNotify - EC_NOTIFY_DC_SLV_SYNC . . . . .	15
<b>2</b>	<b>MainDevice Synchronization (DCM)</b>	<b>17</b>
2.1	Technical overview . . . . .	17
2.1.1	DCM Modes . . . . .	18
2.1.2	Sync signal activation . . . . .	18
2.1.3	DCM in sync . . . . .	18
2.1.4	Controller adjustment . . . . .	18
2.1.5	DCM MainDevice Shift mode . . . . .	21
2.1.6	DCM MainDevice Ref Clock mode . . . . .	22
2.1.7	DCM Linklayer Ref Clock mode . . . . .	22
2.2	Configuration with ET9000 . . . . .	22
2.3	Programmer's Guide . . . . .	24
2.3.1	emDcmConfigure . . . . .	24
2.3.2	emDcmGetStatus . . . . .	29
2.3.3	emDcmResetStatus . . . . .	30
2.3.4	emDcmGetBusShiftConfigured . . . . .	31
2.3.5	emDcmGetLog . . . . .	31
2.3.6	emIoCtl - EC_IOCTL_DCM_GET_LOG . . . . .	31
2.3.7	emDcmShowStatus . . . . .	32
2.3.8	emDcmGetAdjust . . . . .	33
2.3.9	DCM specific error codes . . . . .	33
2.3.10	Notifications . . . . .	34
2.3.11	emNotify - EC_NOTIFY_DCM_SYNC . . . . .	34
2.4	Code example . . . . .	35
<b>3</b>	<b>Running EcMasterDemoDc</b>	<b>37</b>
3.1	Command line parameters . . . . .	37

# 1 Synchronization with Distributed Clocks (DC)

DC clock synchronization enables all EtherCAT® devices (MainDevice and SubDevices) to share the same EtherCAT® System Time.

A “DC-SubDevice” is defined as a SubDevice that 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-SubDevices:

- 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 MainDevice must synchronize itself on the reference clock (ETG.1020) -> DCM

Reference:

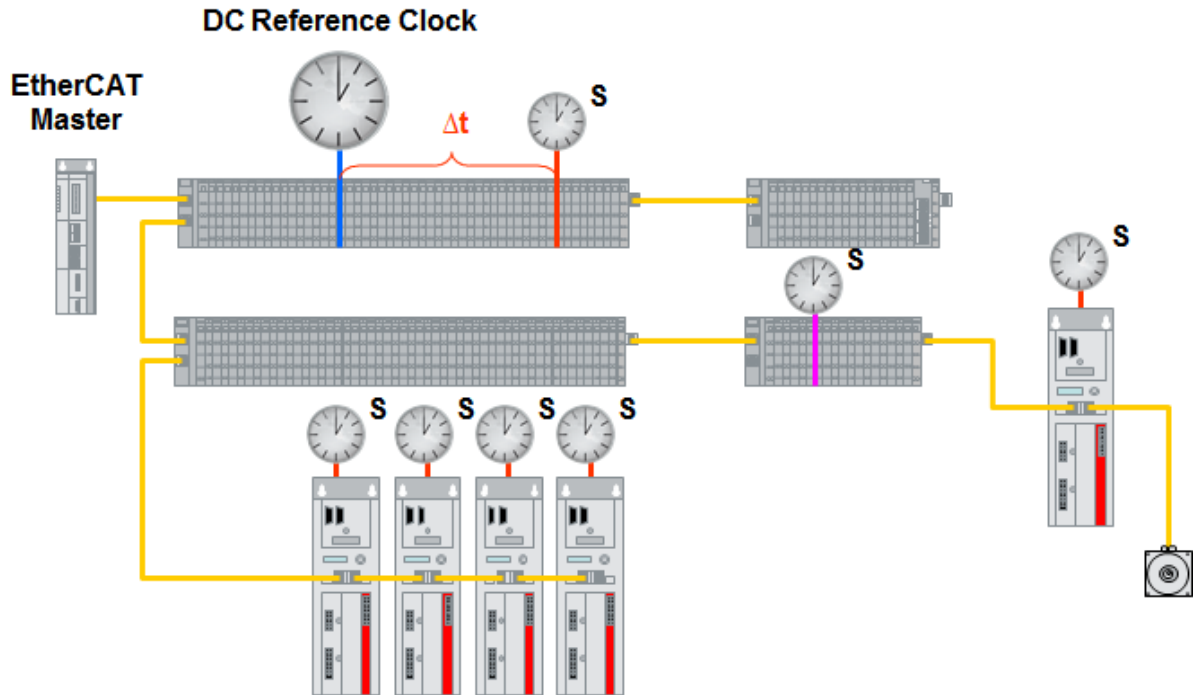
- ETG.1000.3 and ETG.1000.4
- ETG.1020 -> Synchronization
- ETG.8000 -> Distributed Clocks

## 1.1 Technical overview

### 1.1.1 Support SubDevices and topologies

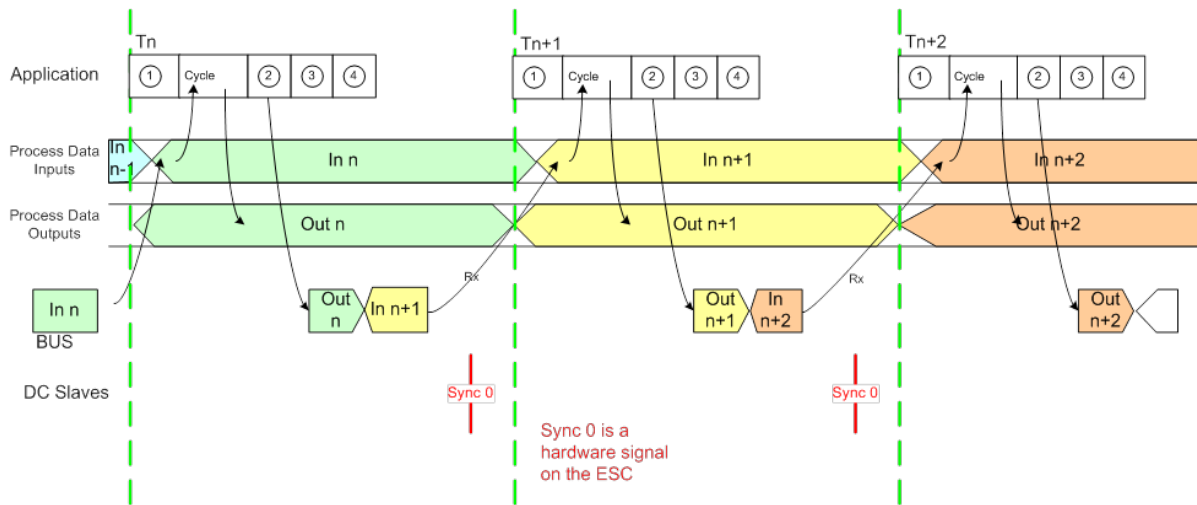
EC-Master supports all currently existing SubDevice types and possible topologies:

- SubDevices 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 SubDevice (auto increment address 0) or in the middle. Only SubDevices 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



### 1.1.2 Typical Bus Timing

The following diagram gives detailed information about the timing of the system:



The timing of the Job Task must be accurate if the system must operate in real-time. It can be disturbed by jitter or drift. Small jitter values are irrelevant, but high jitter values can compromise process data exchange by causing delays that violate strict cycle deadlines. Timing disturbances may have various causes on the MainDevice system like hardware or software delays which are beyond the EC-Master stack's control. However, different DCM modes and parameters are available to compensate the natural drift of MainDevice and SubDevices and to fine-tune synchronization.

### 1.1.3 Implementation Details

The generation of Sync impulses is started after initialization and sending of 10000 FRMW frames which are required to bring SubDevices 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 the start of cyclic operations of SubDevices, which causes the Sync impulse generation to be 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.

### 1.1.4 SubDevices in sync

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

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

Due to technology the SubDevices 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.

### 1.1.5 Sync Window Monitoring

Sync Window Monitoring must be explicitly enabled in the 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 SubDevices!) detected within the settle time. The deviation limit (dwDevLimit) and settle time (dwSettleTime) can be configured using *emDcConfigure()*.

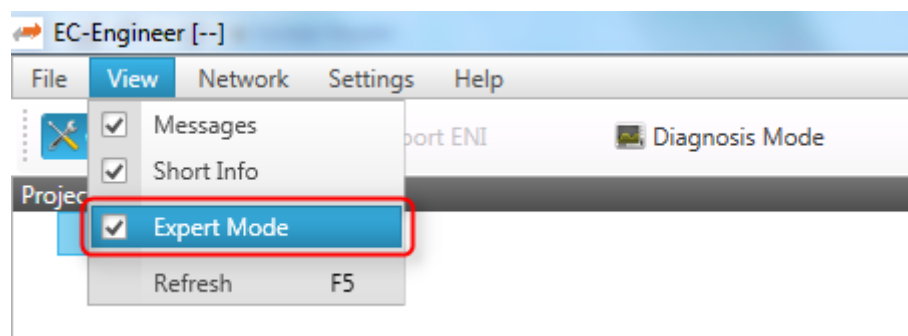
If the configuration only contains the cyclic commands for SAFE-OP or OP (e.g. ET9000) the MainDevice queues acyclic datagrams (Ado 0x092C) for system time difference measurement.

If there are less than two DC SubDevices on the bus (e.g. if the reference clock is the only DC SubDevices 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 SubDevices on bus, SubDevices are immediately considered in sync.

## 1.2 Configuration with EC-Engineer

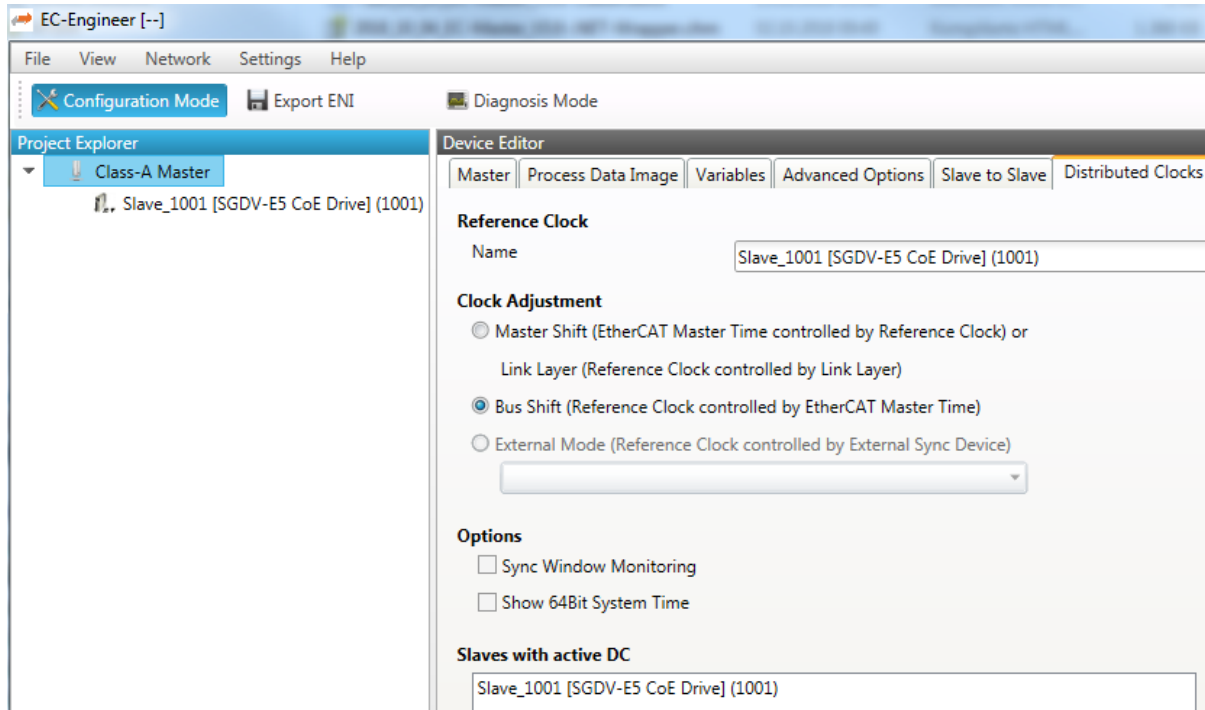
The EC-Engineer automatically chooses the DC settings for SubDevices as proposed by the device's vendor and sets DCM mode to bus shift.

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:



## 1.2.1 Distributed Clocks MainDevice settings (Expert)

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

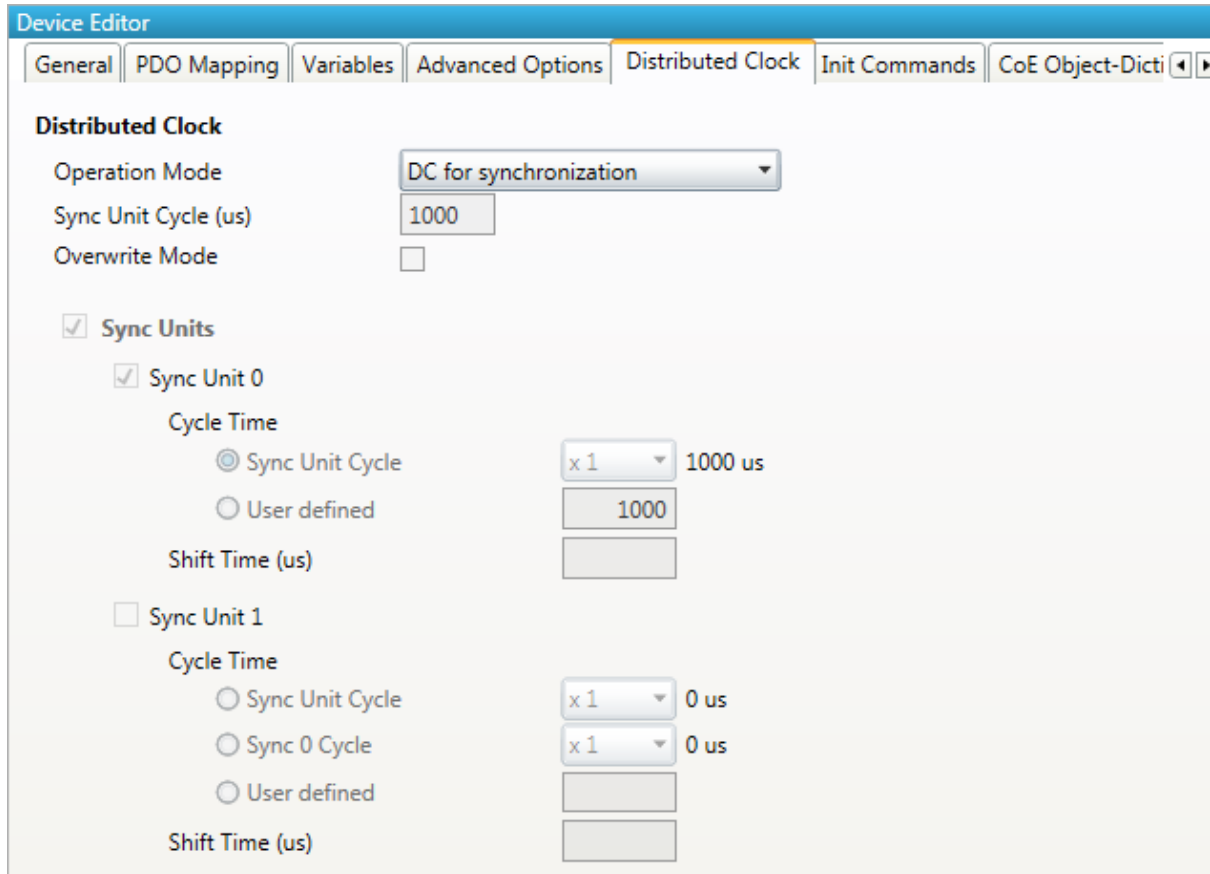


- Reference Clock Name: Name of the reference clock. By default, this is the first SubDevice with DC support.
- MainDevice Shift: The reference clock controls the MainDevice time
- Bus Shift: The MainDevice time controls the reference clock. A command will be inserted in the Cyclic frame to adjust the reference clock system time (write to register 0x0910).
- Continuous Propagation Compensation: A command will be inserted in the Cyclic frame which allows the EtherCAT® MainDevice to measure and compensate the propagation delay time by time.
- Sync Window Monitoring: A command will be inserted in the cyclic frame to read the ESC registers 0x092C. If this is selected the MainDevice will throw a notification.
- Show 64Bit System Time: MainDevice supports SubDevices with 32bit and 64bit system time register (0x0910). If this is selected he will interpret it as 64bit system time.

**Note:** If no reference clock is displayed, please ensure at least one SubDevice in the network is configured for DC operation mode *Distributed Clocks SubDevice settings (Expert)*

## 1.2.2 Distributed Clocks SubDevice settings (Expert)

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



**Device Editor**

General | PDO Mapping | Variables | Advanced Options | **Distributed Clock** | Init Commands | CoE Object-Dicti

**Distributed Clock**

Operation Mode: DC for synchronization

Sync Unit Cycle (us): 1000

Overwrite Mode:

Sync Units

Sync Unit 0

Cycle Time

Sync Unit Cycle: x 1 1000 us

User defined: 1000

Shift Time (us):

Sync Unit 1

Cycle Time

Sync Unit Cycle: x 1 0 us

Sync 0 Cycle: x 1 0 us

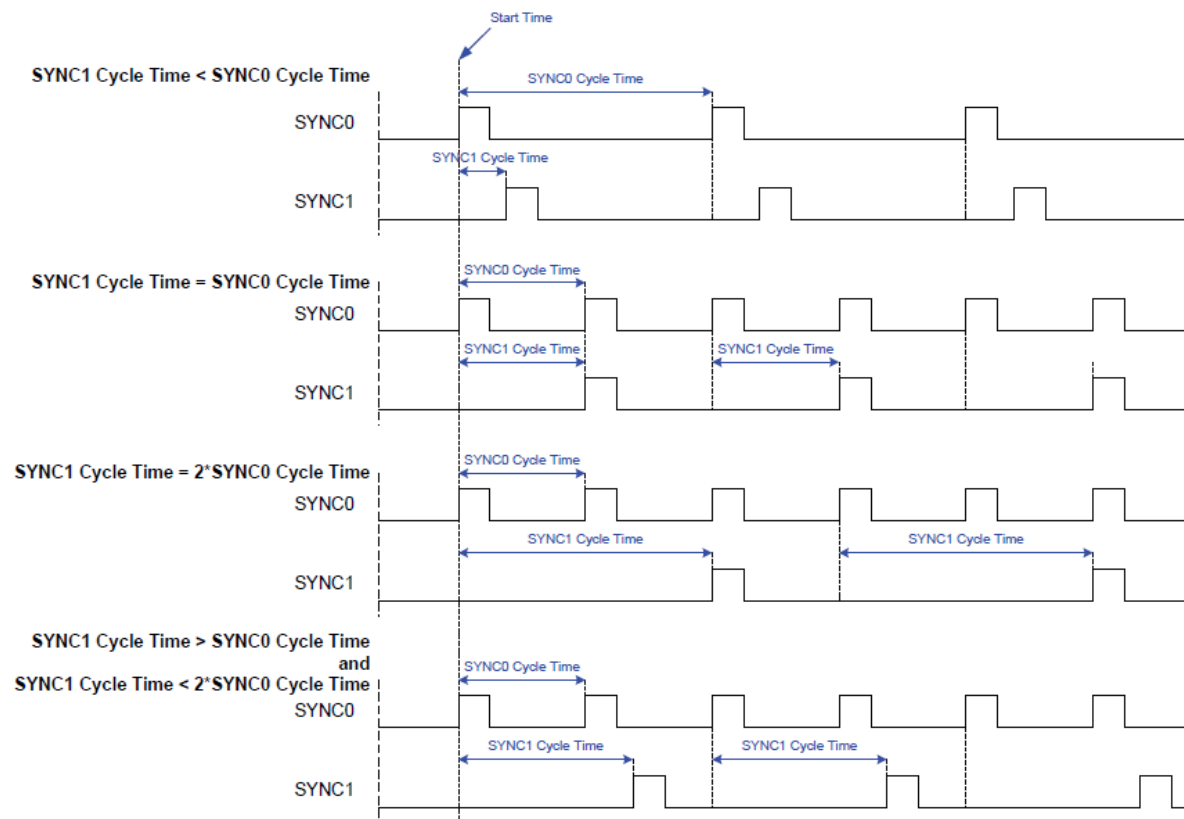
User defined:

Shift Time (us):

- Operation Mode: Selectable DC operation modes. The modes cannot be edited.
- Sync Unit Cycle: Base interval in microseconds which will be used from MainDevice. The Sync Units can be activated and configured to generate signals.

**See also:**

SyncSignal Generation in the ET1100 Datasheet for time describing



## 1.3 Programmer's Guide

### 1.3.1 emDcConfigure

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

```
EC_T_DWORD emDcConfigure (  
    EC_T_DWORD dwInstanceID,  
    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.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pDcConfigure** – [in] Configuration parameter a pointer to a structure of type *EC\_T\_DC\_CONFIGURE*

#### Returns

EC\_E\_NOERROR or error code

struct **EC\_T\_DC\_CONFIGURE**

### Public Members

**EC\_T\_DWORD dwClntId**

[in] Reserved

**EC\_T\_DWORD dwTimeout**

[in] Timeout [ms] for the DC initialization in which time offsets and propagation delays are evaluated ( default: 12 sec )

**EC\_T\_DWORD dwDevLimit**

[in] Maximum permissible deviation of the individual slave clock and the DC reference clock. The maximum deviation is determined by wire or'ed the deviations of the individual slave clocks with one another. The check against the limit is only active if "Sync Window Monitoring" is set in the configuration tool (EC Engineer), which generates a BRD command to read the slave register 0x092C in every cycle. The limit is calculated as follows:

$2^n - 1$  ns, e.g. a dwDevLimit of 4 corresponds to 14 ns.

A value of 0 disables the "Sync Window Monitoring" ( default:  $2^{10} - 1 = 1023$  ns )

**EC\_T\_DWORD dwSettleTime**

[in] Settle time [ms]. At the beginning of the synchronization the slave clocks oscillate strongly. To prevent multiple in-sync and out-of-sync notifications from being generated, a settling time can be set in which no notifications are generated ( default: 1000 msec ).

**EC\_T\_DWORD dwTotalBurstLength**

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

**EC\_T\_DWORD dwBurstBulk**

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

**EC\_T\_BOOL bBulkInLinkLayer**

[in] If EC\_TRUE, bulk is realized by link layer, 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.

**EC\_T\_BOOL bAcycDistributionDisabled**

[in] If EC\_TRUE, acyclic distribution is disabled

**EC\_T\_DWORD 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.

**EC\_T\_BOOL bDcInitBeforeSlaveStateChange**

[in] If EC\_TRUE, DC is initialized before slaves state change to PREOP

**EC\_T\_DWORD dwReserved[4]**

[in/out] Reserved

**See also:**

Chapter “Drift Compensation” of the ETG Document “ESC Datasheet Section 1 - Technology”

### 1.3.2 emDcIsEnabled

static EC\_T\_DWORD **ecatDcIsEnabled** (EC\_T\_BOOL \*pbDcIsEnabled)

EC\_T\_DWORD **emDcIsEnabled** (EC\_T\_DWORD dwInstanceID, EC\_T\_BOOL \*pbDcIsEnabled)  
Determines if DC is enabled and used.

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pbDcIsEnabled** – [out] EC\_TRUE if DC is enabled

**Returns**

EC\_E\_NOERROR or error code

### 1.3.3 emGetBusTime

static EC\_T\_DWORD **ecatGetBusTime** (EC\_T\_UINT64 \*pqwBusTime)

EC\_T\_DWORD **emGetBusTime** (EC\_T\_DWORD dwInstanceID, EC\_T\_UINT64 \*pqwBusTime)  
This function returns the actual bus time in nanoseconds.

**Parameters**

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pqwBusTime** – [out] Bus time [ns]

**Returns**

EC\_E\_NOERROR or error code

### 1.3.4 emDcContDelayCompEnable

static EC\_T\_DWORD **ecatDcContDelayCompEnable** (EC\_T\_VOID)

EC\_T\_DWORD **emDcContDelayCompEnable** (EC\_T\_DWORD dwInstanceID)  
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.

**Parameters**

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

**Returns**

EC\_E\_NOERROR or error code

### 1.3.5 emDcContDelayCompDisable

static EC\_T\_DWORD **ecatDcContDelayCompDisable** (EC\_T\_VOID)

EC\_T\_DWORD **emDcContDelayCompDisable** (EC\_T\_DWORD dwInstanceID)

Disable the continuous propagation delay compensation.

#### Parameters

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

#### Returns

EC\_E\_NOERROR or error code

### 1.3.6 emIoctl - EC\_IOCTL\_DC\_SLV\_SYNC\_STATUS\_GET

**EC\_IOCTL\_DC\_SLV\_SYNC\_STATUS\_GET**

Get the last generated emNotify - EC\_NOTIFY\_DC\_SLV\_SYNC notification.

#### Parameters

- **pbyInBuf** – [in] Should be set to EC\_NULL
- **dwInBufSize** – [in] Should be set to 0
- **pbyOutBuf** – [out] Pointer to *EC\_T\_DC\_SYNC\_NTIFY\_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.

#### Returns

EC\_E\_NOERROR or error code

#### See also:

- *emNotify - EC\_NOTIFY\_DC\_SLV\_SYNC* describes *EC\_T\_DC\_SYNC\_NTIFY\_DESC*
- `emIoctl()` in the *EC-Master Class B* documentation

### 1.3.7 emIoctl - EC\_IOCTL\_DC\_SETSYNCSTARTOFFSET

**EC\_IOCTL\_DC\_SETSYNCSTARTOFFSET**

Set the safety offset applied to the “set DC start time” `InitCmd` during the PS transition.

#### Parameters

- **pbyInBuf** – [in] Pointer to *EC\_T\_DC\_STARTCYCSAFETY\_DESC* data type
- **dwInBufSize** – [in] Size of the input buffer provided at pbyInBuf in bytes
- **pbyOutBuf** – [out] Should be set to EC\_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC\_NULL

#### Returns

EC\_E\_NOERROR or error code

struct **EC\_T\_DC\_STARTCYCSAFETY\_DESC**

## Public Members

EC\_T\_DWORD **dwStartCycSafetyLo**  
[in] Start SYNC Cyc Safety [ns] Lower 32 Bit

EC\_T\_DWORD **dwStartCycSafetyHi**  
[in] Start SYNC Cyc Safety [ns] Upper 32 Bit

Default: 50000000ns

### See also:

`emIoctl ()` in the [EC-Master Class B](#) documentation

## 1.3.8 emIoctl - EC\_IOCTL\_DC\_FIRST\_DC\_SLV\_AS\_REF\_CLOCK

### 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** – [out] Should be set to EC\_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC\_NULL

#### Returns

EC\_E\_NOERROR or error code

## 1.3.9 emFindInpVarByName - “Inputs.BusTime”

The DC system time (written to ESC register 0x0910) is part of the process data with name “Inputs.BusTime”.

```
static EC_T_DWORD ecatFindInpVarByName (
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO *pProcessVarInfoEntry
)
```

```
EC_T_DWORD emFindInpVarByName (
    EC_T_DWORD dwInstanceID,
    const EC_T_CHAR *szVariableName,
    EC_T_PROCESS_VAR_INFO *pProcessVarInfoEntry
)
```

### See also:

- `emFindInpVarByName ()` in the [EC-Master Class B](#) documentation
- `emIoctl ()` in the [EC-Master Class B](#) documentation

### 1.3.10 emIoctl - EC\_IOCTL\_DC\_ENABLE\_ALL\_DC\_SLV

#### 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** – [out] Should be set to EC\_NULL
- **dwOutBufSize** – [in] Should be set to 0
- **pdwNumOutData** – [out] Should be set to EC\_NULL

#### Returns

EC\_E\_NOERROR or error code

#### See also:

emIoctl () in the [EC-Master Class B](#) documentation

### 1.3.11 emNotify - EC\_NOTIFY\_REFCLOCK\_PRESENCE

Distributed clocks reference clock presence notification. It will be received before *emNotify - EC\_NOTIFY\_DC\_SLV\_SYNC* as soon as reference clock was found on bus or removed from bus.

This notification is disabled by default.

#### emNotify - EC\_NOTIFY\_REFCLOCK\_PRESENCE

#### Parameter

- **pbyInBuf**: [in] Pointer to notification descriptor EC\_T\_REFCLOCK\_PRESENCE\_NOTIFY\_DESC
- **dwInBufSize**: [in] sizeof(EC\_T\_REFCLOCK\_PRESENCE\_NOTIFY\_DESC)
- **pbyOutBuf**: [out] Should be set to EC\_NULL
- **dwOutBufSize**: [in] Should be set to 0
- **pdwNumOutData**: [out] Should be set to EC\_NULL

```
struct EC_T_REFCLOCK_PRESENCE_NOTIFY_DESC
```

#### Public Members

EC\_T\_BOOL **bPresent**  
[in] Reference clock present

EC\_T\_SLAVE\_PROP **SlaveProp**  
[in] Slave properties

#### See also:

- `emIoctl - EC_IOCTL_SET_NOTIFICATION_ENABLED` in the [EC-Master Class B](#) documentation for how to control the activation
- `emIoctl ()` in the [EC-Master Class B](#) documentation

### 1.3.12 `emNotify - EC_NOTIFY_DC_STATUS`

Distributed clocks status notification. It will be received after `emNotify - 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 `emNotify - EC_NOTIFY_DC_SLV_SYNC` if SubDevices did not get out of sync.

If `EC_E_NOERROR` is returned and window monitoring is enabled, all SubDevices are in SYNC.

#### `emNotify - EC_NOTIFY_DC_STATUS`

##### Parameter

- `pbyInBuf`: [in] Pointer to `EC_T_DWORD` (`EC_E_NOERROR` on success, Error code otherwise)
- `dwInBufSize`: [in] `sizeof(EC_T_DWORD)`
- `pbyOutBuf`: [out] Should be set to `EC_NULL`
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to `EC_NULL`

##### See also:

- `emIoctl - EC_IOCTL_SET_NOTIFICATION_ENABLED` in the [EC-Master Class B](#) documentation for how to control the deactivation
- `emIoctl ()` in the [EC-Master Class B](#) documentation

### 1.3.13 `emNotify - EC_NOTIFY_DC_SLV_SYNC`

DC SubDevice synchronization notification. Every time the SubDevices are coming in sync or getting out of sync the clients will be notified here. The notification is raised in any case if any DC SubDevices are configured. SubDevices can only be out of sync if *Sync Window Monitoring* is enabled otherwise they are considered in sync.

This notification is enabled by default.

#### `emNotify - EC_NOTIFY_DC_SLV_SYNC`

##### Parameter

- `pbyInBuf`: [in] Pointer to notification descriptor `EC_T_DC_SYNC_NTIFY_DESC`
- `dwInBufSize`: [in] `sizeof(EC_T_DC_SYNC_NTIFY_DESC)`
- `pbyOutBuf`: [out] Should be set to `EC_NULL`
- `dwOutBufSize`: [in] Should be set to 0
- `pdwNumOutData`: [out] Should be set to `EC_NULL`

struct `EC_T_DC_SYNC_NTIFY_DESC`

## Public Members

### EC\_T\_DWORD **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 *emDcConfigure()*.

### EC\_T\_DWORD **IsNegative**

[in] EC\_TRUE : Deviation value is negative, EC\_FALSE: Deviation value is positive

### EC\_T\_DWORD **dwDeviation**

[in] Wire or'ed deviation value [ns] in case of in sync

### EC\_T\_SLAVE\_PROP **SlaveProp**

[in] Slave properties in case of out of sync

### See also:

- *emDcConfigure()*
- `emIoCtl - EC_IOCTL_SET_NOTIFICATION_ENABLED` in the [EC-Master Class B](#) documentation for how to control the deactivation
- `emIoCtl()` in the [EC-Master Class B](#) documentation

## 2 MainDevice Synchronization (DCM)

Applications like motion control need process update every cycle in real time and it is unacceptable to miss data of a cycle or that the SubDevices handle the same process data values twice within a cycle.

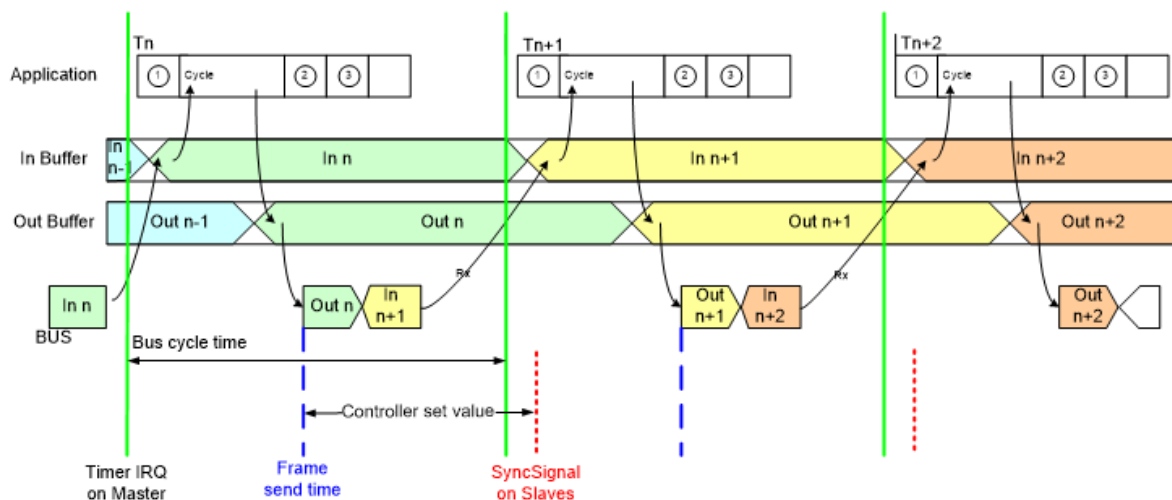
The Distributed Clocks MasterShift Synchronization (DCM) provides controller mechanisms to synchronize the timing at the MainDevice to the SYNC pulses triggering the SubDevice firmware. With DCM, the process data update at the MainDevice and the SubDevices is in correlated timely behavior.

Features:

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

### 2.1 Technical overview

The ENI file contains predefined DCM configuration settings. For example, the EC-Master selects the DCM Mode Bus Shift or DCM Mode MainDevice Shift based on the values specified in the ENI file. Therefore, the application does not need to activate DCM explicitly via API. However, some settings can still be modified programmatically after loading the ENI file. For instance, the parameter “Controller set value” sets the position of the SYNC signals at the SubDevices in relation to the cyclic frame:



The parameter “Controller set value” (`nCtlSetVal`) cannot be derived from the ENI file. It is one of the parameters that can be set using `emDcmConfigure()`.

The actual position of the SYNC signals at the SubDevices, relative to the cyclic frame, typically varies from cycle to cycle due to jitter in the application’s JobTask.

The example program `EcMasterDemoDc` demonstrates, in contrast to `EcMasterDemo`, how to programmatically configure DCM parameters and how the application can generate the DCM log.

The EtherCAT® technology provides sufficient diagnostic capabilities through WKC monitoring to detect communication issues. However, for in-depth analysis, more detailed information, such as the DCM log, is required. The DCM log can be used to evaluate drift control quality and measure jitter values. To enable this, the application’s JobTask must cyclically retrieve the DCM status from the MainDevice. This data should be written from a low-priority thread to avoid impacting real-time performance during diagnostics.

### 2.1.1 DCM Modes

The following DCM Modes are available:

Name	Purpose
BusShift	Synchronize SubDevices to MainDevice timer (Default)
MasterShift	Synchronize MainDevice timer to SubDevice. Feasibility depending on target HW and SW.
LinkLayer-RefClock	Bus Shift using Link Layer clock. Special HW needed.
MasterRef-Clock	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.

The Distributed Clocks MainDevice Synchronization (DCM) in Bus Shift mode adjusts the bus time register of the DC reference clock. All DC SubDevices converge to this time. This mode is useful to synchronize multiple EtherCAT® networks or if it is not possible to adjust the timing of the MainDevice.

**See also:**

*emDcmConfigure()*

### 2.1.2 Sync signal activation

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

### 2.1.3 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 MainDevice awaits that DCM is in sync in MainDevice state transition PREOP->SAFEOP. Therefore the MainDevice state transition may timeout if DCM does not get in sync. Due to the MainDevice'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 SubDevice connected.

### 2.1.4 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 *emDcmConfigure()*.

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_NTIFY_DESC.nCtlErrorNsecCur)
CtlErrFilt	Filtered controller error
Drift [ppm]	Drift between local clock and DC reference clock
CtlErr [1/10 pmil]	For internal use only
CtlOutSum	For internal use only
CtlOutTot	For internal use only
DCStartTime	DC start time send to the SubDevices to activate their SYNC signals
DCMErrorCode	Current error code of the DCM controller (same value as returned by emDcmGet-Status)
DCMInSync	Current InSync state of the DCM controller
DCInSync	Current InSync state of the DC SubDevices
SystemTimeDifference [ns]	Current system time difference of the DC SubDevice if monitoring is enabled

Log file analyze: To understand how the controller values correlates the following table can help:

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

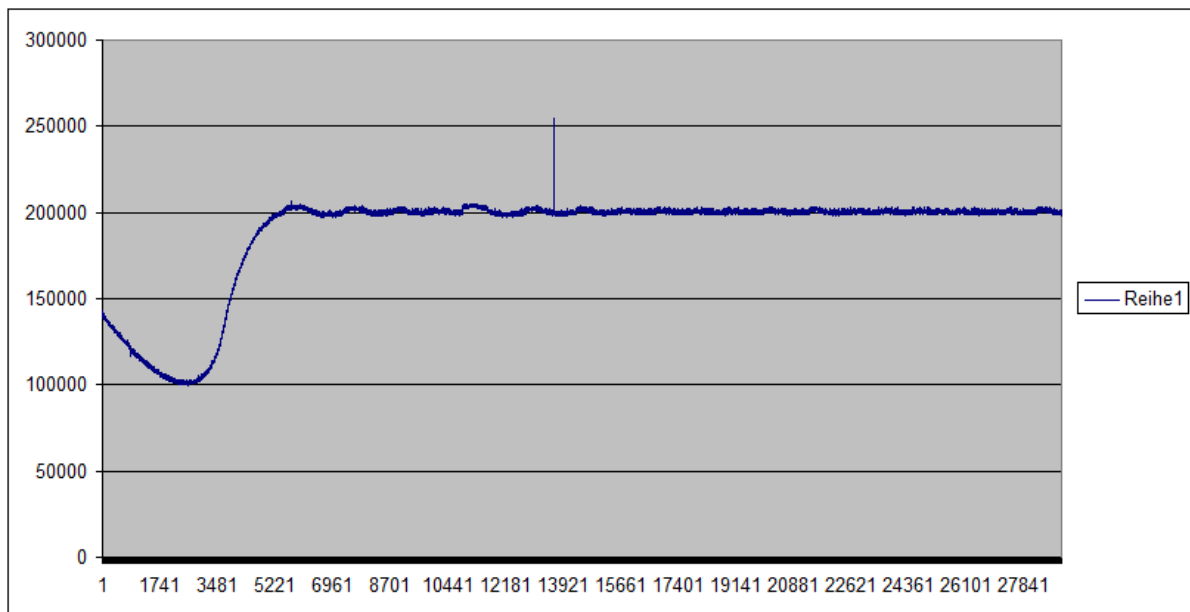


Diagram 2: Drift in ppm (part per million)

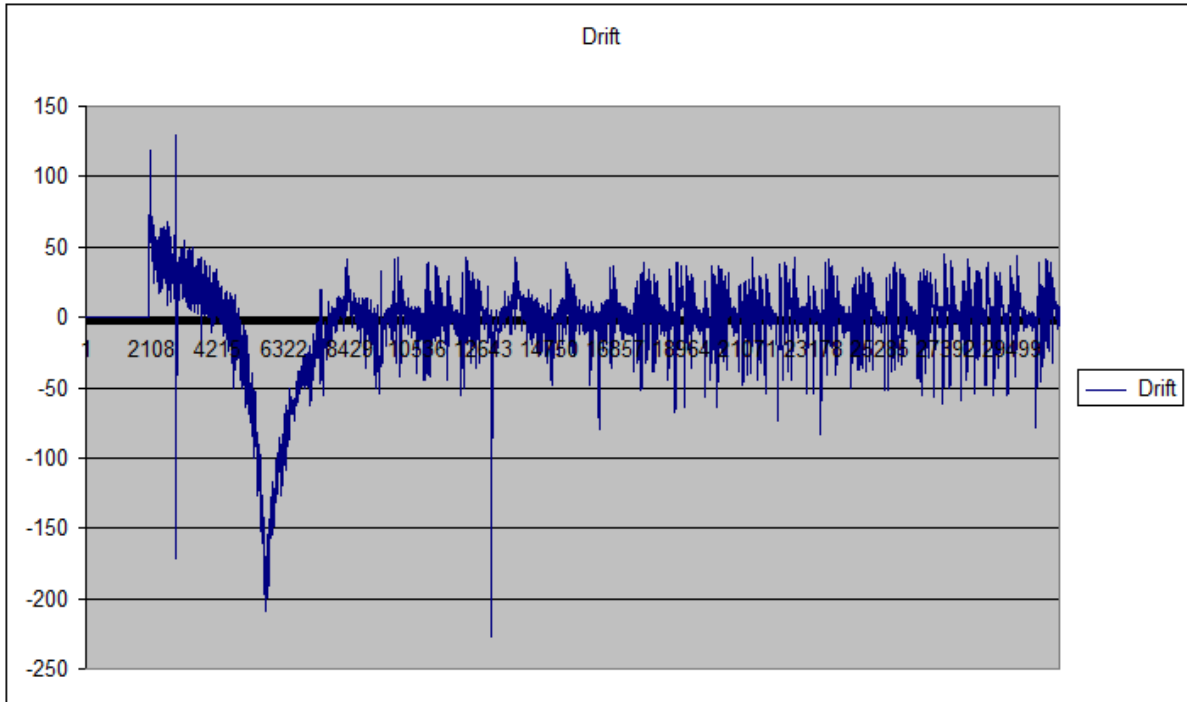


Diagram 3: Controller error in nanoseconds

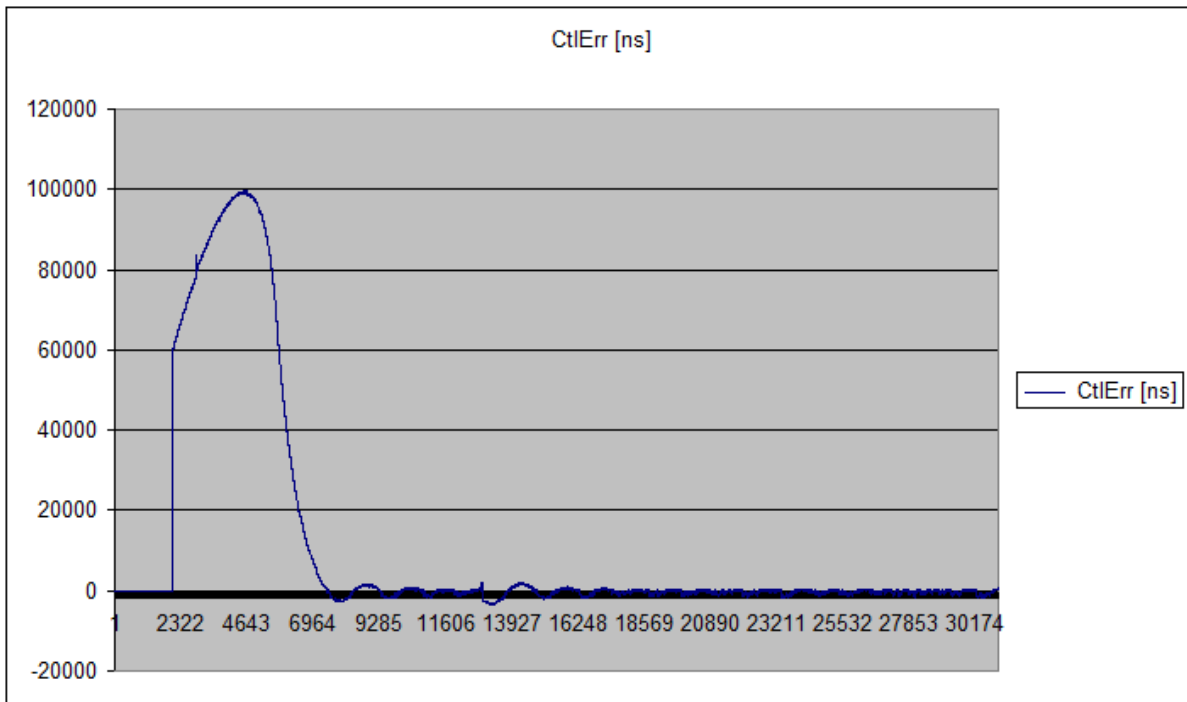
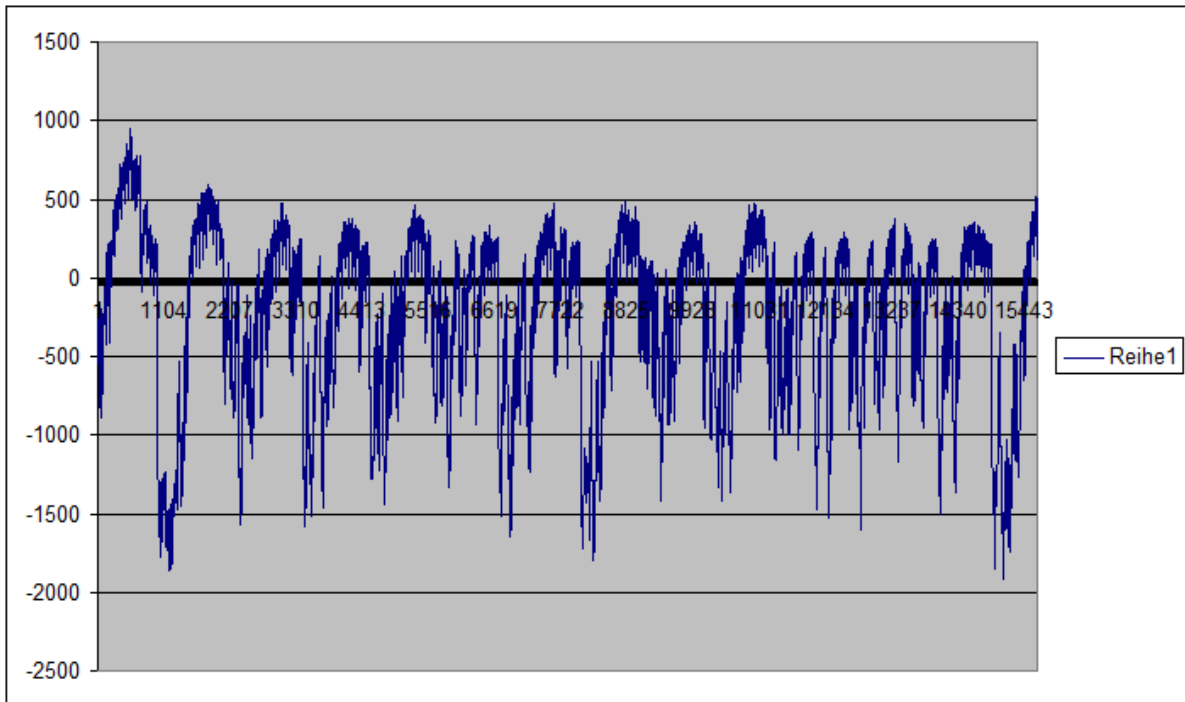


Diagram 4: Controller error in steady state in nanoseconds



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 600 ppm (333333 Hz vs 333000 Hz. E.g. at 1 ms, the cycle time must be between 999.400  $\mu$ s and 1000.600  $\mu$ s)
- 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

## 2.1.5 DCM MainDevice 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 MainDevice shift must not be enabled in the ENI file because it doesn't need any cyclic command. This mode can be activated using `emDcmConfigure()`.

## 2.1.6 DCM MainDevice Ref Clock mode

The DCM MainDevice 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 CPUs. 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 uses the same cyclic command to synchronize the EtherCAT® network with the MainDevice system. This mode can be activated using `emDcmConfigure()`.

## 2.1.7 DCM Linklayer Ref Clock mode

In this mode the Real-time Ethernet Driver 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 SubDevices and during the SubDevice transition PREOP to SAFEOP to start the DC SYNC signals if needed.

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

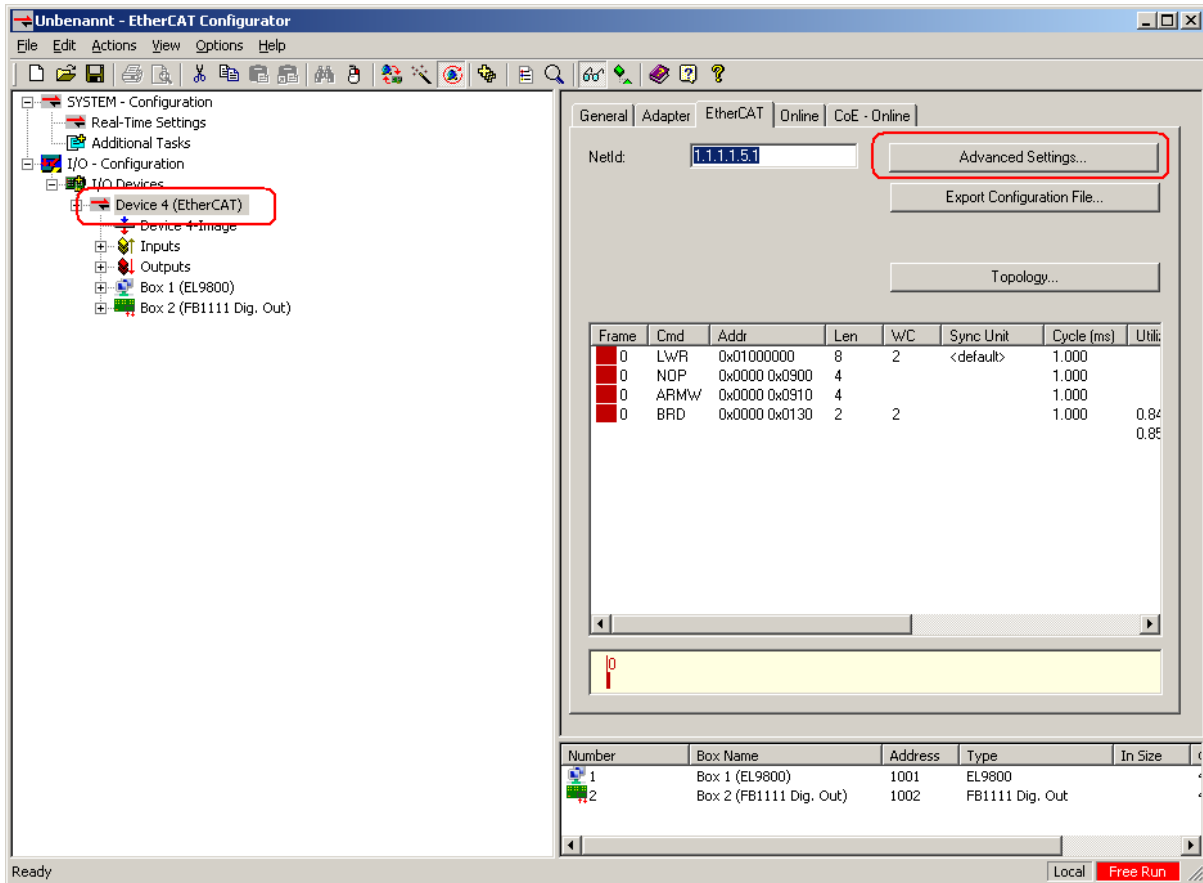
During the call to `EcLinkSendFrame`, the Real-time Ethernet Driver should insert the send time of the frame following the instruction given by `EC_T_LINK_FRAMEDESC::wTimestampOffset` and `EC_T_LINK_FRAMEDESC::wTimestampSize` of the parameter `pLinkFrameDesc`. A value of 0 means that no time stamp should be inserted.

## 2.2 Configuration with ET9000

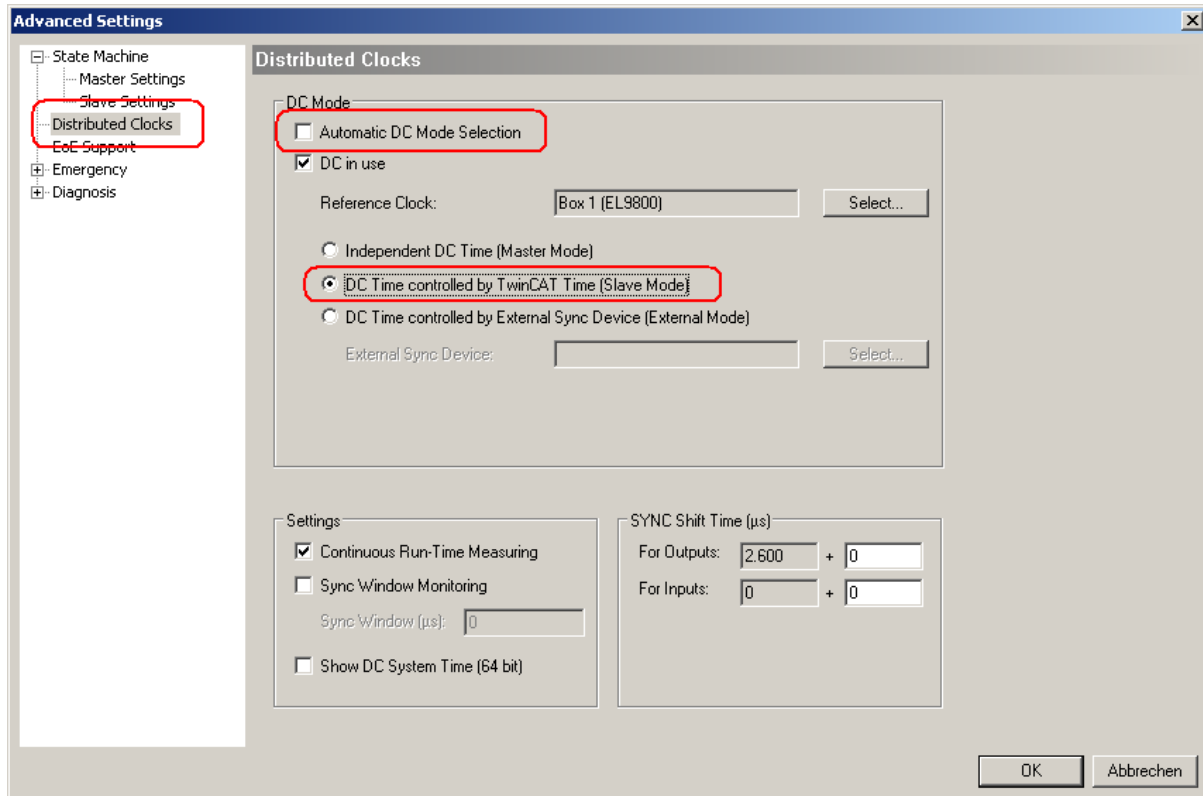
Since version 2.11.0 of the EtherCAT® Configurator from Beckhoff explicitly setup whether the DC time shall be controlled by the EC-Master or not.

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

1. Scan the EtherCAT® Bus
2. Select the EtherCAT® network adapter 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 (SubDevice Mode)”.



4. Now the DC time can be controlled by the EC-Master.

---

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

---

## 2.3 Programmer's Guide

### 2.3.1 emDcmConfigure

```
static EC_T_DWORD ecatDcmConfigure (
    EC_T_DCM_CONFIG *pDcmConfig,
    EC_T_DWORD dwInSyncTimeout
)
EC_T_DWORD emDcmConfigure (
    EC_T_DWORD dwInstanceID,
    EC_T_DCM_CONFIG *pDcmConfig,
    EC_T_DWORD dwInSyncTimeout
)
    Configure DC master synchronization.
```

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pDcmConfig** – [in] Configuration information, a pointer to a structure of type *EC\_T\_DCM\_CONFIG*
- **dwInSyncTimeout** – [in] Currently not implemented

**Returns**

EC\_E\_NOERROR or error code

struct **EC\_T\_DCM\_CONFIG**

**Public Members**

*EC\_T\_DCM\_MODE* **eMode**

[in] DCM mode

*EC\_T\_DCM\_CONFIG\_BUSSHIFT* **BusShift**

[in] BusShift configuration. Valid if eMode is set to eDcmMode\_BusShift.

*EC\_T\_DCM\_CONFIG\_MASTERSHIFT* **MasterShift**

[in] MasterShift configuration. Valid if eMode is set to eDcmMode\_MasterShift.

*EC\_T\_DCM\_CONFIG\_LINKLAYERREFCLOCK* **LinkLayerRefClock**

[in] LinkLayerRefClock configuration. Valid if eMode is set to eDcmMode\_LinkLayerRefClock.

*EC\_T\_DCM\_CONFIG\_MASTERREFCLOCK* **MasterRefClock**

[in] MasterRefClock configuration. Valid if eMode is set to eDcmMode\_MasterRefClock.

*EC\_T\_DCM\_CONFIG\_DCX* **Dcx**

[in] DCX configuration. Valid if eMode is set to eDcmMode\_Dcx.

enum **EC\_T\_DCM\_MODE**

*Values:*

enumerator **eDcmMode\_Off**

DCM disabled

enumerator **eDcmMode\_BusShift**

DCM BusShift mode

enumerator **eDcmMode\_MasterShift**

DCM MasterShift mode

enumerator **eDcmMode\_LinkLayerRefClock**

DCM LinkLayer Ref Clock mode

enumerator **eDcmMode\_MasterRefClock**

DCM Master Ref Clock mode

enumerator **eDcmMode\_Dcx**

DCM DCX External synchronization mode

enumerator **eDcmMode\_MasterShiftByApp**

DCM MasterShift controlled by application mode

struct **EC\_T\_DCM\_CONFIG\_BUSSHIFT**

## Public Members

### EC\_T\_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).

### EC\_T\_INT **nCtlGain**

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

### EC\_T\_INT **nCtlDriftErrorGain**

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

### EC\_T\_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.

### EC\_T\_BOOL **bLogEnabled**

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

### EC\_T\_DWORD **dwInSyncLimit**

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

### EC\_T\_DWORD **dwInSyncSettleTime**

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

### EC\_T\_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.

### EC\_T\_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.

### EC\_T\_DWORD **dwInSyncStartDelayCycle**

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

### EC\_T\_VOID **\*pGetTimeElapsedSinceCycleStartContext**

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

### EC\_PF\_DC\_GETTIMEELAPSEDSINCECYCLESTART **pfnGetTimeElapsedSinceCycleStart**

[in] Setting this function enables synchronising the DC SYNC0 to the CycleStart instead of the time frame send. DC SYNC0 is therefore related to the CycleStart instead of the time frame send. Function must return EC\_E\_NOERROR to work. If 0 (nullptr, EC\_NULL) EC\_E\_NOTSUPPORTED will be returned on attempt to call.

struct **EC\_T\_DCM\_CONFIG\_MASTERSHIFT**

## Public Members

### EC\_T\_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).

### EC\_T\_INT **nCtlGain**

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

### EC\_T\_INT **nCtlDriftErrorGain**

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

### EC\_T\_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.

### EC\_T\_BOOL **bLogEnabled**

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

### EC\_T\_DWORD **dwInSyncLimit**

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

### EC\_T\_DWORD **dwInSyncSettleTime**

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

### EC\_T\_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.

### EC\_T\_DWORD **dwInSyncStartDelayCycle**

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

### *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.

### EC\_T\_VOID **\*pGetTimeElapsedSinceCycleStartContext**

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

### EC\_PF\_DC\_GETTIMEELAPSEDSINCECYCLESTART **pfnGetTimeElapsedSinceCycleStart**

[in] Setting this function enables synchronising the DC SYNC0 to the CycleStart instead of the time frame send. DC SYNC0 is therefore related to the CycleStart instead of the time frame send. Function must return EC\_E\_NOERROR to work. If 0 (nullptr, EC\_NULL) EC\_E\_NOTSUPPORTED will be returned on attempt to call.

### EC\_T\_VOID **\*pAdjustCycleTimeContext**

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

**EC\_PF\_DC\_ADJUSTCYCLETIME pfnAdjustCycleTime**

[in] Mandatory function to adjust the cycle time of the master. The master must adjust its cycle time by the passed amount to be in sync with the slaves. If 0 (nullptr, EC\_NULL) MasterShift will not work.

struct **EC\_T\_DCM\_CONFIG\_LINKLAYERREFCLOCK**

**Public Members****EC\_T\_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).

**EC\_T\_BOOL bLogEnabled**

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

**EC\_T\_DWORD dwInSyncLimit**

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

**EC\_T\_DWORD dwInSyncSettleTime**

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

***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.

struct **EC\_T\_DCM\_CONFIG\_MASTERREFCLOCK**

**Public Members****EC\_T\_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).

**EC\_T\_BOOL bLogEnabled**

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

**EC\_T\_DWORD dwInSyncLimit**

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

**EC\_T\_DWORD dwInSyncSettleTime**

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

**EC\_T\_VOID \*pGetHostTimeContext**

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

**EC\_PF\_DC\_GETHOSTTIME pfnGetHostTime**

[in] Mandatory function to offer the host time for the MasterRefClock mode. This time will be distributed to the slaves. If 0 (nullptr, EC\_NULL) MasterRefClock mode will not work.

struct **EC\_T\_DCM\_CONFIG\_DCX**

Contains the configuration information for the DCX external synchronization mode.

**See also:**

Feature Pack “External Synchronization” for further details.

struct **EC\_T\_DC\_STARTTIME\_CB\_DESC**

**Public Members**

EC\_T\_VOID \***pvContext**

[in] Arbitrarily application-defined parameter passed to pfnCallback

**EC\_PF\_DC\_STARTTIME\_CB pfnCallback**

[in] DC start time callback function pointer. If not EC\_NULL, DC start time calculated by application, otherwise by master.

typedef EC\_T\_DWORD (\***EC\_PF\_DC\_STARTTIME\_CB**)(EC\_T\_VOID \*pvContext, EC\_T\_WORD wSlaveFixedAddr, EC\_T\_UINT64 \*pqwDcStartTime)

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.

**Param pvContext**

[in] Arbitrarily application-defined parameter passed to callback

**Param wSlaveFixedAddr**

[in] Slave fixed address

**Param pqwDcStartTime**

[out] DC start time for specific slave

**Return**

EC\_E\_NOERROR or error code

**2.3.2 emDcmGetStatus**

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

```

EC_T_DWORD emDcmGetStatus (
    EC_T_DWORD dwInstanceID,
    EC_T_DWORD *pdwErrorCode,
    EC_T_INT *pnDiffCur,
    EC_T_INT *pnDiffAvg,
    EC_T_INT *pnDiffMax
)

```

Get DC master synchronization controller status.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pdwErrorCode** – [out] Pointer to current error code of the DCM controller. Possible values are:
  - EC\_E\_NOTREADY DCM control loop is not running
  - EC\_E\_BUSY DCM control loop is running and try to get InSync
  - *DCM\_E\_MAX\_CTL\_ERROR\_EXCEED* Set if the controller error exceeds the InSyncLimit
  - *DCM\_E\_DRIFT\_TO\_HIGH* DCM control loop not able to compensate drift. Drift above 600ppm
- **pnDiffCur** – [out] Pointer to current difference between set value and actual value of controller in nanoseconds
- **pnDiffAvg** – [out] Pointer to average difference between set value and actual value of controller in nanoseconds
- **pnDiffMax** – [out] Pointer to maximum difference between set value and actual value of controller in nanoseconds

#### Returns

- 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)

### 2.3.3 emDcmResetStatus

```
static EC_T_DWORD ecatDcmResetStatus (EC_T_VOID)
```

```
EC_T_DWORD emDcmResetStatus (EC_T_DWORD dwInstanceID)
```

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

#### Parameters

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

#### Returns

- EC\_E\_NOERROR or error code

### 2.3.4 emDcmGetBusShiftConfigured

static EC\_T\_DWORD **ecatDcmGetBusShiftConfigured** (EC\_T\_BOOL \*pbBusShiftConfigured)

EC\_T\_DWORD **emDcmGetBusShiftConfigured** (  
 EC\_T\_DWORD dwInstanceID,  
 EC\_T\_BOOL \*pbBusShiftConfigured  
 )

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

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pbBusShiftConfigured** – [out] EC\_TRUE if DCM bus shift mode is supported by the current configuration

#### Returns

EC\_E\_NOERROR or error code

### 2.3.5 emDcmGetLog

static EC\_T\_DWORD **ecatDcmGetLog** (EC\_T\_CHAR \*\*pszLog)

EC\_T\_DWORD **emDcmGetLog** (EC\_T\_DWORD dwInstanceID, EC\_T\_CHAR \*\*pszLog)  
 Get logging information from the DCM controller.

This function returns non-zero pointer only if bLogEnabled was set to EC\_TRUE in *EC\_T\_DCM\_CONFIG*.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pszLog** – [out] Pointer to a string containing the current logging information

#### Returns

EC\_E\_NOERROR or error code

#### See also:

*Controller adjustment* for content description of pszLog

### 2.3.6 emIoctl - EC\_IOCTL\_DCM\_GET\_LOG

**EC\_IOCTL\_DCM\_GET\_LOG**

Get logging information from the DCM controller.

#### Parameters

- **pbyInBuf** – [in] Should be set to EC\_NULL
- **dwInBufSize** – [in] Should be set to 0
- **pbyOutBuf** – [out] Pointer to struct *EC\_T\_DCM\_LOG*
- **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.

#### Returns

EC\_E\_NOERROR or error code

struct **EC\_T\_DCM\_LOG**

### Public Members

**EC\_T\_DWORD dwMsecCounter**  
[out] Current MsecCounter

**EC\_T\_INT nCtlSetVal**  
[out] Configured controller set val [ns]

**EC\_T\_UINT64 qwBusTime**  
[out] Current BusTime

**EC\_T\_INT nCtlErrorNsec**  
[out] Current controller error [ns]

**EC\_T\_INT nDrift**  
[out] Current calculated drift [ppm]

**EC\_T\_DWORD dwErrorCode**  
[out] Last returned error code by controller

**EC\_T\_BOOL bDcmInSync**  
[out] EC\_TRUE if DCM is in sync, EC\_FALSE if out of sync

**EC\_T\_BOOL bDcInSync**  
[out] EC\_TRUE if DC is in sync, EC\_FALSE if out of sync

**EC\_T\_UINT64 qwDcStartTime**  
[out] Last used DC StartTime

**EC\_T\_INT nSystemTimeDifference**  
[out] Last read System Time Difference (ESC register 0x092C)

#### See also:

`emIoCtl ()` in the [EC-Master Class B](#) documentation

## 2.3.7 emDcmShowStatus

static **EC\_T\_DWORD ecatDcmShowStatus** (**EC\_T\_VOID**)

**EC\_T\_DWORD emDcmShowStatus** (**EC\_T\_DWORD** dwInstanceID)  
Show DC master synchronization status as DbgMsg (for development purposes only).

#### Parameters

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

#### Returns

**EC\_E\_NOERROR** or error code

### 2.3.8 emDcmGetAdjust

static EC\_T\_DWORD **ecatDcmGetAdjust** (EC\_T\_INT \*pnAdjustPermil)

EC\_T\_DWORD **emDcmGetAdjust** (EC\_T\_DWORD dwInstanceID, EC\_T\_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.

#### Parameters

- **dwInstanceID** – [in] Instance ID (Multiple EtherCAT Network Support)
- **pnAdjustPermil** – [out] Current adjustment value of the timer

#### Returns

EC\_E\_NOERROR or error code

### 2.3.9 DCM specific error codes

#### DCM\_E\_ERROR

0x981201C0: Unspecific DCM Error

#### DCM\_E\_NOTINITIALIZED

0x981201C1: Not initialized

#### DCM\_E\_MAX\_CTL\_ERROR\_EXCEED

0x981201C2: DCM controller - synchronization out of limit

#### DCM\_E\_NOMEMORY

0x981201C3: Not enough memory

#### DCM\_E\_INVALID\_HWLAYER

0x981201C4: Hardware layer - (BSP) invalid

#### DCM\_E\_TIMER\_MODIFY\_ERROR

0x981201C5: Hardware layer - error modifying timer

#### DCM\_E\_TIMER\_NOT\_RUNNING

0x981201C6: Hardware layer - timer not running

#### DCM\_E\_WRONG\_CPU

0x981201C7: Hardware layer - function called on wrong CPU

#### DCM\_E\_INVALID\_SYNC\_PERIOD

0x981201C8: Invalid DC sync period length (invalid clock master?)

#### DCM\_E\_INVALID\_SETVAL

0x981201C9: DCM controller SetVal too small

#### DCM\_E\_DRIFT\_TO\_HIGH

0x981201CA: DCM controller - Drift between local timer and ref clock to high

#### DCM\_E\_BUS\_CYCLE\_WRONG

0x981201CB: DCM controller - Bus cycle time (dwBusCycleTimeUseC) doesn't match real cycle

**DCX\_E\_NO\_EXT\_CLOCK**

0x981201CC: DCX controller - No external synchronization clock found

**DCM\_E\_INVALID\_DATA**

0x981201CD: DCM controller - Invalid data

**2.3.10 Notifications**

At startup the MainDevice raises the notifications *emNotify* - *EC\_NOTIFY\_DC\_SLV\_SYNC*, *emNotify* - *EC\_NOTIFY\_DC\_STATUS* and *emNotify* - *EC\_NOTIFY\_DCM\_SYNC* at MainDevice state transition from INIT to PREOP.

The order is typically as follows ( *emNotify* - *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) [...]

**2.3.11 emNotify - EC\_NOTIFY\_DCM\_SYNC**

DCM InSync notification.

This notification is enabled by default.

**emNotify - EC\_NOTIFY\_DCM\_SYNC****Parameter**

- *pbyInBuf*: [in] Pointer to notification descriptor *EC\_T\_DCM\_SYNC\_NTIFY\_DESC*
- *dwInBufSize*: [in] *sizeof(EC\_T\_DCM\_SYNC\_NTIFY\_DESC)*
- *pbyOutBuf*: [out] Should be set to *EC\_NULL*
- *dwOutBufSize*: [in] Should be set to 0
- *pdwNumOutData*: [out] Should be set to *EC\_NULL*

struct **EC\_T\_DCM\_SYNC\_NTIFY\_DESC**

## Public Members

### EC\_T\_DWORD **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.

### EC\_T\_INT **nCtlErrorNsecCur**

[in] Current difference [ns] between set value and actual value of controller

### EC\_T\_INT **nCtlErrorNsecAvg**

[in] Average difference [ns] between set value and actual value of controller

### EC\_T\_INT **nCtlErrorNsecMax**

[in] Maximum difference [ns] between set value and actual value of controller

### See also:

emIoCtl - EC\_IOCTL\_SET\_NOTIFICATION\_ENABLED in the [EC-Master Class B](#) documentation for controlling the deactivation

## 2.4 Code example

The example program EcMasterDemoDc demonstrates, in contrast to EcMasterDemo, how to programmatically configure DCM parameters and how the application can generate the DCM log.

### DCM Configuration (optional)

```

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

/* configure the master */
dwRes = ecacConfigureNetwork(eCnfType_Filename, (EC_T_BYTE*)"ENI.xml", (EC_T_
↳DWORD)OsStrlen("ENI.xml"));

/* register client */
EC_T_REGISTERRESULTS oRegisterClientResults;
OsMemset(&oRegisterClientResults, 0, sizeof(EC_T_REGISTERRESULTS));
dwRes = ecacRegisterClient(EcMasterNotifyCallback, pAppContext, &
↳oRegisterClientResults);

/* configure DC */
EC_T_DC_CONFIGURE oDcConfigure;
OsMemset(&oDcConfigure, 0, sizeof(EC_T_DC_CONFIGURE));
oDcConfigure.dwTimeout = ETHERCAT_DC_TIMEOUT;
dwRes = ecacDcConfigure(&oDcConfigure);

/* configure DCM Bus Shift */
EC_T_DCM_CONFIG oDcmConfig;
OsMemset(&oDcmConfig, 0, sizeof(EC_T_DCM_CONFIG));
oDcmConfig.eMode = eDcmMode_BusShift;
oDcmConfig.u.BusShift.nCtlSetVal = DCM_CONTROLLER_SETVAL_NANOSEC;
oDcmConfig.u.BusShift.bLogEnabled = EC_TRUE;
dwRes = ecacDcmConfigure(&oDcmConfig, 0 /* dwInSyncTimeout */);

/* set EtherCAT devices to OPERATIONAL state */
dwRes = ecacSetMasterState(ETHERCAT_STATE_CHANGE_TIMEOUT, eEcatState_OP);

```

**See also:**`emDcConfigure()`**See also:**`emDcmConfigure()`**DCM Log (optional)**

```
do
{
    /* wait for next cycle (event from scheduler task) */
    /* ... */

    /* process all received frames (read new input values) */
    dwRes = ecatExecJob(eUsrJob_ProcessAllRxFrames, &oJobParms);

    /* ... */

    EC_T_CHAR* szDcmLog = EC_NULL;
    ecatDcmGetLog(&szDcmLog);
    if (EC_NULL != szDcmLog)
    {
        ((CAtEmLogging*)pEcLogContext)->LogDcm(szDcmLog);
    }

    /* ... */
} while (!pAppContext->bJobTaskShutdown);
```

**See also:**`emDcmGetLog()`

### 3 Running EcMasterDemoDc

The EcMasterDemoDc is available “out of the box” for different operating systems. It is an EC-Master example application that handles the following tasks:

- Showing basic EtherCAT® communication
- MainDevice stack initialization into OPERATIONAL state
- DC and DCM configuration
- Process Data operations for e.g. Beckhoff EL2004, EL1004 and EL4132
- Periodic diagnosis task
- Periodic Job Task in polling mode
- Logging

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

```
> EcMasterDemoDc -ndis 192.168.157.2 1 -f eni.xml -t 0 -v 3 -dcmmode busshift
```

#### 3.1 Command line parameters

```
EcMasterDemoDc <LinkLayer> [-f ENI-FileName] [-t time] [-b cycle time] [-a affinity] [-v level] [-perf [level]] [-log prefix [msg cnt]] [-lic key] [-oem key] [-maxbussslaves cnt] [-flash address] [-printvars] [-sp [port]] [-ctloff] [-rec [prefix [frame cnt]]] [-junctionred] [-dcmmode <mode> [<synctocyclestart>]] [-dcmlog]
```

The parameters are as follows:

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

**-t** <time>  
Running duration [ms]. When the time expires the demo application exits completely.

**<time>**  
Time [ms], 0 = forever (default = 120000)

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

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

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

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

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

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

**-perf** [<level>]  
Enable max. and average time measurement in µs for all EtherCAT® jobs (e.g. ProcessAllRxFrames).

- <level>**  
Depending on level the performance histogram can be activated as well.
- log** **<prefix>** [**<msg cnt>**]  
Use given file name prefix for log files.
- <prefix>**
- <msg cnt>**  
Messages count for log buffer allocation
- lic** **<key>**  
Set License key.
- <key>**  
License key string
- oem** **<key>**  
Use OEM key
- <key>**  
64 bit OEM key.
- junctionred**  
Enable junction redundancy (automatic mode)
- flash** **<address>**  
Flash outputs
- <address>**  
0=all, >0 = SubDevice station address
- sp** [**<port>**]  
If platform has support for IP Sockets, this command-line option enables the Remote API Server to be started. The Remote API Server is going to listen on TCP Port 6000 (or port parameter if given) and is available for connecting Remote API Clients.
- <port>**  
RAS server port
- rec** [**<prefix>** [**<frame cnt>**]]  
Packet capture file recording
- <prefix>**  
File name prefix
- <frame cnt>**  
Frame count for log buffer allocation
- dcmmode** **<mode>** [**<synctocyclestart>**]  
Set DCM mode
- <mode>**  
off | busshift | mastershift | masterrefclock | linklayerrefclock
- <synctocyclestart>**  
Sync to cycle start: 0 = disabled (default), 1 = enabled
- dcmllog**  
Enable DCM logging (default: disabled)
- ctlloff**  
Disable DCM control loop for diagnosis