Protocol API

EtherCAT Master V4

V4.5.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC150601API06EN | Revision 6 | English | 2020-09 | Released | Public

Table of contents

1 Introduction 5
1.1 About this document 5
1.2 List of revisions 5
1.3 Functional overview 6
1.4 System requirements 6
1.5 Intended audience 6
1.6 Specifications 7
1.6.1 Technical Data 7
1.7 Terms, abbreviations and definitions 9
1.8 References 9

2 Getting started / Configuration 10
2.1 Configuration of the master 10
2.1.1 Using the configuration tool SYCON.net 10
2.1.2 Detailed description of master parameters 11

3 Overview 12
3.1 Task structure of the EtherCAT Master V4 stack 12
3.2 Diagnosis 13
3.2.1 DC diagnostics 13
3.2.2 Slave diagnostic information 13
3.2.3 Diagnostic log 13
3.2.4 Bus scan 14
3.2.5 LEDs controlled by EtherCAT Master 15
3.3 Process Data Reception 17
3.4 Network topology recommendations 17
3.4.1 Store-and-forward switches not supported 17

4 The application interface 18
4.1 Addressing schemes used in EtherCAT Master 18
4.1.1 Auto-increment address 18
4.1.2 Fixed station address 18
4.1.3 Topology position 19
4.1.4 Device index 19
4.2 Distributed Clocks 20
4.3 Synchronization configuration 20
4.3.1 Synchronization modes 20
4.3.2 Packets 23
4.4 State control 25
4.4.1 Architecture of master state control 25
4.4.2 Architecture of slave state control 26
4.4.3 Master state 27
4.4.4 Master state (Legacy) 32
4.4.5 Slave state 36
4.5 Status indications 41
4.5.1 Registration and deregistration of status indications 41
4.5.2 Available indications 43
4.6 Diagnostic log 47
4.6.1 Entry format of diagnostic log 47
4.6.2 Reading and clearing diagnostic log entries 62
4.6.3 Diagnostic log indication handling 65
4.7 CoE services 70
4.7.1 Slave state accessibility 70
4.7.2 SDO access 70
4.7.3 SDOINFO access 80
4.7.4 SDO access (Legacy) 97
4.7.5 SDOINFO access (Legacy) 101
4.8 FoE services 108
4.8.1 Slave state accessibility 108
4.8.2 Fragmentation of write file (FoE) 108
4.8.3 Fragmentation of read IDN (SoE) 110
4.8.4 Packets 112
4.8.5 FoE fragmentation flowcharts 118

4.9 SoE services 120
4.9.1 Slave state accessibility 120
4.9.2 Fragmentation of write IDN (SoE) 120
4.9.3 Fragmentation of read IDN (SoE) 122
4.9.4 Packets 124
4.9.5 SoE fragmentation flowcharts 130
4.10 Distributed Clocks diagnostics 132
4.10.1 Packets 132
4.10.2 Legacy packets 139
4.11 Config readout 140
4.11.1 Get timing information 140
4.11.2 Get WcState information 141
4.11.3 Get cyclic command mapping 145
4.11.4 Get cyclic slave mapping 152
4.12 Retrieval of slave diagnostic information 156
4.12.1 Provided lists 156
4.12.2 Limitations of configured slaves list 156
4.12.3 Limitations of active/faulted slaves list 156
4.12.4 Addressing scheme 156
4.12.5 Usage of slave diagnostic information packets 157
4.12.6 Structure of per slave diagnostic data 158
4.12.7 Packets 160
4.13 Retrieval of topology information 168
4.13.1 Get topology information entries 168
4.13.2 Get topology information packet 169
4.14 ESC/SII access 170
4.14.1 ESC register access 170
4.14.2 ESC SII access 174
4.14.3 Legacy ESC SII access (ECM V3.X API) 178
4.15 ExtSync 183
4.15.1 Description of ExtSync functionality 183
4.15.2 Get ExtSync information packet 184
4.15.3 Reset ExtSync Max Deviations 187
4.16 Sync with external Pin 188
4.16.1 Select Sync for External Pin 188
4.16.2 Enumeration of available sync and timesync modules 191
4.17 Reading frame error counters 194
4.18 Bus scan 196
4.18.1 Packet parameter `ulPortState` 196
4.18.2 Generic bus scan 197
4.18.3 Legacy bus scan 201

5 Feature Configuration via Tag List 207
5.1 EtherCAT Master Tag List Parameter 207

6 Status/Error codes overview 208
6.1 EtherCAT Master packet status codes 208
6.2 EtherCAT Master V4 LLD error codes 214
6.3 EtherCAT Master V4 EMC error codes 216
6.4 EtherCAT Master V4 AoE error codes 224
6.5 EtherCAT Master V4 CoE error codes 226
6.6 EtherCAT Master V4 EoE error codes 228
6.7 EtherCAT Master V4 FoE error codes 229
6.8 EtherCAT Master V4 SoE error codes 230
6.9 EtherCAT Master V4 ENI error codes 233
6.10 EtherCAT Master V4 AL Status codes 239
6.11 EtherCAT Master V4 IF error codes 241
6.12 EtherCAT Master V4 AP Task error codes 242

7 Appendix 243
7.1 Accessing the protocol stack by programming the AP task's queue 243
7.1.1 Getting the receiver task handle of the process queue 243
7.2 Extended status 244
7.3 Legal notes 245
7.4 List of figures 249
7.5 List of tables 250
7.6 Contacts 254

1 Introduction

1.1 About this document

This manual describes the application interface of the EtherCAT Master protocol stack. The aim of this manual is to support and guide you through the integration process of the given stack into your own application.

1.2 List of revisions

Rev	Date	Name	Revisions
5	2017-01-23	SB, HH	EtherCAT Master V4.4.0
			Section *FoE services* added.
6	2020-09-22	HHE	EtherCAT Master V4.5.0
			Section *Slave current state indication* added.
			Section *SDOINFO access*: Note about timeout added.
			Section *Sync with external Pin* added.
			Section *Reading frame error counters* added.
			Section *Feature Configuration via Tag List* added.
			Section *Status/Error codes overview* updated.

Table 1: List of Revisions

1.3 Functional overview

The main functionality from application view is:

- configure master and bus
- exchange of cyclic data
- slave diagnosis

1.4 System requirements

This software package has following system requirements to its environment:

- netX-Chip as CPU hardware platform

1.5 Intended audience

This manual is suitable for software developers with the following background:

- Knowledge of the programming language C
- Knowledge of the use of the real-time operating system rcX
- Knowledge of the Hilscher Task Layer Reference Model
- Knowledge of the netX DPM Interface
- Knowledge of the IEC 61158 Part 2-6 Type 12 specification documents

1.6 Specifications

The data below applies to the EtherCAT Master firmware and stack version V4.5.0.

1.6.1 Technical Data

Technical Data

Maximum number of cyclic input data	about 4600 Bytes if no LRW command is used for process data
Maximum number of cyclic output data	about 4600 Bytes if no LRW command is used for process data
Maximum number of supported slaves	388 if using HIL_GET_SLAVE_HANDLES_REQ service is used for determining number of slaves
Minimum bus cycle time	250 microseconds
Acyclic communication	CoE (CANopen over EtherCAT) SDO, SDOINFO, Emergencies
	SoE (Servo Drive Profile over EtherCAT)
	EoE (Ethernet over EtherCAT)
Functions	Slave diagnostics
Topology	Line Ring (since V4.4)
Distributed Clocks	supported on all supported topologies
Data transport layer	Ethernet II, IEEE 802.3, 100MBit/s Full-Duplex
Size of CONFIG.NXD file	Max. about 1 MByte
Size of ETHERCAT.XML file	Max. about 1 MByte on cifX50 (RAM Disk limit) Max. about 3 MByte on Flash-based devices with 4MByte chip
Bus scan	supported
Allowed range of slave station addresses	1 – 14335
Mailbox protocols	CoE, EoE, FoE, SoE
Synchronization via ExtSync	supported
ENI Slave-to-slave copy infos	supported

Firmware/stack available for netX

netX 50	no
netX 100, netX 500	yes

Limitations

- The size of the bus configuration file is limited by the size of the RAM Disk (1 Megabyte) on cifX50
- Store-and-Forward Switches cannot be used within network topology due to hard receive timing model
- EtherCAT Master V4 uses INTRAM3 and XM3_IO1 on netX100/500, so Xc3 cannot be used for other protocols.
- HIL_GET_SLAVE_HANDLES_REQ can only communicate up to 388 slaves.
- Process data on LFW is restricted by the DPM memory definition to 5760 bytes

Configurator Limitations

- SyCON.net
  - Currently, only CoE can be configured.
  - No ExtSync support within SyCON.net (no PDO information)
  - No ability to configure CopyInfos
  - BOOT states of supporting slaves cannot be entered (no SyncManager configuration)

1.7 Terms, abbreviations and definitions

Term	Description
AoE	ADS over EtherCAT
AP (-task)	Application (-task) on top of the stack
CoE	CANopen over EtherCAT
DC	Distributed Clocks
DDF	Data Description File
DPM	Dual Port Memory
EEPROM	Electronically Erasable Programmable Read-Only Memory
ESC	EtherCAT Slave Controller
ESM	EtherCAT State Machine
ETG	EtherCAT Technology Group
EtherCAT	Ethernet for Control and Automation Technology
FoE	File Access over EtherCAT
HAL	Hardware Abstraction Layer
LFW	Loadable firmware
LOM	Linkable object modules
OD	Object dictionary
PDO	Process Data Object (process data channel)
SDO	Service Data Object (representing an acyclic data channel)
SHM	Shared memory
SII	Slave Information Interface
SoE	Servo Drive Profile over EtherCAT
XML	Extended Markup Language

Table 2: Terms, Abbreviations and Definitions

All variables, parameters and data used in this manual have the LSB/MSB (“Intel”) data format. This corresponds to the convention of the Microsoft C Compiler.

1.8 References

This document based on the following documents respectively specifications:

1	Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX Dual-Port Memory Interface, Revision 17, English, 2020
2	Hilscher Gesellschaft für Systemautomation mbH: Driver Manual, cifX Device Driver, Windows 2000/XP/Vista/7/8/10, Revision 29, English, 2020
3	IEC 61158 Part 2-6 Type 12 specification documents
4	Hilscher Gesellschaft für Systemautomation mbH: Specification - netX IO Synchronization. Revision 6, English, 2010
5	ETG.1020 Protocol Enhancements
6	ETG.1500 Master Classes
7	ETG.2100 Network Information
8	Hilscher Gesellschaft für Systemautomation mbH: Network scan, Revision 5, English, 2017
9	Hilscher Gesellschaft für Systemautomation mbH: Packet API, netX Dual-Port Memory, Packet-based services (netX 10/50/51/52/100/500), Revision 4, English, 2020

Table 3: References

2 Getting started / Configuration

This section explains some essential information you should know when starting to work with the EtherCAT Master V4 Protocol API.

2.1 Configuration of the master

The master can be configured by using different means. This includes the following methods:

- Configuration via SYCON.net
  - Timing parameters are specified by the user
- Configuration via EtherCAT Network Information (ENI) files
  - Timing parameters are extracted from ENI in the specified locations.

The configuration via SYCON.net evaluates the ESI files provided for the slaves to be used and is therefore the easiest method.

2.1.1 Using the configuration tool SYCON.net

The easiest way to configure the EtherCAT Master is using Hilscher's configuration tool SYCON.net.

- First, you need to create a project in SYCON.net. This is described in detail in the SYCON.net documentation.
- Configure the bus and master parameters as described in the SYCON.net documentation.
- After you completed your project, you can right-click on the icon of the EtherCAT Master and select "Connect".
- You will see that the name of the EtherCAT Master will get a green background. Now right-click on the icon again and select "Download".
- This will download the configuration files into the firmware. It is stored on a file system in a channel dependent directory (`"PORT_0"` for channel 0, `"PORT_1"` for channel 1, etc.).
- After the download is finished, the driver requests the EtherCAT Master firmware to perform a Channel-Init. All current connections will be shut down by the firmware and a restart will be performed.
- During this restart, the configuration that has been downloaded previously will be evaluated and used.

2.1.2 Detailed description of master parameters

Both the bus and the master need to be configured. The accurate choice of the bus parameters is the foundation of correctly operating data exchange on the EtherCAT network.

The following table contains relevant information about the bus parameters (including the master's parameters) for the EtherCAT Master V4 firmware such as a short explanation of the meaning of the parameter and ranges of allowed values:

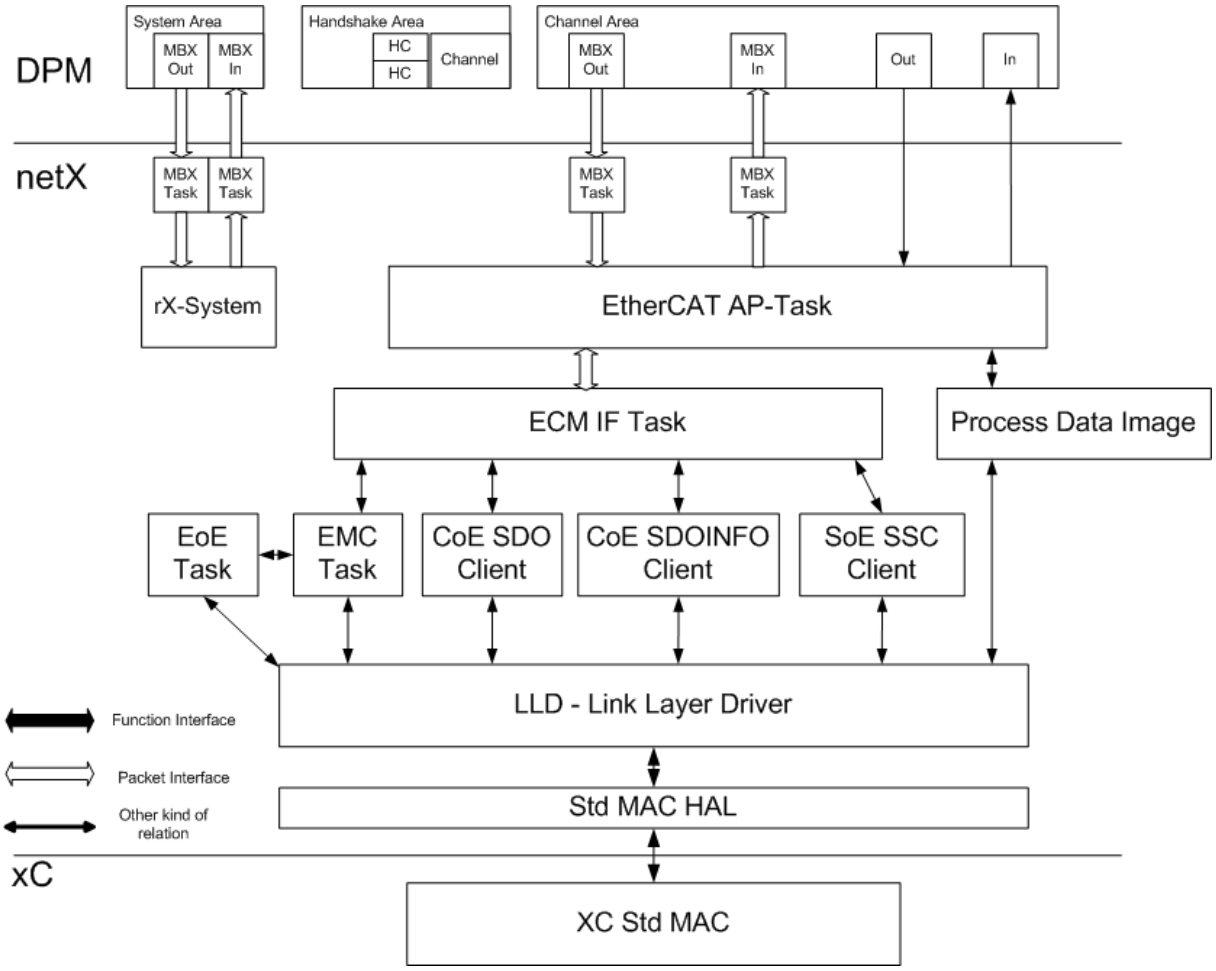
Parameter	Description	Range of Value / Value
Bus Cycle Time	The bus cycle time specifies the actual bus cycle used.	
Process Data Output Size	This parameter determines the size of the area to be used for Process Data Output. It may not exceed the size of available space in DPM which is 5760 bytes.	Minimum Value 0 Maximum Value 5760
Process Data Input Size	This parameter determines the size of the area to be used for Process Data Input. It may not exceed the size of available space in DPM which is 5760 bytes.	Minimum Value 0 Maximum Value 5760

Table 4: Bus and Master Parameters, their Meanings and their Ranges of allowed Values

3 Overview

3.1 Task structure of the EtherCAT Master V4 stack

The illustration below displays the internal structure of the tasks which together represent the EtherCAT Master V4 Stack:



The dual-port memory is used for exchange of information, data and packets. Configuration and IO data will be transferred using this way.

The user application only accesses the task located in the highest layer namely the AP task which constitutes the application interface of the EtherCAT Master V4 stack.

The AP task represents the interface between the EtherCAT Master V4 protocol stack and the dual-port memory. It is responsible for:

- Control of LEDs
- Diagnosis
- Packet routing
- Update of the IO data

The triple buffer mechanism provides a consistent synchronous access procedure from both sides (DPM and AP task). The triple buffer technique ensures that the access will always affect the last written cell.

3.2 Diagnosis

The following diagnostic capabilities are provided by the EtherCAT Master protocol stack:

- Diagnostic log, provides access to EtherCAT-specific diagnostic events
- DC Diagnostics
- Get Bus Info
- Bus scan
- Slave Diagnostic Information

3.2.1 DC diagnostics

The EtherCAT Master provides access to recorded deviations when running with a DC configuration. For details, see section *Distributed Clocks diagnostics* (page 132).

3.2.2 Slave diagnostic information

The slave diagnostic information provides status information on what happened at each slave specifically. For details, see section *Retrieval of slave diagnostic information* (page 156).

3.2.3 Diagnostic log

The EtherCAT Master protocol stack allows the application to be informed about various events such as:

- Change of bus state
- Failure of Init commands
- Failure or warning in slave
- Bus on/bus off
- Channel init
- DPM watchdog error
- Change in topology
- Bus scan
- Internal error

For details about how to use the diagnostic log, see section *Diagnostic log* (page 47).

3.2.4 Bus scan

Bus scan provides the possibility to scan the network for available slaves. The bus scan request overrides the configured mode and switches the master internally to a similar operation mode as the unconfigured mode.

After the bus scan request has completed, the bus scan results can be read from all connected slaves.

For details on using bus scan, see section *Bus scan* (page 196).

3.2.5 LEDs controlled by EtherCAT Master

The following table describes how the LEDs are controlled by EtherCAT Master.

LED	Color	State	Meaning
RUN	**LED green**		
	(off)	Off	**INIT**: The device is in state INIT.
	(green)	Blinking (2,5 Hz)	**PRE-OPERATIONAL**: The device is in PRE-OPERATIONAL state.
	(green)	Flickering (10 Hz)	The device is not configured.
	(green)	Single flash	**SAFE-OPERATIONAL**: The device is in SAFE-OPERATIONAL state.
	(green)	On	**OPERATIONAL**: The device is in OPERATIONAL state.
ERR	**LED red**		
	(red)	Single flash	Bus Sync error threshold
	(red)	Double flash	Internal Stop of the bus cycle
	(red)	Triple Flash	DPM watchdog has expired.
	(red)	Quadruple Flash	No Master license present in the device.
	(red)	Blinking (2,5 Hz)	Error in the configuration database.
	(red)	Single Flickering	Channel Init was executed at the Master. Remarks: Transient error so can happen to be not visible at all.
	(red)	Double Flickering	Slave is missing. Unconfigured Slave No matching mandatory slave list No bus connected
	(red)	Flickering (10 Hz)	Boot-up was stopped due to an error.
	(off)	Off	Master has no errors.
L/A	**LED green**		
	(green)	On	**Link:** The device is linked to the Ethernet, but does not send/receive Ethernet frames.
	(green)	Flickering (load dependent)	**Activity:** The device is linked to the Ethernet and sends/receives Ethernet frames.
	(off)	Off	The device has no link to the Ethernet.

Table 5: LED states for the EtherCAT Master

LED State	Definition
On	The indicator is constantly on.
Off	The indicator is constantly off.
Single flash	The indicator shows one short flash (200 ms) followed by a long “off“ phase (1,000 ms).
Double flash	The indicator shows a sequence of two short flashes (each 200 ms), separated by a short off phase (200 ms). The sequence is finished by a long off phase (1,000 ms).
Triple Flash	The indicator shows a sequence of three short flashes (each 200 ms), separated by a short off phase (200 ms). The sequence is finished by a long off phase (1,000 ms).
Quadruple Flash	The indicator shows a sequence of four short flashes (each 200 ms), separated by a short off phase (200 ms). The sequence is finished by a long off phase (1,000 ms).
Blinking (2,5 Hz)	The indicator turns on and off with a frequency of 2,5 Hz: “on” for 200 ms, followed by “off” for 200 ms.
Single Flickering	The indicator is switched on and off once: 'on' for 50 ms, followed by 'off' for 500 ms.
Double Flickering	The indicator is switched on and off and on once: 'on' / 'off' / 'on' each for approximately 50 ms, followed by 'off' for 500 ms.
Flickering (10 Hz)	The indicator turns on and off with a frequency of 10 Hz: 'on' for 50 ms, followed by 'off' for 50 ms.
Flickering (load dependent)	The indicator turns on and off with a frequency of approximately 10 Hz to indicate high Ethernet activity: on for approximately 50 ms, followed by off for 50 ms. The indicator turns on and off in irregular intervals to indicate low Ethernet activity.

Table 6: LED state definitions for the EtherCAT Master protocol

3.3 Process Data Reception

The EtherCAT master uses a hard timing model to coordinate transmitting and receiving of I/O data.

The timing model is required to implement redundancy and distributed clocks being used together. The master has to combine received data from both ports to a single process data image.

If such a frame arrives outside of the receive end time point, the frame is considered as not being received. If it carries the actual input process data, the frame is considered as not being received and the fall-back behavior is active. This is done by default in order to clear the related input process data area.

The EtherCAT master will start up such a network since acyclic commands are not processed through the cyclic receive handler.

3.4 Network topology recommendations

3.4.1 Store-and-forward switches not supported

Do not use store-and-forward switches on EtherCAT networks in production usage at all. Those switches cause large additional delays within the network.

Depending on the configuration such a store and forward device adds easily up to 50-100 µs of additional cable delay.

The EtherCAT master will start up such a network through the cyclic receive handler since acyclic commands are not processed. However, the input process image will not show such delayed data.

The receive timing calculation cannot know about network components that are not visible as EtherCAT slaves.

3.4.1.1 DC disturbance

Such devices like store-and-forward switches introduce larger delays and jitter due to their internal functioning. Therefore, setups for distributed clocks should not integrate such devices into the topology.

4 The application interface

4.1 Addressing schemes used in EtherCAT Master

4.1.1 Auto-increment address

The auto-increment addressing is the addressing method on the EtherCAT bus associated with topology dependent addressing.

The following table shows the relation between the auto-increment address and the position in topology:

Auto-Increment Address	Position in topology
0x0000	First slave in topology (position 1)
0xFFFF	Second slave in topology
0xFFFE	Third slave in topology
0x0001-n	Slave in topology at position n

Table 7: Auto-increment address related to topology position

For accessing acyclic services during generic bus scan, the master uses a derived scheme called topology position which is defined as following wrap-around calculation:

```
uint16_t usTopologyPosition = 0x0001 – usAutoIncAddress;
```

For details about topology position, see section *Topology position* on page 19.

This addressing scheme is used in the following services:

- *Legacy ESC SII access (ECM V3.X API)* (page 178)
- *Legacy bus scan* (page 201)

4.1.2 Fixed station address

The fixed station address is used as long as a configuration is active in the master and bus scan is not active. The fixed station address is defined by configuration and is not topology dependent.

This addressing scheme is used in the following services:

- *Slave state* (page 36)
- *Diagnostic log* (page 47)
- *CoE services* (page 70)
- *SoE services* (page 120)
- *ESC register access* (page 170)
- *ESC SII access* (page 174)
- *Legacy ESC SII access (ECM V3.X API)* (page 178)

4.1.3 Topology position

The Topology position addressing scheme is used to simplify internal structure based on the supported station address numbering allowed.

Following table shows topology Position and position in topology:

Topology Position	Position in topology
0x0001	First slave in topology (position 1)
0x0002	Second slave in topology
0x0003	Third slave in topology
0x0000+n	Slave in topology at position n

Table 8: Topology position scheme related to topology position on bus

For calculating auto-increment address from the topology position the following wrap-around calculation is used:

```
uint16_t usAutoIncAddress = 0x0001 – usTopology Position;
```

For details about auto-increment Address, see section *Auto-increment address* on page 18.

This addressing scheme is used in the following services when generic bus scan is active:

- *CoE services* (page 70)
- *SoE services* (page 120)
- *ESC register access* (page 170)
- *ESC SII access* (page 174)
- *Legacy ESC SII access (ECM V3.X API)* (page 178)

  If fixed addressing using
- *Generic bus scan* (page 197)

4.1.4 Device index

The device index is solely derived from configuration load order. It does not resemble any kind of position-based addressing.

This addressing scheme is used in the following service: *Get slave connection information service* (page 164).

4.2 Distributed Clocks

The EtherCAT master always uses the first available DC supporting slave as DC reference clock.

The reasons for this are mainly:

- Lower jitter compared to PHY jitter
- Broadcast Read on DcSysTimeDiff register possible

4.3 Synchronization configuration

The synchronization configuration applies only to LFW/SHM. Its purpose is configuring the DPM handling accordingly. This allows applications to run synchronously to the bus cycle if needed.

4.3.1 Synchronization modes

4.3.1.1 Free-run

In Free-Run Mode, the application can exchange process data at any time. However, the handshake bit handling is not related to the bus cycle i.e. the toggle back by netX happens at any time. The application cannot determine the bus cycle reference from the handshake bits in this mode.

4.3.1.2 I/O sync mode 1

The I/O Sync Mode 1 allows determining the bus cycle reference from the input handshake bit. However, the input and output process data exchanges are not handled at the same time within the stack. Therefore, the application has to obey particular rules within this mode.

The inputs are only provided to the application every bus cycle when proper handshaking is done. If the application does not toggle the input data handshake to the netX, the master will not provide new inputs until the application has given the handshake to the netX. In the error case, the master is incrementing the input update error `PDInCnt`.

The output exchange is accepted at any time but the recommendation is to loosely couple the output exchange via the application cycle. If the application does not provide output data to be transmitted in time, the master will send the previously exchanged output process data.

The latency of I/O sync mode 1 is one cycle since when the application receive data from the bus and the time its output data is passed on the bus is one cycle later. The available time for the application processing data is slightly shorter than I/O sync mode 2.

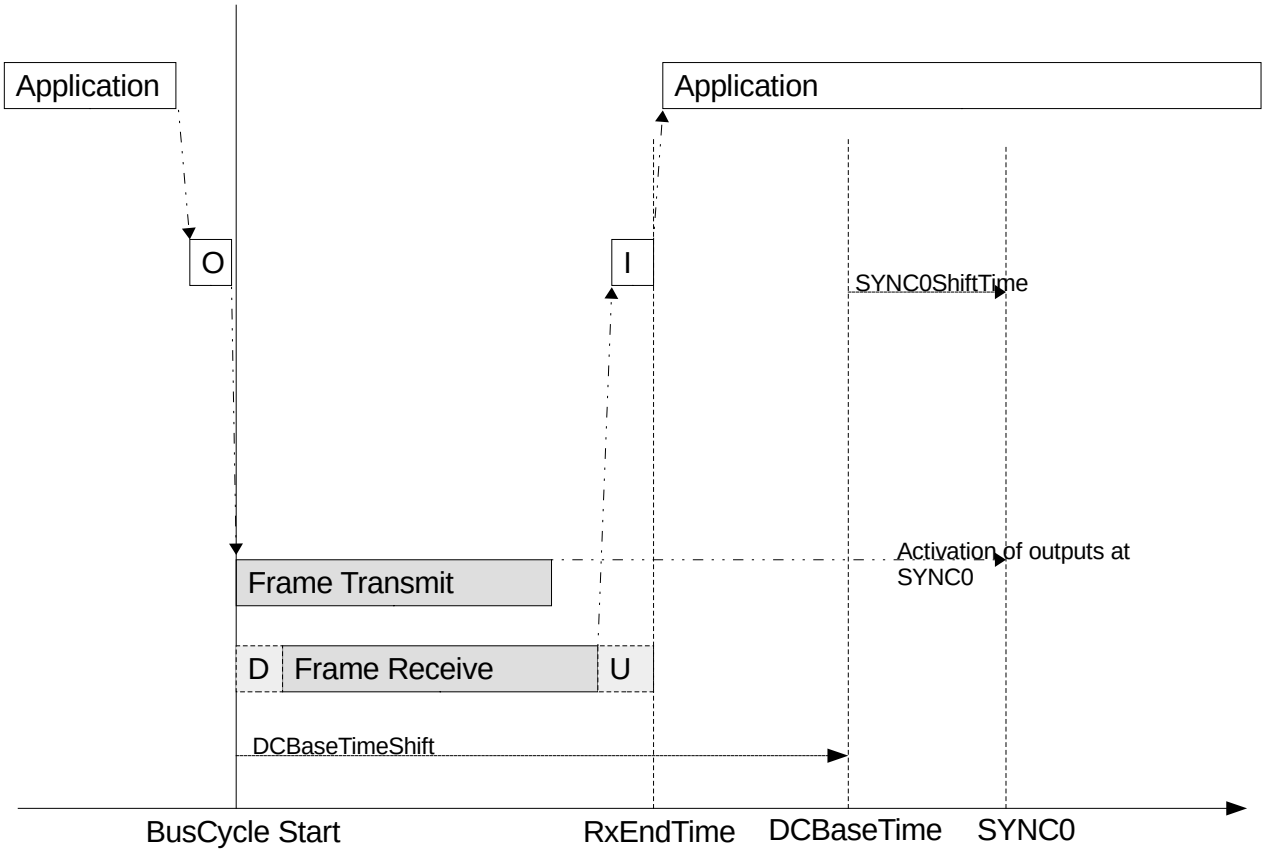


Figure 1: I/O sync mode 1 Timing Diagram

4.3.1.3 I/O sync mode 2

The I/O Sync Mode 2 allows determining the bus cycle reference from the input handshake bit. However, the input and output process data exchanges are not handled at the same time within the stack. Therefore, the **application** has to obey particular rules within this mode.

The inputs are only provided to the application every bus cycle when proper handshaking is done. If the application does not toggle the input data handshake to the netX, the master will not provide new inputs until the application has given the handshake to the netX. In case of an error, the master is incrementing the input update error `PDInCnt`.

The output exchange is accepted at any time but the recommendation is to loosely couple the output exchange via the application cycle. If the application does not provide output data to be transmitted in time, the master will send the previously exchanged output process data.

The latency of I/O sync mode 2 is two cycles since when the application receive data from the bus and the time its output data is passed on the bus is two cycles later.

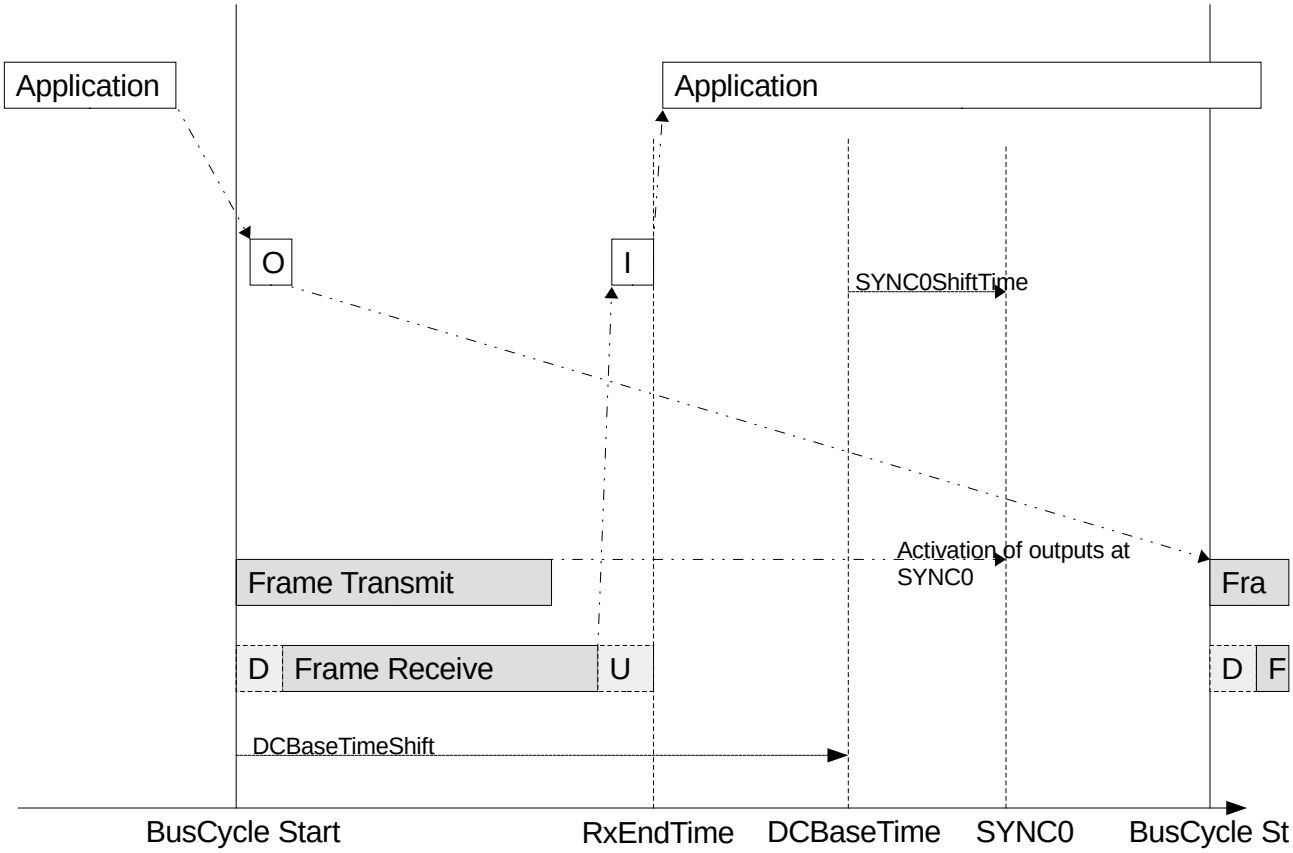


Figure 2: I/O sync mode 2 timing diagram

4.3.2 Packets

4.3.2.1 Set handshake configuration

This packet has to be used for reconfiguring the synchronization modes provided through DPM.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_SET_HANDSHAKE_CONFIG_REQ_DATA_Ttag
{
  /* Input process data handshake mode */
  uint8_t                          bPDInHskMode;
  /* Input process data trigger source. Currently unused, set to zero. */
  uint8_t                          bPDInSource;
  /* Threshold for input process data handshake handling errors */
  uint16_t                         usPDInErrorTh;

  /* Output process data handshake mode */
  uint8_t                          bPDOutHskMode;
  /* Output process data trigger source. Currently unused, set to zero. */
  uint8_t                          bPDOutSource;
  /* Threshold for output process data handshake handling errors */
  uint16_t                         usPDOutErrorTh;

  /* Synchronization handshake mode */
  uint8_t                          bSyncHskMode;
  /* Synchronization source */
  uint8_t                          bSyncSource;
  /* Threshold for synchronization handshake handling errors */
  uint16_t                         usSyncErrorTh;

  /* Reserved for future use. Set to zero. */
  uint32_t                         aulReserved[2];
} HIL_SET_HANDSHAKE_CONFIG_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_SET_HANDSHAKE_CONFIG_REQ_Ttag
{
  HIL_PACKET_HEADER_T                tHead;
  HIL_SET_HANDSHAKE_CONFIG_REQ_DATA_T tData;
} HIL_SET_HANDSHAKE_CONFIG_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	20	Packet Data Length in bytes
ulCmd	UINT32	0x2F34	HIL_SET_HANDSHAKE_CONFIG_REQ
Data			
bPDInHskMode	UINT8		Input Process Data Handshake Mode
bPDInSource	UINT8		Input Process Data Trigger Source
usPDInErrorTh	UINT16	0	Threshold for input process data handshake handling error.
bPDOutHskMode	UINT8		Output Process Data Handshake Mode
bPDOutSource	UINT8		Output Process Data Trigger Source
usPDOutErrorTh	UINT16	0	Threshold for output process data handshake handling error
bSyncHskMode	UINT8	0	Synchronization Handshake Mode
bSyncSource	UINT8	0	Synchronization Source
usSyncErrorTh	UINT16	0	Threshold for synchronization handshake handling error

Table 9: HIL_SET_HANDSHAKE_CONFIG_REQ – Set handshake configuration request

Available modes

bPDInHskMode	bPDInSource	bPDOutHskMode	bPDOutSource	Description
0/4	0	0/4	0	Free-Run
5	0	4	0	I/O Sync Mode 1
6	0	4	0	I/O Sync Mode 2

For mode descriptions, see section *Synchronization modes* (page 20).

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T  HIL_SET_HANDSHAKE_CONFIG_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section *Status/Error codes overview*
ulCmd	UINT32	0x2F35	HIL_SET_HANDSHAKE_CONFIG_CNF

Table 10: HIL_SET_HANDSHAKE_CONFIG_CNF – Set handshake configuration confirmation

4.4 State control

4.4.1 Architecture of master state control

The master implements a separation between target state setting and changing the current state. Therefore, the packets for setting the target state will only set the new state to reach. The confirmations return immediately. So, these do not indicate completion of the state change at all.

In order to determine completion and current network status, the following services have to be used:

- *Get current master state* (page 29), polling-based, application actively requests for current status
- *Status indications* (page 41), event-based, master signals indications to the application

Bus off and setting the target state

The new target state can be set when the master is set to Bus Off. The request will be returned with the error code `ECM_INFO_EMC_BUS_IS_OFF` (0x40CD0017). However, the action of the packet is still executed.

Error codes related to successful operation

- `SUCCESS_HIL_OK` (0x00000000)

  The master has accepted the new target phase and will proceed to it.

- `ECM_INFO_EMC_BUS_IS_OFF` (0x40CD0017)

  The master has accepted the new target state. However, Bus Off locks it to inactive state.

Indications and setting the target state

When the state indication indicates the same phase as the last issued Set Target State request, the phase change has been completed successfully.

When the state indication is received with `ulStopReason` unequal 0, the phase change has been aborted due to an error during phase change.

Note: The diagnostic log can provide additional detail on the reason. For details about how to use the diagnostic log, see section *Diagnostic log* (page 47).

4.4.2 Architecture of slave state control

When the master state control is not active, the slaves can be set to specific state e.g. BOOT.

The master implements a separation between target state setting and changing the current state. Therefore, the packets for setting the target state will only set the new state to reach. The confirmations return immediately. So, these do not indicate completion of the state change at all.

In order to determine completion and current slave status, the following services have to be used:

- Get current slave state (page 38), polling-based, application actively requests for current status

Bus off

During Bus off, no state setting of slaves is allowed.

Error codes related to successful operation

- `SUCCESS_HIL_OK` (0x00000000)

  The master has accepted the new target phase and will proceed to it.

Getting current state and setting the target state

When the Get Slave Current State confirmation indicates the same phase as the last issued Set Slave Target State request, the phase change has been completed successfully.

When the Get Slave Current State confirmation is received with `ulActiveError` unequal 0, the phase change has been aborted due to an error during phase change.

Note: The diagnostic log can provide additional detail on the reason. For details about how to use the diagnostic log, see section *Diagnostic log* (page 47).

4.4.3 Master state

4.4.3.1 Set master target state

The service is used for requesting a new master target state specified in `bTargetState`.

If the packet has been returned with `ulSta` equal to `SUCCESS_HIL_OK`, the master will change the bus status to the newly requested target state.

For details on how to determine the current network status, see section *Architecture of master state control* (page 25).

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_SET_MASTER_TARGET_STATE_REQ_DATA_Ttag
{
  uint8_t bTargetState;
} ECM_IF_SET_MASTER_TARGET_STATE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SET_MASTER_TARGET_STATE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SET_MASTER_TARGET_STATE_REQ_DATA_T tData;
} ECM_IF_SET_MASTER_TARGET_STATE_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	1	Packet Data Length in bytes
ulCmd	UINT32	0x9E00	`ECM_IF_CMD_SET_MASTER_TARGET_STATE_REQ`
Data			
bTargetState	UINT8	1, 2, 4, 8	Target state. See *Table 12: Possible values of `bTargetState`*

Table 11: `ECM_IF_CMD_SET_MASTER_TARGET_STATE_REQ` – Set master target state request

Defined values for bTargetState

Value	Definition / description
0x01	ECM_IF_STATE_INIT Master is requested to be in state INIT
0x02	ECM_IF_STATE_PREOP Master is requested to be in state PREOP
0x04	ECM_IF_STATE_SAFEOP Master is requested to be in state SAFEOP
0x08	ECM_IF_STATE_OP Master is requested to be in state OP

Table 12: Possible values of bTargetState

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_SET_MASTER_TARGET_STATE_CNF_DATA_Ttag
{
  uint8_t bTargetState;
} ECM_IF_SET_MASTER_TARGET_STATE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SET_MASTER_TARGET_STATE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SET_MASTER_TARGET_STATE_CNF_DATA_T tData;
} ECM_IF_SET_MASTER_TARGET_STATE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	1	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E01	ECM_IF_CMD_SET_MASTER_TARGET_STATE_CNF
Data			
bTargetState	UINT8	1, 2, 4, 8	Same value as specified in request for target state

Table 13: ECM_IF_CMD_SET_MASTER_TARGET_STATE_CNF – Set master target state confirmation

4.4.3.2 Get current master state

This service retrieves the current and target network status of the master.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_MASTER_CURRENT_STATE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_GET_MASTER_CURRENT_STATE_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E02	ECM_IF_CMD_GET_MASTER_CURRENT_STATE_REQ

Table 14: ECM_IF_CMD_GET_MASTER_CURRENT_STATE_REQ – Get master current state request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_MASTER_CURRENT_STATE_CNF_DATA_Ttag
{
  uint8_t bCurrentState;
  uint8_t bTargetState;
  uint32_t ulStopReason;
  uint32_t ulMasterStatusFlags;
} ECM_IF_GET_MASTER_CURRENT_STATE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_MASTER_CURRENT_STATE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_MASTER_CURRENT_STATE_CNF_DATA_T tData;
} ECM_IF_GET_MASTER_CURRENT_STATE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	10	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E03	ECM_IF_CMD_GET_MASTER_CURRENT_STATE_CNF
Data			
bCurrentState	UINT8	0,1,2,4,8, 0x18,0x1D,0x1E,0x1F	Current state of master See *Table 16: Possible values of bCurrentState*
bTargetState	UINT8	1,2,4,8	Target state of master See *Table 17: Possible values of bTargetState*
ulStopReason	UINT32	0 or valid reason code	If this value equals 0, the state change is either progressing or successful. If this value is unequal to 0, the last state change has been aborted.. See *Status/Error codes overview*
ulMasterStatusFlags	UINT32	0-7	Master status flags See *Table 18: Meaning of ulMasterFlags*

Table 15: ECM_IF_CMD_GET_MASTER_CURRENT_STATE_CNF – Get master current state confirmation

Defined values for bCurrentState

Value	Definition / description
0x00	ECM_IF_STATE_BUSOFF Master is in state Bus off
0x01	ECM_IF_STATE_INIT Master is in state INIT
0x02	ECM_IF_STATE_PREOP Master is in state PREOP
0x04	ECM_IF_STATE_SAFEOP Master is in state SAFEOP
0x08	ECM_IF_STATE_OP Master is in state OP
0x18	ECM_IF_STATE_LEAVE_OP Master is leaving OP state. This state is signaled when master begins processing a state change away from OP
0x1D	ECM_IF_STATE_BUSSCAN_COMPLETE_NO_PREOP Legacy bus scan completed
0x1E	ECM_IF_STATE_BUSSCAN Bus scan in progress
0x1F	ECM_IF_STATE_BUSSCAN_COMPLETE Bus scan is completed and PREOP is reached with all slaves.

Table 16: Possible values of bCurrentState

Defined values for bTargetState

Value	Definition / description
0x01	ECM_IF_STATE_INIT Master is requested to be in state INIT
0x02	ECM_IF_STATE_PREOP Master is requested to be in state PREOP
0x04	ECM_IF_STATE_SAFEOP Master is requested to be in state SAFEOP
0x08	ECM_IF_STATE_OP Master is requested to be in state OP

Table 17: Possible values of bTargetState

Bit mask for `ulMasterFlags`

Bit No.	Definition / description
31-3	RESERVED Reserved, set to 0.
2	MSK_ECM_IF_MASTER_STATUS_FLAGS_AT_LEAST_ONE_MANDATORY_SLAVE_NOT_IN_OP If this bit is set, at least one mandatory slave is not in OP when master is in OP. But, the slave is still connected.
1	MSK_ECM_IF_MASTER_STATUS_FLAGS_DC_XRMW_STOPPED If this bit is set, the DC handling stopped sending ARMW/FRMW telegrams. The DC Slaves are not synchronizing their sys time in that case.
0	MSK_ECM_IF_MASTER_STATUS_FLAGS_AT_LEAST_ONE_MANDATORY_SLAVE_LOST If this bit is set, at least one mandatory slave is not connected to master anymore.

Table 18: Meaning of `ulMasterFlags`

4.4.4 Master state (Legacy)

The legacy packets are available for applications which have been developed for ECM V3.X.

4.4.4.1 Set target state (Legacy)

This service is used for requesting a target state specified in usNewEcState.

If the packet has been returned with `ulSta` equal to `SUCCESS_HIL_OK`, the master will change the bus status to the newly requested target state.

For details on how to determine the current network status, see section *Architecture of master state control* (page 25).

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SET_ECSTATE_REQ_DATA_Ttag
{
  uint16_t usNewEcState; /* see defines ETHERCAT_MASTER_BUSSTATE_*, allowed values are
_INIT, _PREOP, _SAFEOP, _OP  */
} ETHERCAT_MASTER_PACKET_SET_ECSTATE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SET_ECSTATE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_SET_ECSTATE_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_SET_ECSTATE_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	2	Packet Data Length in bytes
ulCmd	UINT32	0x650048	`ETHERCAT_MASTER_CMD_SET_ECSTATE_REQ`
Data			
usNewEcState	UINT16	1, 2, 4, 8	Target state. See *Table 20: Possible values of `usNewEcState`*

Table 19: `ETHERCAT_MASTER_CMD_SET_ECSTATE_REQ` – Set master target state request (Legacy)

Defined values for `usNewEcState`

Value	Definition / description
0x01	`ECM_IF_STATE_INIT / ETHERCAT_MASTER_BUSSTATE_INIT` Master is requested to be in state INIT
0x02	`ECM_IF_STATE_PREOP / ETHERCAT_MASTER_BUSSTATE_PREOP` Master is requested to be in state PREOP
0x04	`ECM_IF_STATE_SAFEOP / ETHERCAT_MASTER_BUSSTATE_SAFEOP` Master is requested to be in state SAFEOP
0x08	`ECM_IF_STATE_OP / ETHERCAT_MASTER_BUSSTATE_OP` Master is requested to be in state OP

Table 20: Possible values of `usNewEcState`

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SET_ECSTATE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ETHERCAT_MASTER_PACKET_SET_ECSTATE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x650049	ETHERCAT_MASTER_CMD_SET_ECSTATE_CNF

Table 21: ETHERCAT_MASTER_CMD_SET_ECSTATE_CNF – Set master target state confirmation (Legacy)

4.4.4.2 Get Current State (Legacy)

This service retrieves the current network status of the master.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ECSTATE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ETHERCAT_MASTER_PACKET_GET_ECSTATE_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x650046	ETHERCAT_MASTER_CMD_GET_ECSTATE_REQ

Table 22: ETHERCAT_MASTER_CMD_GET_ECSTATE_REQ – Get master current state request (Legacy)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ECSTATE_CNF_DATA_Ttag
{
  uint16_t    usCurrentEcState;  /* see defines ETHERCAT_MASTER_BUSSTATE_*, following
values are reported:
  - ETHERCAT_MASTER_BUSSTATE_UNKNOWN: master not initialized
  - ETHERCAT_MASTER_BUSSTATE_INIT, ETHERCAT_MASTER_BUSSTATE_PREOP,
ETHERCAT_MASTER_BUSSTATE_SAFEOP, ETHERCAT_MASTER_BUSSTATE_OP  */
} ETHERCAT_MASTER_PACKET_GET_ECSTATE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ECSTATE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_ECSTATE_CNF_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_ECSTATE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	2	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x650047	ETHERCAT_MASTER_CMD_GET_ECSTATE_CNF
Data			
usCurrentEcState	UINT16	0, 1, 2, 4, 8	Current state of master. See *Table 24: Possible values of usCurrentEcState*

Table 23: ETHERCAT_MASTER_CMD_GET_ECSTATE_CNF – Get master current state confirmation (Legacy)

Defined values for `usCurrentEcState`

Value	Definition / description
0x0000	`ECM_IF_STATE_BUSOFF / ETHERCAT_MASTER_BUSSTATE_UNKNOWN` Master is in state Bus off
0x0001	`ECM_IF_STATE_INIT / ETHERCAT_MASTER_BUSSTATE_INIT` Master is in state INIT
0x0002	`ECM_IF_STATE_PREOP / ETHERCAT_MASTER_BUSSTATE_PREOP` Master is in state PREOP
0x0004	`ECM_IF_STATE_SAFEOP / ETHERCAT_MASTER_BUSSTATE_SAFEOP` Master is in state SAFEOP
0x0008	`ECM_IF_STATE_OP / ETHERCAT_MASTER_BUSSTATE_OP` Master is in state OP

Table 24: Possible values of `usCurrentEcState`

4.4.5 Slave state

4.4.5.1 Set slave target state

This service is used for requesting a slave target state specified in variable `bTargetState`.

If the packet has been returned with `ulSta` equal to `SUCCESS_HIL_OK`, the master will change the slave's status to the newly requested target state.

For details on how to determine the current slave status, see section *Architecture of slave state control* (page 26).

The following addressing schemes are used:

- *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_SET_SLAVE_TARGET_STATE_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint8_t bTargetState;
} ECM_IF_SET_SLAVE_TARGET_STATE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SET_SLAVE_TARGET_STATE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SET_SLAVE_TARGET_STATE_REQ_DATA_T tData;
} ECM_IF_SET_SLAVE_TARGET_STATE_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	3	Packet Data Length in bytes
ulCmd	UINT32	0x9E04	`ECM_IF_CMD_SET_SLAVE_TARGET_STATE_REQ`
Data			
usStationAddress	UINT16		See section *Fixed station address* (page 18)
bTargetState	UINT8	1,2,3,4,8	Target state. See *Table 26: Possible values of `bTargetState`*

Table 25: `ECM_IF_CMD_SET_SLAVE_TARGET_STATE_REQ` – Set slave target state request

Defined values for `bTargetState`

Value	Definition / description
0x01	`ECM_IF_STATE_INIT` Slave is requested to be in state INIT
0x02	`ECM_IF_STATE_PREOP` Slave is requested to be in state PREOP
0x03	`ECM_IF_STATE_BOOT` Slave is requested to be in state BOOT
0x04	`ECM_IF_STATE_SAFEOP` Slave is requested to be in state SAFEOP
0x08	`ECM_IF_STATE_OP` Slave is requested to be in state OP

Table 26: Possible values of `bTargetState`

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_SET_SLAVE_TARGET_STATE_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint8_t bTargetState;
} ECM_IF_SET_SLAVE_TARGET_STATE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SET_SLAVE_TARGET_STATE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SET_SLAVE_TARGET_STATE_CNF_DATA_T tData;
} ECM_IF_SET_SLAVE_TARGET_STATE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	3	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E05	ECM_IF_CMD_SET_SLAVE_TARGET_STATE_CNF
Data			
usStationAddress	UINT16	Valid address	Same value as specified in request for station address
bTargetState	UINT8	1,2,3,4,8	Same value as specified in request for target state

Table 27: ECM_IF_CMD_SET_SLAVE_TARGET_STATE_CNF – Set slave target state confirmation

4.4.5.2 Get current slave state

This service retrieves the current and target network status of a specified slave.

The following addressing schemes are used:

- 4.1.2 Fixed station address

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_SLAVE_CURRENT_STATE_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
} ECM_IF_GET_SLAVE_CURRENT_STATE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_SLAVE_CURRENT_STATE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_SLAVE_CURRENT_STATE_REQ_DATA_T tData;
} ECM_IF_GET_SLAVE_CURRENT_STATE_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	2	Packet Data Length in bytes
ulCmd	UINT32	0x9E06	ECM_IF_CMD_GET_SLAVE_CURRENT_STATE_REQ
Data			
usStationAddress	UINT16		See section *Fixed station address* (page 18).

Table 28: ECM_IF_CMD_GET_SLAVE_CURRENT_STATE_REQ – Get slave current state request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_SLAVE_CURRENT_STATE_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint8_t bCurrentState;
  uint8_t bTargetState;
  uint32_t ulActiveError;
} ECM_IF_GET_SLAVE_CURRENT_STATE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_SLAVE_CURRENT_STATE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_SLAVE_CURRENT_STATE_CNF_DATA_T tData;
} ECM_IF_GET_SLAVE_CURRENT_STATE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	8	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E07	ECM_IF_CMD_GET_SLAVE_CURRENT_STATE_CNF
Data			
usStationAddress	UINT16		Same value as given in request
bCurrentState	UINT8	0, 1, 2, 3, 4, 8, 0x11, 0x12, 0x13, 0x14	Current state of slave. See *Table 30: Possible values of bCurrentState*
bTargetState	UINT8	1, 2, 3, 4, 8	Target state of slave See *Table 31: Possible values of bTargetState*
ulActiveError	UINT32		If it equals 0, the state change is progressing or successful. If it is unequal 0, the last state change stopped. See Status/Error codes overview

Table 29: ECM_IF_CMD_GET_SLAVE_CURRENT_STATE_CNF – Get slave current state confirmation

Defined values for bCurrentState

Value	Definition / description
0x00	ECM_IF_STATE_BUSOFF Slave is in state Bus off
0x01	ECM_IF_STATE_INIT Slave is in state INIT
0x02	ECM_IF_STATE_PREOP Slave is in state PREOP
0x03	ECM_IF_STATE_BOOT Slave is in state BOOT
0x04	ECM_IF_STATE_SAFEOP Slave is in state SAFEOP
0x08	ECM_IF_STATE_OP Slave is in state OP
0x11	ECM_IF_STATE_INIT_ERR Slave is in state INIT+ERR
0x12	ECM_IF_STATE_PREOP_ERR Slave is in state PREOP+ERR
0x13	ECM_IF_STATE_BOOT_ERR Slave is in state BOOT+ERR
0x14	ECM_IF_STATE_SAFEOP_ERR Slave is in state SAFEOP+ERR

Table 30: Possible values of bCurrentState

Defined values for bTargetState

Value	Definition / description
0x01	ECM_IF_STATE_INIT Slave is requested to be in state INIT
0x02	ECM_IF_STATE_PREOP Slave is requested to be in state PREOP
0x03	ECM_IF_STATE_BOOT Slave is requested to be in state BOOT
0x04	ECM_IF_STATE_SAFEOP Slave is requested to be in state SAFEOP
0x08	ECM_IF_STATE_OP Slave is requested to be in state OP

Table 31: Possible values of bTargetState

4.5 Status indications

4.5.1 Registration and deregistration of status indications

4.5.1.1 Register for status indications service

This packet registers an application task for receiving status indications.

The following groups of status indications will be sent to the application task after successful registration:

- *Available indications* (page 43)

If the application does not want to receive those indications anymore, it has to use the service described in section *Unregister from status indications service* (page 42).

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T          HIL_REGISTER_APP_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x2F10	HIL_REGISTER_APP_REQ

Table 32: HIL_REGISTER_APP_REQ – Register for status indications request

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T          HIL_REGISTER_APP_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x2F11	HIL_REGISTER_APP_CNF

Table 33: HIL_REGISTER_APP_CNF – Register for status indications confirmation

4.5.1.2 Unregister from status indications service

This packet deregisters an application task from receiving status indications.

The following status indications will not continue to be sent to the application task anymore after successful deregistration:

- 4.5.2 Available indications

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T          HIL_UNREGISTER_APP_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x2F12	HIL_UNREGISTER_APP_REQ

Table 34: HIL_UNREGISTER_APP_REQ –Unregister from status indications request

Packet structure reference

```
typedef HIL_EMPTY_PACKET_T          HIL_UNREGISTER_APP_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x2F13	HIL_UNREGISTER_APP_CNF

Table 35: HIL_UNREGISTER_APP_CNF – Unregister from status indications confirmation

4.5.2 Available indications

4.5.2.1 Master state indication

This packet indicates the new state of the master. For details on how this indication relates to master state control, see section *Architecture of master state control* (page 25).

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_MASTER_CURRENT_STATE_IND_DATA_Ttag
{
  uint8_t bCurrentState;
  uint8_t bTargetState;
  uint32_t ulStopReason;
  uint32_t ulMasterStatusFlags;
} ECM_IF_MASTER_CURRENT_STATE_IND_DATA_T;

typedef ECM_IF_MASTER_CURRENT_STATE_IND_DATA_T
ECM_IF_GET_MASTER_CURRENT_STATE_IND_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_MASTER_CURRENT_STATE_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_MASTER_CURRENT_STATE_IND_DATA_T tData;
} ECM_IF_MASTER_CURRENT_STATE_IND_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	10	Packet Data Length in bytes
ulCmd	UINT32	0x9E10	`ECM_IF_CMD_GET_MASTER_CURRENT_STATE_IND`
Data			
bCurrentState	UINT8	0,1,2,4,8, 0x18,0x1D,0x1E,0x1F	Current state of master. See *Table 37: Possible values of `bCurrentState`*
bTargetState	UINT8	1,2,4,8	Target state of master See *Table 38: Possible values of `bTargetState`*
ulStopReason	UINT32		If it equals 0, the state change is progressing or successful. If it is unequal 0, the last state change stopped. See Status/Error codes overview
ulMasterStatusFlags	UINT32	0-7	Master status flags See *Table 18: Meaning of `ulMasterFlags`*

Table 36: `ECM_IF_MASTER_CURRENT_STATE_IND_T` – Master state indication

Defined values for bCurrentState

Value	Definition / description
0x00	ECM_IF_STATE_BUSOFF Master is in state Bus off
0x01	ECM_IF_STATE_INIT Master is in state INIT
0x02	ECM_IF_STATE_PREOP Master is in state PREOP
0x04	ECM_IF_STATE_SAFEOP Master is in state SAFEOP
0x08	ECM_IF_STATE_OP Master is in state OP
0x18	ECM_IF_STATE_LEAVE_OP Master is leaving OP state This state is signaled when master begins processing a state change away from OP
0x1D	ECM_IF_STATE_BUSSCAN_COMPLETE_NO_PREOP Legacy bus scan completed
0x1E	ECM_IF_STATE_BUSSCAN Bus scan in progress
0x1F	ECM_IF_STATE_BUSSCAN_COMPLETE Bus scan is completed and PREOP is reached with all slaves.

Table 37: Possible values of bCurrentState

Defined values for bTargetState

Value	Definition / description
0x01	ECM_IF_STATE_INIT Master is requested to be in state INIT
0x02	ECM_IF_STATE_PREOP Master is requested to be in state PREOP
0x04	ECM_IF_STATE_SAFEOP Master is requested to be in state SAFEOP
0x08	ECM_IF_STATE_OP Master is requested to be in state OP

Table 38: Possible values of bTargetState

4.5.2.2 Slave current state indication

This packet indicates the current state of all slaves.

The following addressing scheme is used: section *Fixed station address* (page 18).

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_SLAVE_CURRENT_STATE_IND_DATA_ENTRY_Ttag
{
  uint16_t usStationAddress;
  uint16_t usCurrentStatus;
  uint32_t ulLastError;
} ECM_IF_SLAVE_CURRENT_STATE_IND_DATA_ENTRY_T;

#define ECM_IF_SLAVE_CURRENT_STATE_IND_MAX_ENTRIES (HIL_MAX_DATA_SIZE /
sizeof(ECM_IF_SLAVE_CURRENT_STATE_IND_DATA_ENTRY_T))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_SLAVE_CURRENT_STATE_IND_DATA_Ttag
{
  ECM_IF_SLAVE_CURRENT_STATE_IND_DATA_ENTRY_T
atEntries[ECM_IF_SLAVE_CURRENT_STATE_IND_MAX_ENTRIES];
} ECM_IF_SLAVE_CURRENT_STATE_IND_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SLAVE_CURRENT_STATE_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SLAVE_CURRENT_STATE_IND_DATA_T tData;
} ECM_IF_SLAVE_CURRENT_STATE_IND_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	8 * N	Packet Data Length in bytes N = number of slaves N_{max} = ECM_IF_SLAVE_CURRENT_STATE_IND_MAX_ENTRIES
ulCmd	UINT32	0x9E12	ECM_IF_CMD_SLAVE_CURRENT_STATE_IND
Data			
atEntries[]. usStationAddress	UINT16		See section *Fixed station address* (page 18)
atEntries []. usCurrentStatus	UINT16	0, 1, 2, 3, 4, 8, 0x11, 0x12, 0x13, 0x14	Current state of slave, see Table 40
atEntries[]. ulLastError	UINT32		See section Status/Error codes overview

Table 39: ECM_IF_SLAVE_CURRENT_STATE_IND – Current slave state indication

Defined values for usCurrentState

Value	Definition / description
0x00	ECM_IF_STATE_BUSOFF Slave is in state Bus off
0x01	ECM_IF_STATE_INIT Slave is in state INIT
0x02	ECM_IF_STATE_PREOP Slave is in state PREOP
0x03	ECM_IF_STATE_BOOT Slave is in state BOOT
0x04	ECM_IF_STATE_SAFEOP Slave is in state SAFEOP
0x08	ECM_IF_STATE_OP Slave is in state OP
0x11	ECM_IF_STATE_INIT_ERR Slave is in state INIT+ERR
0x12	ECM_IF_STATE_PREOP_ERR Slave is in state PREOP+ERR
0x13	ECM_IF_STATE_BOOT_ERR Slave is in state BOOT+ERR
0x14	ECM_IF_STATE_SAFEOP_ERR Slave is in state SAFEOP+ERR

Table 40: usCurrentState values

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SLAVE_CURRENT_STATE_RES_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_SLAVE_CURRENT_STATE_RES_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32	0	
ulCmd	UINT32	0x9E13	ECM_IF_CMD_SLAVE_CURRENT_STATE_RES

Table 41: ECM_IF_SLAVE_CURRENT_STATE_RES – Response to master state indication

4.6 Diagnostic log

The diagnostic log provides a single entry point to EtherCAT-specific diagnostic information from the master. This includes several data points e.g. topology state, slave state and so on.

4.6.1 Entry format of diagnostic log

4.6.1.1 General format

The general format is based on a header and a union to provide a common format for all possible entries. The basic layout is provided here:

Diagnostic log entry structure

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_Ttag
{
  ECM_DIAG_ENTRY_HEADER_T tHead;
  ECM_DIAG_ENTRY_DATA_T tData;
} ECM_DIAG_ENTRY_T;
```

Variable name	Type	Meaning
tHead	ECM_DIAG_ENTRY_HEADER_T	Entry header see section *Entry header* (page 48) for details
tData	ECM_DIAG_ENTRY_DATA_T	Entry data see section *Entry data format* (page 50) for details

Table 42: Structure ECM_DIAG_ENTRY_T

4.6.1.2 Entry header

The entry header is used for all diagnostic log entries. Its main purpose is to designate the type and the time stamp of the entry.

ECM_DIAG_ENTRY_HEADER_T Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_HEADER_Ttag
{
  uint16_t usEntryType;
  uint64_t ullTimestampNs;
} ECM_DIAG_ENTRY_HEADER_T;
```

Variable name	Type	Meaning
usEntryType	UINT16	Entry type
ulTimestampNs	UINT64	Timestamp (Nanoseconds, see above)

Table 43: Structure ECM_DIAG_ENTRY_HEADER_T

Defined values for usEntryType

Value	Definition / description
0x0001	VAL_ECM_DIAG_ENTRY_TYPE_NEW_STATE New master state see section *Entry: New communication state* (page 50) for data format
0x0002	VAL_ECM_DIAG_ENTRY_TYPE_BUS_ON Bus on see section *Entry: BusOn* (page 50) for data format
0x0003	VAL_ECM_DIAG_ENTRY_TYPE_BUS_OFF Bus off see section *Entry: BusOff* (page 51) for data format
0x0004	VAL_ECM_DIAG_ENTRY_TYPE_CHANNEL_INIT A channel Init occurred see section *Entry: ChannelInit* (page 51) for data format
0x0005	VAL_ECM_DIAG_ENTRY_TYPE_DPM_WATCHDOG A DPM watchdog error occurred see section *Entry: DPM watchdog* (page 51) for data format
0x0007	VAL_ECM_DIAG_ENTRY_TYPE_INTERNAL_ERROR Internal error See section *Entry: Internal error* (page 51) for data format
0x0008	VAL_ECM_DIAG_ENTRY_TYPE_ALL_SLAVES_LOST All slaves were disconnected. see section *Entry: All slaves lost* (page 52) for data format
0x0009	VAL_ECM_DIAG_ENTRY_TYPE_BUS_SCAN_REQUESTED A bus scan has been requested. see section *Entry: Bus scan requested* (page 52) for data format
0x000A	VAL_ECM_DIAG_ENTRY_TYPE_IDENTITY_MISMATCH Identity data mismatch for a slave detected. see section *Entry: Identity mismatch* (page 52) for data format

Value	Definition / description
0x000B	VAL_ECM_DIAG_ENTRY_TYPE_COE_INITCMD_FAILED CoE InitCmd Failed see section *Entry: CoE InitCmd failed* (page 53) for data format
0x000C	VAL_ECM_DIAG_ENTRY_TYPE_SOE_INITCMD_FAILED SoE InitCmd Failed see section *Entry: SoE InitCmd failed* (page 54) for data format
0x000F	VAL_ECM_DIAG_ENTRY_TYPE_REG_INITCMD_WARNING Register InitCmd Warning see section *Entry: Reg InitCmd warning* (page 55) for data format
0x0010	VAL_ECM_DIAG_ENTRY_TYPE_REG_INITCMD_FAILED Register InitCmd Failed see section *Entry: Reg InitCmd failed* (page 56) for data format
0x0011	VAL_ECM_DIAG_ENTRY_TYPE_ALCONTROL_FAILED ALControl Request Failed see section *Entry: ALControl failed* (page 57) for data format
0x0012	VAL_ECM_DIAG_ENTRY_TYPE_SII_ASSIGN_TO_ECAT_FAILED SII Assign To ECAT Failed see section *Entry: SII Assign to EtherCAT failed* (page 58) for data format
0x0013	VAL_ECM_DIAG_ENTRY_TYPE_SII_ASSIGN_TO_PDI_FAILED SII Assign To PDI Failed see section Entry: SII Assign to PDI failed (page 58) for data format
0x0014	VAL_ECM_DIAG_ENTRY_TYPE_SII_READ_REQUEST_FAILED SII Read Request Failed see section Entry: SII Read request failed (page 59) for data format
0x0015	VAL_ECM_DIAG_ENTRY_TYPE_SLAVE_WANING Slave Warning see section *Entry: Slave warning* (page 60) for data format
0x0016	VAL_ECM_DIAG_ENTRY_TYPE_SLAVE_ERROR Slave Error see section *Entry: Slave error* (page 61) for data format

Table 44: Possible values of `usEntryType` for diagnostic log

`ullTimestampNs`

`ullTimestampNs` forms a large 64-bit nanosecond timestamp.

`ullTimestampNs` ranges from 0 ... 0xFFFFFFFFFFFFFF and is displayed in nanoseconds.

4.6.1.3 Entry data format

The union referenced via `tData` contains all possible formats. The currently used format depends on the entry type (`usEntryType`) set in the header.

ECM_DIAG_ENTRY_DATA_T Structure Reference

```
typedef __HIL_PACKED_PRE union __HIL_PACKED_POST ECM_DIAG_ENTRY_DATA_Ttag
{
  ECM_DIAG_ENTRY_NEW_STATE_T tNewState;
  ECM_DIAG_ENTRY_INTERNAL_ERROR_T tInternalError;
  ECM_DIAG_ENTRY_IDENTITY_MISMATCH_T tIdentityMismatch;
  ECM_DIAG_ENTRY_COE_INITCMD_FAILED_T tCoEInitCmdFailed;
  ECM_DIAG_ENTRY_SOE_INITCMD_FAILED_T tSoEInitCmdFailed;
  ECM_DIAG_ENTRY_VOE_INITCMD_FAILED_T tVoEInitCmdFailed;
  ECM_DIAG_ENTRY_REG_INITCMD_INFO_T tRegInitCmdInfo;
  ECM_DIAG_ENTRY_ALCONTROL_FAILED_T tAlControlFailed;
  ECM_DIAG_ENTRY_SII_ASSIGN_FAILED_T tSiiAssignFailed;
  ECM_DIAG_ENTRY_SII_REQUEST_FAILED_T tSiiRequestFailed;
  ECM_DIAG_ENTRY_SLAVE_WARNING_T tSlaveWarning;
  ECM_DIAG_ENTRY_SLAVE_ERROR_T tSlaveError;
} ECM_DIAG_ENTRY_DATA_T;
```

The elements of the union will be detailed in following chapters.

Note: Sections may repeat certain structure references for clarity.

4.6.1.4 Entry: New communication state

This entry designates what master state has been reached.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_NEW_STATE_Ttag
{
  uint8_t bState;
} ECM_DIAG_ENTRY_NEW_STATE_T;
```

Variable name	Type	Meaning
bState	UINT8	New master state

Table 45: Structure `ECM_DIAG_ENTRY_NEW_STATE_T`

Coding of `bState`

For a coding of `bState`, see Table 16 on page 30.

4.6.1.5 Entry: BusOn

This entry is recorded when a BusOn has been requested.

There is no additional data contained within the entry data.

4.6.1.6 Entry: BusOff

This entry is recorded when a BusOff has been requested.

There is no additional data contained within the entry data.

4.6.1.7 Entry: ChannelInit

This entry is recorded when a ChannelInit has been requested.

There is no additional data contained within the entry data.

4.6.1.8 Entry: DPM watchdog

This entry is recorded when a DPM watchdog timeout event has occurred.

There is no additional data contained within the entry data.

4.6.1.9 Entry: Internal error

This entry is recorded whenever the master detects an internal error. In case of problems, it is helpful to provide the data of the entry and a description of the problem to our support.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_INTERNAL_ERROR_Ttag
{
  uint32_t ulFunctionId;
  uint32_t ulErrorCode;
} ECM_DIAG_ENTRY_INTERNAL_ERROR_T;
```

Entry structure description

Variable name	Type	Meaning
ulFunctionId	UINT32	Function ID
ulErrorCode	UINT32	Error Code

Table 46: Structure ECM_DIAG_ENTRY_INTERNAL_ERROR_T

4.6.1.10 Entry: All slaves lost

This entry is recorded when no slave is left to communicate with. In that case, the master will autonomously fallback to INIT and retry to startup the bus again.

There is no additional data contained within the entry data.

4.6.1.11 Entry: Bus scan requested

This entry is recorded when the application requested a bus scan.

There is no additional data contained within the entry data.

4.6.1.12 Entry: Identity mismatch

This entry is recorded when a slave is not matching the expected slave in terms of identity data.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_IDENTITY_MISMATCH_Ttag
{
  uint16_t usTopologyPosition;
  uint16_t usCompareFlags;
  uint32_t ulExpectedVendorId;
  uint32_t ulExpectedProductCode;
  uint32_t ulExpectedRevisionNo;
  uint32_t ulExpectedSerialNo;
  uint32_t ulFoundVendorId;
  uint32_t ulFoundProductCode;
  uint32_t ulFoundRevisionNo;
  uint32_t ulFoundSerialNo;
} ECM_DIAG_ENTRY_IDENTITY_MISMATCH_T;
```

Entry structure description

Variable name	Type	Meaning
usTopologyPosition	UINT16	1 to n For details, see section *Topology position* (page 19)
usCompareFlags	UINT16	See table below
ulExpectedVendorId	UINT32	Configured Vendor Id at topology position
ulExpectedProductCode	UINT32	Configured Product Code at topology position
ulExpectedRevisionNo	UINT32	Configured Revision Number at topology position
ulExpectedSerialNo	UINT32	Configured Serial Number at topology position
ulFoundVendorId	UINT32	Actual detected Vendor Id at topology position
ulFoundProductCode	UINT32	Actual detected Product Code at topology position
ulFoundRevisionNo	UINT32	Actual detected Revision Number at topology position
ulFoundSerialNo	UINT32	Actual detected Serial Number at topology position

Table 47: Structure `ECM_DIAG_ENTRY_IDENTITY_MISMATCH_T`

Meaning of usCompareFlags

Bit No.	Definition / description
31-4	RESERVED Reserved, set to 0.
3	MSK_ECM_DIAG_ENTRY_IDENTITY_MISMATCH_COMPARE_SERIAL_NO If this bit is set, the Serial Number comparison failed if enabled
2	MSK_ECM_DIAG_ENTRY_IDENTITY_MISMATCH_COMPARE_REVISION_NO If this bit is set, the Revision Number comparison failed if enabled
1	MSK_ECM_DIAG_ENTRY_IDENTITY_MISMATCH_COMPARE_PRODUCT_CODE If this bit is set, the Product Code comparison failed if enabled
0	MSK_ECM_DIAG_ENTRY_IDENTITY_MISMATCH_COMPARE_VENDOR_ID If this bit is set, the VendorId comparison failed if enabled

Table 48: Meaning of usCompareFlags

4.6.1.13 Entry: CoE InitCmd failed

This entry is recorded when a CoE InitCmd for a slave fails.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_COE_INITCMD_FAILED_Ttag
{
  uint16_t usStationAddress;
  uint16_t usIndex;
  uint8_t bSubIndex;
  uint8_t bAction;
  uint16_t fIsCompleteAccess;
  uint32_t ulResult;
} ECM_DIAG_ENTRY_COE_INITCMD_FAILED_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
usIndex	UINT16	Index of SDO access
bSubIndex	UINT16	Subindex of SDO access
bAction	UINT8	
fIsCompleteAccess	UINT16	!= 0 when access is a complete access == 0 when access is a single subindex access
ulResult	UINT32	Error Result

Table 49: Structure ECM_DIAG_ENTRY_COE_INITCMD_FAILED_T

4.6.1.14 Entry: SoE InitCmd failed

This entry is recorded when a SoE InitCmd for a slave fails.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_SOE_INITCMD_FAILED_Ttag
{
  uint16_t usStationAddress;
  uint8_t bDriveNo;
  uint16_t usIDN;
  uint8_t bElements;
  uint8_t bAction;
  uint32_t ulResult;
} ECM_DIAG_ENTRY_SOE_INITCMD_FAILED_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
bDriveNo	UINT8	Drive number of SoE access
usIDN	UINT16	IDN number of SoE access
bElements	UINT8	Element flags of SoE access
bAction	UINT8	
ulResult	UINT32	Error Result

Table 50: Structure ECM_DIAG_ENTRY_SOE_INITCMD_FAILED_T

4.6.1.15 Entry: Reg InitCmd warning

This entry is recorded when the master detects register access faults on slaves that are not considered severe but yet worth being noted.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_REG_INITCMD_INFO_Ttag
{
  uint16_t usStationAddress;
  uint8_t bCmd;
  uint16_t usAdo;
  uint16_t usLength;
  uint32_t ulResult;
} ECM_DIAG_ENTRY_REG_INITCMD_INFO_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
bCmd	UINT8	EtherCAT Telegram Command
usAdo	UINT16	Physical address to be accessed with EtherCAT telegram
usLength	UINT16	Length to be accessed with EtherCAT telegram
ulResult	UINT32	Error Result

Table 51: Structure ECM_DIAG_ENTRY_REG_INITCMD_INFO_T

4.6.1.16 Entry: Reg InitCmd failed

This entry is recorded when the master detects register access faults on slaves that are considered severe and result into a stop of continuing the state switching.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_REG_INITCMD_INFO_Ttag
{
  uint16_t usStationAddress;
  uint8_t bCmd;
  uint16_t usAdo;
  uint16_t usLength;
  uint32_t ulResult;
} ECM_DIAG_ENTRY_REG_INITCMD_INFO_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
bCmd	UINT8	EtherCAT Telegram Command
usAdo	UINT16	Physical address to be accessed with EtherCAT telegram
usLength	UINT16	Length to be accessed with EtherCAT telegram
ulResult	UINT32	Error Result

Table 52: Structure ECM_DIAG_ENTRY_REG_INITCMD_INFO_T

4.6.1.17 Entry: ALControl failed

This entry is recorded when the master detects that a requested ESM status in a device could not be reached.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_ALCONTROL_FAILED_Ttag
{
  uint16_t usStationAddress;
  uint8_t bTargetState;
  uint16_t usAlStatusCode;
  uint32_t ulResult;
} ECM_DIAG_ENTRY_ALCONTROL_FAILED_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
bTargetState	UINT8	Target state that was tried to be reached
usAlStatusCode	UINT16	AlStatusCode from the slave that failed
ulResult	UINT32	Error Result

Table 53: Structure `ECM_DIAG_ENTRY_ALCONTROL_FAILED_T`

4.6.1.18 Entry: SII Assign to EtherCAT failed

This entry is recorded when the master detects that assigning the SII to EtherCAT bus did not complete.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_SII_ASSIGN_FAILED_Ttag
{
  uint16_t usStationAddress;
  uint32_t ulResult;
} ECM_DIAG_ENTRY_SII_ASSIGN_FAILED_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
ulResult	UINT32	Error Result

Table 54: Structure ECM_DIAG_ENTRY_SII_ASSIGN_FAILED_T

4.6.1.19 Entry: SII Assign to PDI failed

This entry is recorded when the master detects that assigning the SII to PDI did not complete.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_SII_ASSIGN_FAILED_Ttag
{
  uint16_t usStationAddress;
  uint32_t ulResult;
} ECM_DIAG_ENTRY_SII_ASSIGN_FAILED_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
ulResult	UINT32	Error Result

Table 55: Structure ECM_DIAG_ENTRY_SII_ASSIGN_FAILED_T

4.6.1.20 Entry: SII Read request failed

This entry is recorded when the master detects when reading an offset within SII failed.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_SII_REQUEST_FAILED_Ttag
{
  uint16_t usStationAddress;
  uint32_t ulSiiWordOffset;
  uint32_t ulResult;
} ECM_DIAG_ENTRY_SII_REQUEST_FAILED_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
ulSiiWordOffset	UINT16	Word offset in SII that could not be read. 0 => first word (at byte offset 0) 1 => second word (at byte offset 2)
ulResult	UINT32	Error Result

Table 56: Structure ECM_DIAG_ENTRY_SII_REQUEST_FAILED_T

4.6.1.21 Entry: Slave warning

This entry is recorded when the master detects certain faults on slaves that are not considered severe but yet worth being noted.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_SLAVE_WARNING_Ttag
{
  uint16_t usStationAddress;
  uint32_t ulWarningType;
  uint32_t ulWarningParam; /* value depends on ulWarningType */
} ECM_DIAG_ENTRY_SLAVE_WARNING_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
ulWarningType	UINT32	See table below
ulWarningParam	UINT32	Meaning depends on ulWarningType

Table 57: Structure `ECM_DIAG_ENTRY_SLAVE_WARNING_T`

Definition of codes for `ulWarningType`

Value	Definition / description
0x0001	`ECM_DIAG_ENTRY_SLAVE_WARNING_TYPE_ADVERTISED_64BIT_DC_NOT_WORKING` The slave has the 64bit DC supported flag set in Feature Register but the actual registers only support 32 bit DC.
0x0002	`ECM_DIAG_ENTRY_SLAVE_WARNING_TYPE_ADVERTISED_DC_NOT_WORKING` The slave has the DC supported flag set in Feature Register but the DC registers do not work at all.
0x0003	`ECM_DIAG_ENTRY_SLAVE_WARNING_TYPE_SLAVE_DID_NOT_ACCEPT_EOE_SET_IP_PARAMS` Slave did not accept EoE SetIpParams request. *`ulWarningParam`* contains the associated error code

Table 58: Available reason codes for `ulWarningType`

4.6.1.22 Entry: Slave error

This entry contains information about errors happening when the master is processing actions on a particular slave.

Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_DIAG_ENTRY_SLAVE_ERROR_Ttag
{
  uint16_t usStationAddress;
  uint32_t ulErrorType;
  uint32_t ulErrorParam; /* value depends on ulErrorType */
} ECM_DIAG_ENTRY_SLAVE_ERROR_T;
```

Entry structure description

Variable name	Type	Meaning
usStationAddress	UINT16	Fixed station address For details, see section *Fixed station address* (page 18)
ulErrorType	UINT32	See table below
ulErrorParam	UINT32	Meaning depends on ulErrorType

Table 59: Structure `ECM_DIAG_ENTRY_SLAVE_ERROR_T`

Definition of codes for `ulErrorType`

Value	Definition / description
0x0001	`ECM_DIAG_ENTRY_SLAVE_ERROR_TYPE_SYNC_NOT_POSSIBLE_WITHOUT_WORKING_DC` The slave was parameterized to have DC Sync configuration. Yet, the slave does not support the required DC for that.

Table 60: Available reason codes for `ulErrorType`

4.6.2 Reading and clearing diagnostic log entries

4.6.2.1 Read diagnostic log entry service

This packet reads the oldest available entry from the diagnostic log. That entry will be removed from the log. A subsequent read request will read the next entry which has now become the oldest entry.

If there are no entries, the request will return with an error.

For event-based handling, the diagnostic log supports indications. For a description, see section *Diagnostic log indication handling* (page 65).

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_DIAG_LOG_ENTRY_REQ_Ttag
{
  HIL_PACKET_HEADER_T                                tHead;
} ECM_IF_READ_DIAG_LOG_ENTRY_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E30	ECM_IF_CMD_READ_DIAG_LOG_ENTRY_REQ

Table 61: ECM_IF_CMD_READ_DIAG_LOG_ENTRY_REQ – Read diagnostic log entry service

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_READ_DIAG_LOG_ENTRY_CNF_DATA_Ttag
{
  uint32_t ulLostEntries;
  ECM_DIAG_ENTRY_T tDiagEntry;
} ECM_IF_READ_DIAG_LOG_ENTRY_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_DIAG_LOG_ENTRY_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_READ_DIAG_LOG_ENTRY_CNF_DATA_T tData;
} ECM_IF_READ_DIAG_LOG_ENTRY_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0 in case of error 28 otherwise	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E31	ECM_IF_CMD_READ_DIAG_LOG_ENTRY_CNF
Data			
ulLostEntries	UINT32	0…Size of ringbuffer	Number of lost entries, see below
tDiagEntry	ECM_DIAG_ENTRY_T		Diagnostic log entry. For detailed description see section *Entry format of diagnostic log* (page 47)

Table 62: ECM_IF_CMD_READ_DIAG_LOG_ENTRY_CNF – Read diagnostic log entry confirmation

Data field ulLostEntries

The field ulLostEntries specifies how many entries where lost since the previous ECM_IF_CMD_READ_DIAG_LOG_ENTRY_REQ and this one. The diagnostic log mechanism is implemented as a ring buffer which will overwrite the oldest entry when it is full during the time of a new entry being added.

4.6.2.2 Clear diagnostic log entry

This service clears the diagnostic log. If an application is restarted, it may be helpful to clear the diagnostic log. Otherwise, the application may read out entries that have no relevance anymore.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_CLEAR_DIAG_LOG_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_CLEAR_DIAG_LOG_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E32	ECM_IF_CMD_CLEAR_DIAG_LOG_REQ

Table 63: ECM_IF_CMD_CLEAR_DIAG_LOG_REQ –Clear diagnostic log request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_CLEAR_DIAG_LOG_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_CLEAR_DIAG_LOG_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E33	ECM_IF_CMD_CLEAR_DIAG_LOG_CNF

Table 64: ECM_IF_CMD_CLEAR_DIAG_LOG_CNF – Clear diagnostic log confirmation

4.6.3 Diagnostic log indication handling

This chapter will first outline the handling of the diagnostic log indications. Afterwards, the packets will be described.

The main purpose of the diagnostic log is to provide a history of EtherCAT or master specific events. This extends the diagnosis capabilities. In addition, the diagnostic log can be used for diagnosing network boot up problems.

Note:	As the diagnostic log is an asynchronously handled queue with respect to the master state machines, the reasons why the event happened may not exist anymore at the time of the read out.

4.6.3.1 Flow diagram of handling of diagnostic log indications

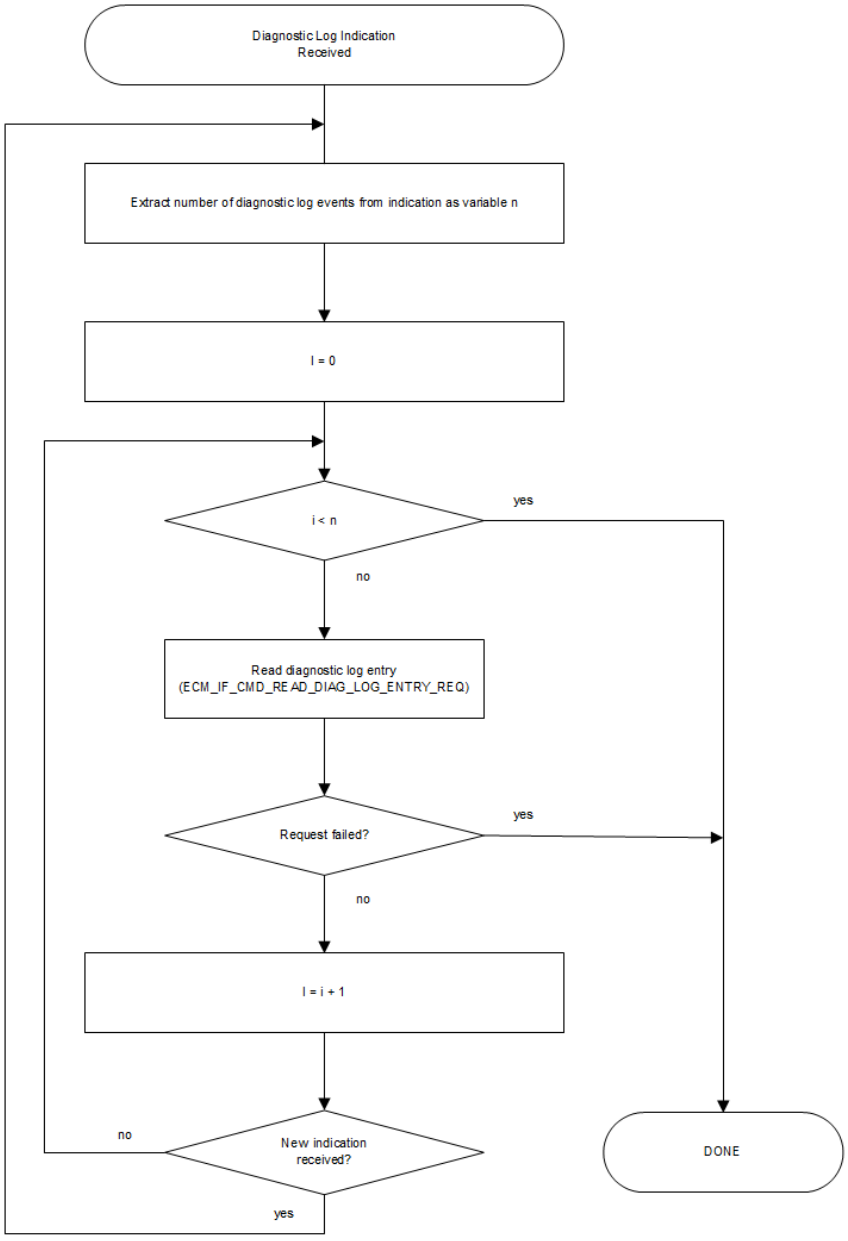


Figure 3: Flow diagram diagnostic log indications handling

The diagnostic log indications provide the current count of entry in the diagnostic log. Therefore, a state machine, handling the read out, must be able to reset the diagnostic log entry count as shown in the flow diagram.

Step read diagnostic log entry

This step includes presenting and/or dealing with the diagnostic log entry when the request has been successful. The decisions taken on the diagnostic log are up to the application. However, the read out is asynchronous to the state machine in the master. Therefore, the actual reason for the event may not be valid anymore.

4.6.3.2 Register for diagnostic log indications service

This packet registers an application task for receiving indications that new diagnostic entries are available in the diagnostic log.

The following status indication will be sent to the application task after successful registration:

- *New diagnostic log entries available indication* (page 69)

If the application does not want to receive those indications anymore, it has to use the Unregister from diagnostic log indications (page 68).

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_DIAG_LOG_INDICATIONS_REGISTER_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_DIAG_LOG_INDICATIONS_REGISTER_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E36	ECM_IF_CMD_DIAG_LOG_INDICATIONS_REGISTER_REQ

Table 65: ECM_IF_CMD_DIAG_LOG_INDICATIONS_REGISTER_REQ – Register for diagnostic log indications request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_DIAG_LOG_INDICATIONS_REGISTER_CNF_DATA_Ttag
{
  uint16_t usReserved;
} ECM_IF_DIAG_LOG_INDICATIONS_REGISTER_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_DIAG_LOG_INDICATIONS_REGISTER_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_DIAG_LOG_INDICATIONS_REGISTER_CNF_DATA_T tData;
} ECM_IF_DIAG_LOG_INDICATIONS_REGISTER_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0 in case of error 2 otherwise	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E37	ECM_IF_CMD_DIAG_LOG_INDICATIONS_REGISTER_CNF
Data			
usNumOfDiagEntries	UINT16		Number of Diagnostic Entries (only present in case of successful execution)

Table 66: ECM_IF_CMD_DIAG_LOG_INDICATIONS_REGISTER_CNF – Register for diagnostic log indications confirmation

4.6.3.3 Unregister from diagnostic log indications service

This packet deregisters an application task from receiving diagnostic indications.

The following diagnostic indications will not be sent anymore to the application after successful deregistration:

- *New diagnostic log entries available indication* (page 69)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_DIAG_LOG_INDICATIONS_UNREGISTER_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_DIAG_LOG_INDICATIONS_UNREGISTER_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E38	ECM_IF_CMD_DIAG_LOG_INDICATIONS_UNREGISTER_REQ

Table 67: ECM_IF_CMD_DIAG_LOG_INDICATIONS_UNREGISTER_REQ – Unregister from diagnostic log indications request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_DIAG_LOG_INDICATIONS_UNREGISTER_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_DIAG_LOG_INDICATIONS_UNREGISTER_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E39	ECM_IF_CMD_DIAG_LOG_INDICATIONS_UNREGISTER_CNF

Table 68: ECM_IF_CMD_DIAG_LOG_INDICATIONS_UNREGISTER_CNF - Unregister from diagnostic log indications confirmation

4.6.3.4 New diagnostic log entries available indication

This indication is sent every time a new diagnostic log entry is available if the application has previously been registered for this service using the *Register for diagnostic log indications service* (page 67).

The parameter `usNumOfDiagEntries` contains the number of new entries in the log which can subsequently be read out using the *Read diagnostic log entry service* (62).

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_NEW_DIAG_LOG_ENTRIES_IND_DATA_Ttag
{
  uint16_t usNumOfDiagEntries;
} ECM_IF_NEW_DIAG_LOG_ENTRIES_IND_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_NEW_DIAG_LOG_ENTRIES_IND_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_NEW_DIAG_LOG_ENTRIES_IND_DATA_T tData;
} ECM_IF_NEW_DIAG_LOG_ENTRIES_IND_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	2	Packet Data Length in bytes
ulCmd	UINT32	0x9E34	`ECM_IF_CMD_NEW_DIAG_LOG_ENTRIES_IND`
Data			
usNumOfDiagEntries	UINT16		Number of diagnostic entries

Table 69: `ECM_IF_CMD_NEW_DIAG_LOG_ENTRIES_IND` – New diagnostic log entries available indication

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_NEW_DIAG_LOG_ENTRIES_RES_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_NEW_DIAG_LOG_ENTRIES_RES_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue handle, unchanged
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E35	`ECM_IF_CMD_NEW_DIAG_LOG_ENTRIES_RES`

Table 70: `ECM_IF_CMD_NEW_DIAG_LOG_ENTRIES_RES` – New diagnostic log entries available response

4.7 CoE services

CoE services includes the following services if supported by slave:

- SDO access (data)
- SDOINFO access (structure info, not all slaves)

4.7.1 Slave state accessibility

CoE access is possible in the following slave states if supported by slave:

- PREOP
- SAFEOP
- OP

Slaves may limit access depending on slave state to certain objects.

4.7.2 SDO access

4.7.2.1 Fragmentation of download/write SDO

Flow charts are presented in section *SDO fragmentation flowcharts* (page 78).

When the data fits into a single fragment, the following sequence is used:

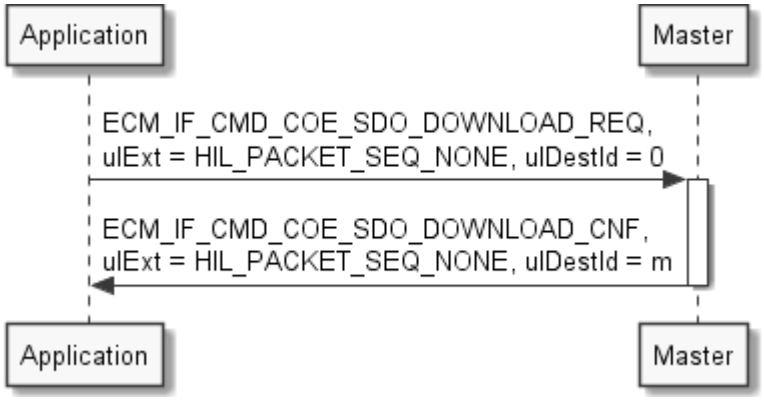


Figure 4: Single fragment handling of download/write SDO Service

When two or more fragments are required for the transfer, the following sequence diagrams apply:

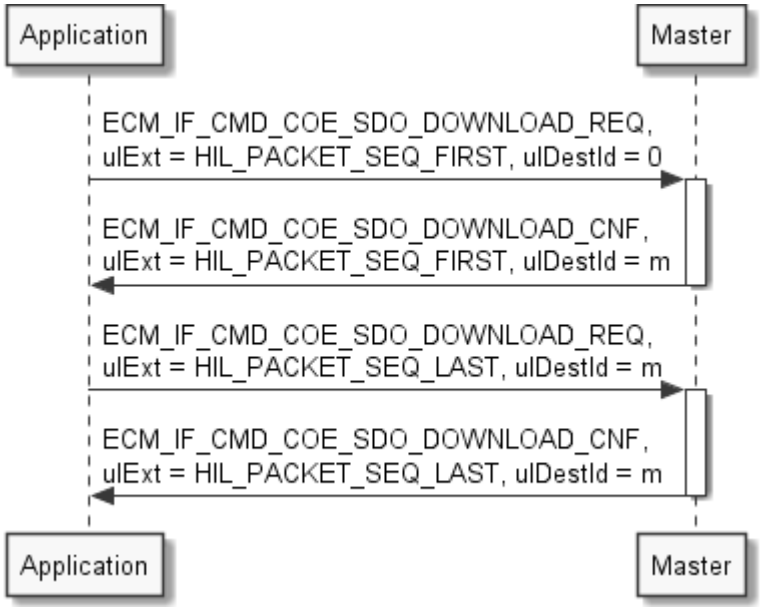


Figure 5: Two fragment handling of download/write SDO Service

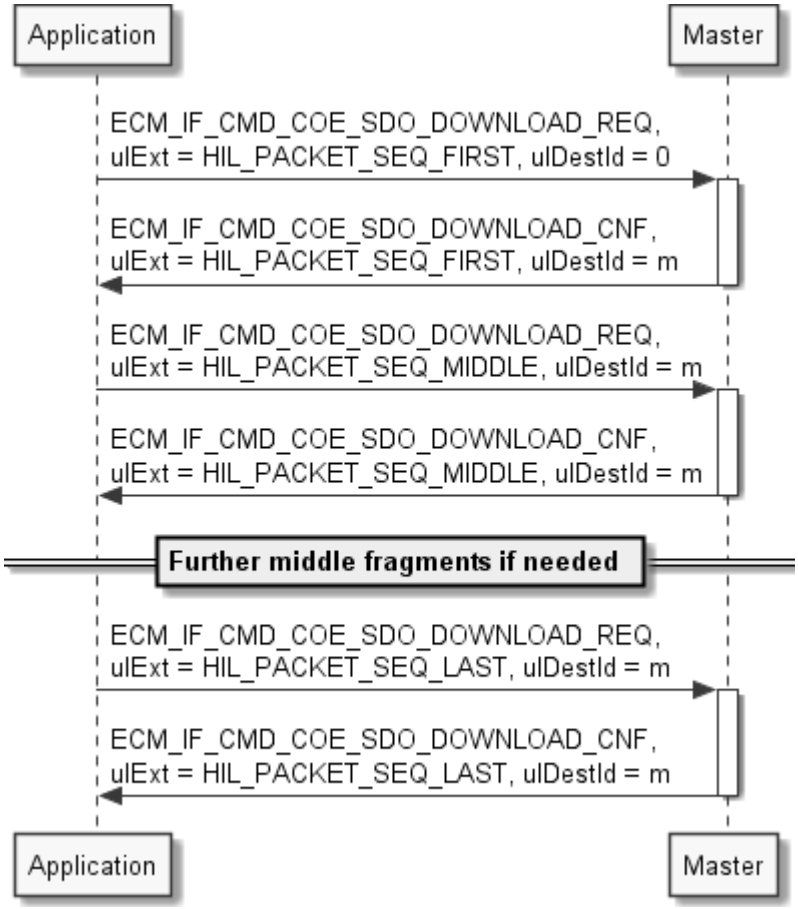


Figure 6: Multiple fragment handling of download/write SDO Service

4.7.2.2 Fragmentation of upload/read SDO

Flow charts are presented in section *SDO fragmentation flowcharts* (page 78).

When the data fit into a single fragment, the following sequence is used:

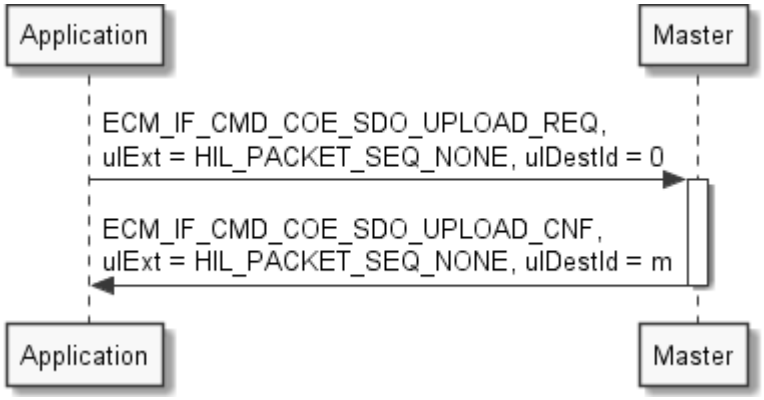


Figure 7: Single fragment handling of upload/read SDO Service

When two or more fragments are required for the transfer, the following sequence diagrams apply:

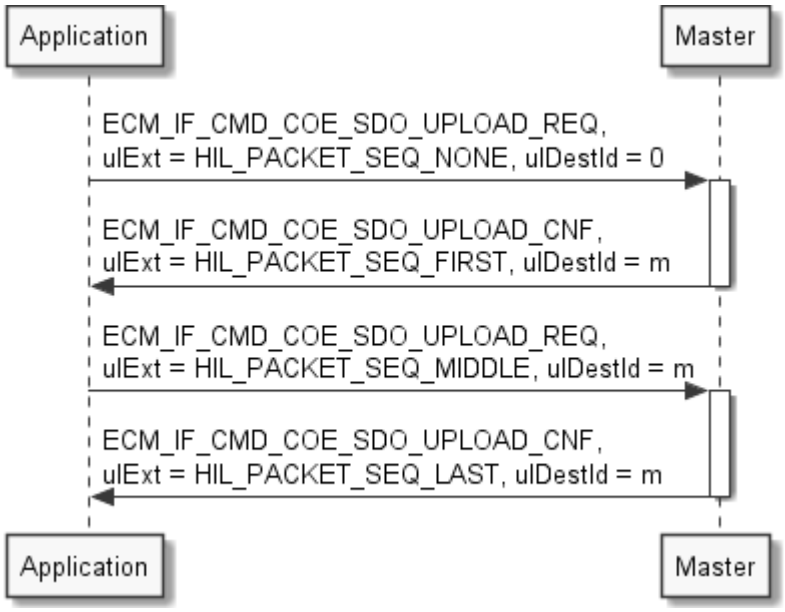


Figure 8: Two fragment handling of upload/read SDO Service

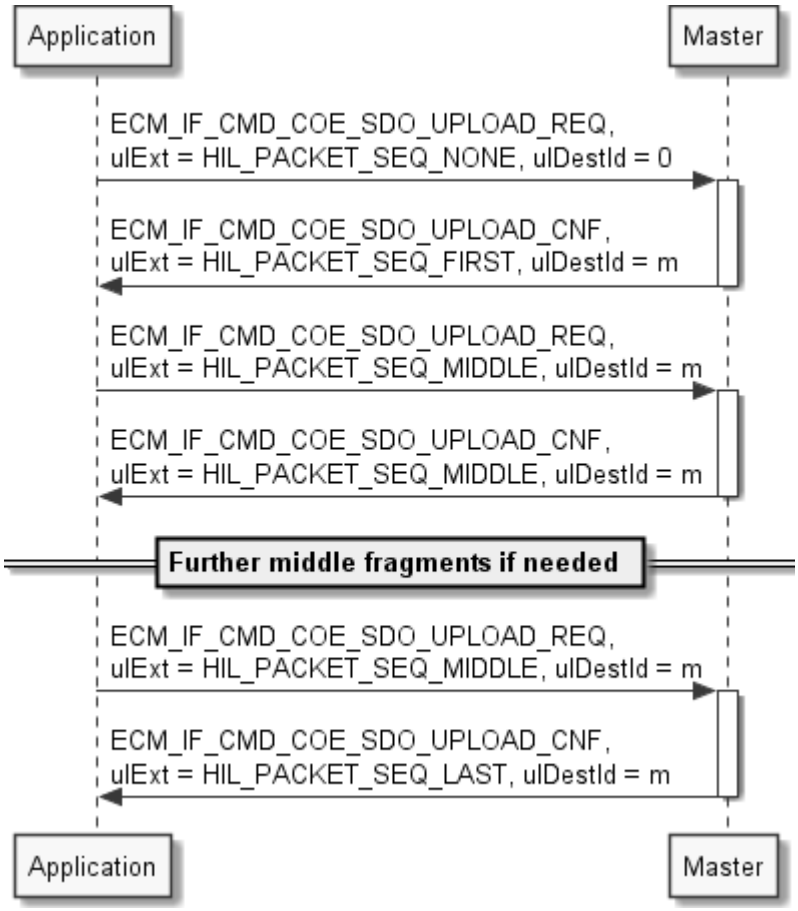


Figure 9: Multiple fragment handling of upload/read SDO Service

4.7.2.3 Download/write SDO

This packet allows writing to an object/ subobject. It supports complete access and fragmentation.

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

`ulDestId` has to be handled as follows:

- First fragment has `ulDestId` == 0
- Stack returns first fragment confirmation with `ulDestId` != 0
- This `ulDestId` has to be provided to all subsequent fragments

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDO_DOWNLOAD_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_COE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_COE_TRANSPORT_TYPE_AOE is selected */
  uint16_t usObjIndex;
  uint8_t bSubIndex;
  uint8_t fCompleteAccess;
  uint32_t ulTotalBytes; /* has to be set to summed length of all abData of all fragments
*/
  uint32_t ulTimeoutMs;
  uint8_t abData[1024];
} ECM_IF_COE_SDO_DOWNLOAD_REQ_DATA_T;
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDO_DOWNLOAD_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDO_DOWNLOAD_REQ_DATA_T tData;
} ECM_IF_COE_SDO_DOWNLOAD_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	18 + n	Packet Data Length in bytes
ulCmd	UINT32	0x9A00	`ECM_IF_CMD_COE_SDO_DOWNLOAD_REQ`
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: CoE transport (`ECM_IF_COE_TRANSPORT_COE`), 1: AoE transport (`ECM_IF_COE_TRANSPORT_AOE`)
usAoEPort	UINT16	0…65535	AoEPort (only used if usTransportType = 1)
usObjIndex	UINT16	0…65535	Index of the object
bSubIndex	UINT8	0 … 255 0 … 1	If a single subindex is requested, it refers to its subindex If complete access, it defines the start sub index
fCompleteAccess	BOOL	`TRUE`, `FALSE`	If `TRUE`, complete access is requested If `FALSE`, a single subindex is accessed
ulTotalBytes	UINT32	0 ... 2^{32}-1	Summed length of all `abData` of all fragments
ulTimeoutMs	UINT32		Timeout in ms
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as `ulLen - 18`

Table 71: `ECM_IF_CMD_COE_SDO_DOWNLOAD_REQ` – Download/write SDO request

Timeout value ulTimeoutMs

It is recommended to use at least 1000 ms as value. However, slaves exist that need higher timeout value due to their way of functioning.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDO_DOWNLOAD_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType;
  uint16_t usAoEPort;
  uint16_t usObjIndex;
  uint8_t bSubIndex;
  uint8_t fCompleteAccess;
  uint32_t ulTotalBytes;
  uint32_t ulTimeoutMs;
} ECM_IF_COE_SDO_DOWNLOAD_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDO_DOWNLOAD_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDO_DOWNLOAD_CNF_DATA_T tData;
} ECM_IF_COE_SDO_DOWNLOAD_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	18	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9A01	ECM_IF_CMD_COE_SDO_DOWNLOAD_CNF
Data			
usStationAddress	UINT16	Valid address	Value from request
usTransportType	UINT16	0,1	Value from request
usAoEPort	UINT16		Value from request
usObjIndex	UINT16	0…65535	Value from request
bSubIndex	UINT8	0 … 255	Value from request
fCompleteAccess	BOOL	TRUE, FALSE	Value from request
ulTotalBytes	UINT32		Value from request
ulTimeoutMs	UINT32		Value from request

Table 72: ECM_IF_CMD_COE_SDO_DOWNLOAD_CNF – Download/write SDO confirmation

4.7.2.4 Upload/read SDO

This packet allows reading to an object/ subobject. It supports complete access and fragmentation.

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

`ulDestId` has to be handled as follows:

- The first fragment has `ulDestId` == 0
- The stack returns the first fragment confirmation with `ulDestId` != 0
- This returned `ulDestId` has to be provided to all subsequent fragments.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDO_UPLOAD_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_COE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_COE_TRANSPORT_TYPE_AOE is selected */
  uint16_t usObjIndex;
  uint8_t bSubIndex;
  uint8_t fCompleteAccess;
  uint32_t ulTimeoutMs;
  uint32_t ulMaxTotalBytes;
} ECM_IF_COE_SDO_UPLOAD_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDO_UPLOAD_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDO_UPLOAD_REQ_DATA_T tData;
} ECM_IF_COE_SDO_UPLOAD_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	18	Packet Data Length in bytes
ulCmd	UINT32	0x9A02	`ECM_IF_CMD_COE_SDO_UPLOAD_REQ`
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: CoE transport (`ECM_IF_COE_TRANSPORT_COE`), 1: AoE transport (`ECM_IF_COE_TRANSPORT_AOE`)
usAoEPort	UINT16	0…65535	AoEPort (only used if usTransportType = 1)
usObjIndex	UINT16	0…65535	Index of the object
bSubIndex	UINT8	0 … 255 0 … 1	If a single subindex is requested, it refers to its subindex If complete access, it defines the start sub index
fCompleteAccess	BOOL	TRUE, FALSE	If `TRUE`, complete access is requested If `FALSE`, a single subindex is accessed
ulTimeoutMs	UINT32		Timeout in ms
ulMaxTotalBytes	UINT32	0 ... 2^{32}-1	Maximum number of total data bytes to be requested

Table 73: `ECM_IF_CMD_COE_SDO_UPLOAD_REQ` – Upload/read SDO request

Timeout value `ulTimeoutMs`

It is recommended to set this value at least to 1000 ms. However, some slaves require even higher timeout values due to their internal functionality.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDO_UPLOAD_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType;
  uint16_t usAoEPort;
  uint16_t usObjIndex;
  uint8_t bSubIndex;
  uint8_t fCompleteAccess;
  uint32_t ulTimeoutMs;
  uint32_t ulTotalBytes; /* summed length of all abData confirmation fragments */
  uint8_t abData[1024]; /* actual length is given by ulLen -
offsetof(ECM_IF_COE_SDO_UPLOAD_CNF_DATA_T, abData) */
} ECM_IF_COE_SDO_UPLOAD_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDO_UPLOAD_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDO_UPLOAD_CNF_DATA_T tData;
} ECM_IF_COE_SDO_UPLOAD_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	18 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9A03	`ECM_IF_CMD_COE_SDO_UPLOAD_CNF`
Data			
usStationAddress	UINT16	Valid address	Value from request
usTransportType	UINT16	0, 1	Value from request
usAoEPort	UINT16		Value from request
usObjIndex	UINT16		Value from request
bSubIndex	UINT8	0 … 255	Value from request
fCompleteAccess	BOOL	TRUE, FALSE	Value from request
ulTimeoutMs	UINT32		Timeout in ms
ulTotalBytes	UINT32		Summed length of all abData of all fragments
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as ulLen - 18

Table 74: `ECM_IF_CMD_COE_SDO_UPLOAD_CNF` – Upload/read SDO confirmation

4.7.2.5 SDO fragmentation flowcharts

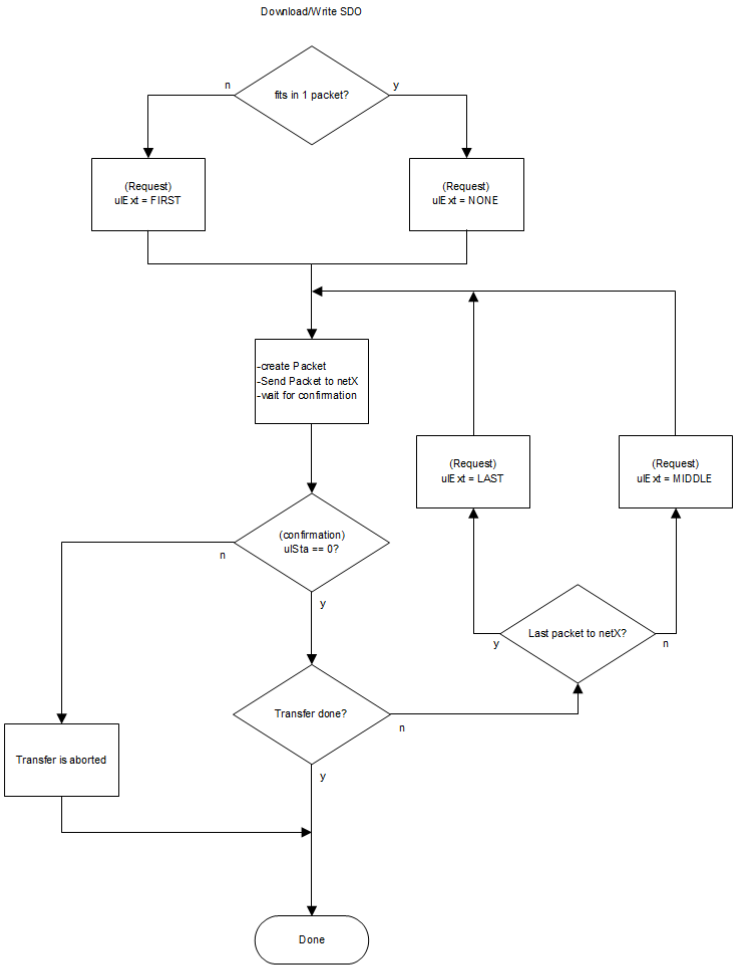


Figure 10: Flowchart for download / write SDO fragmentation

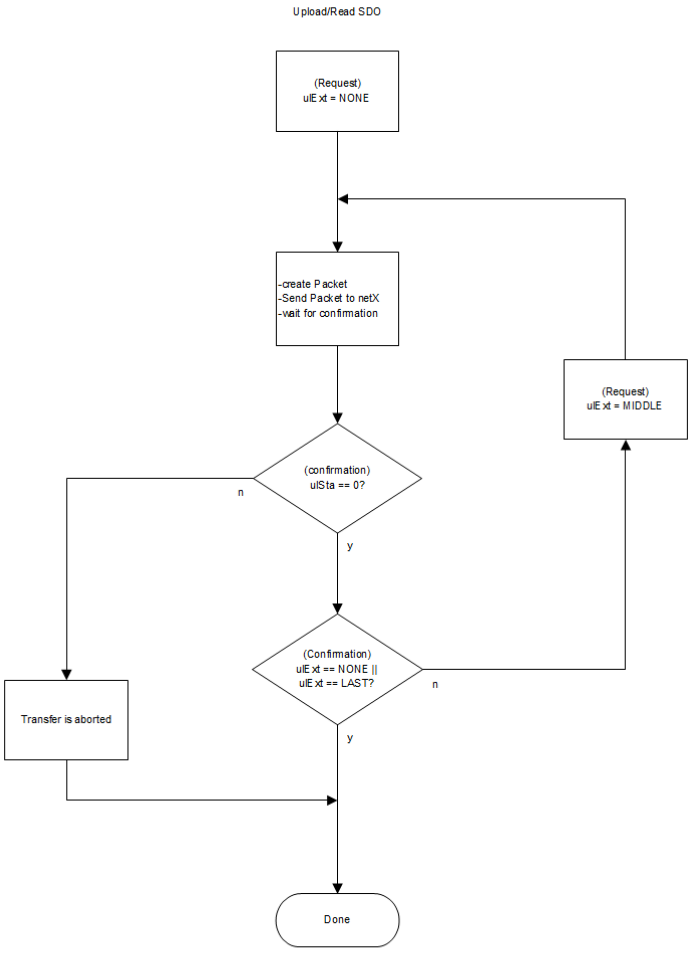


Figure 11: Flowchart for upload/ read SDO Fragmentation

4.7.3 SDOINFO access

The application can use these services to get SDO information from a slave:

- Get object list
- Get object description
- Get entry description

All SDOinfo services are processed acyclic. Please note that the response time of the SDOinfo services dependens on the slave mailbox response time, the bus cycle time and the amount of other acyclic traffic. Based on this, the timeout has to be choosen with an appropriate margin, in order that the next request starts after the previous request has been finished.

4.7.3.1 Fragmentation of SDOINFO.GetOdList

Flow charts are presented in section *SDOINFO fragmentation flowcharts* (page 94).

When the data fit into a single fragment, the following sequence is used:

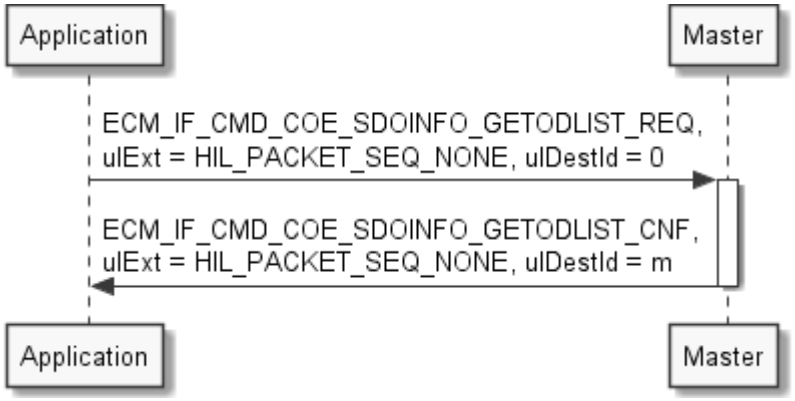


Figure 12: Single fragment handling of GetOdList SDOINFO service

When two or more fragments are required for the transfer, the following sequence diagrams apply:

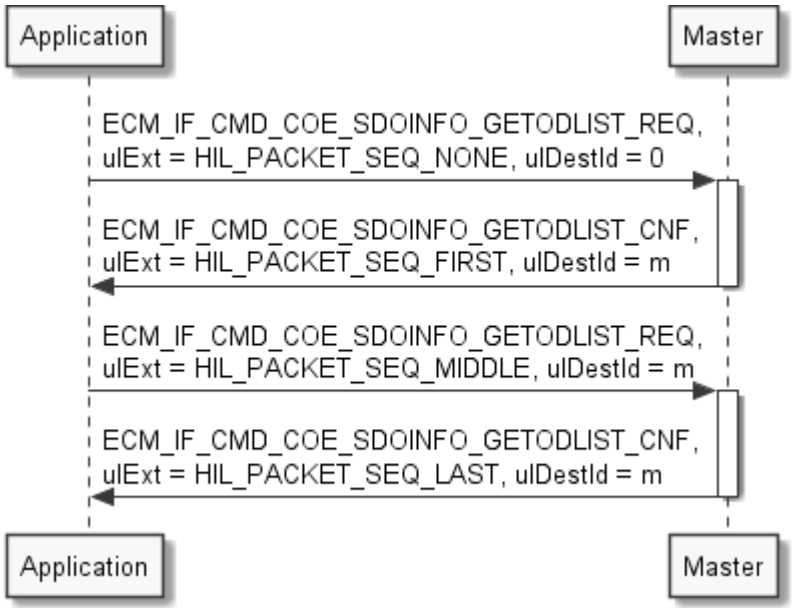


Figure 13: Two fragment handling of GetOdList SDOINFO service

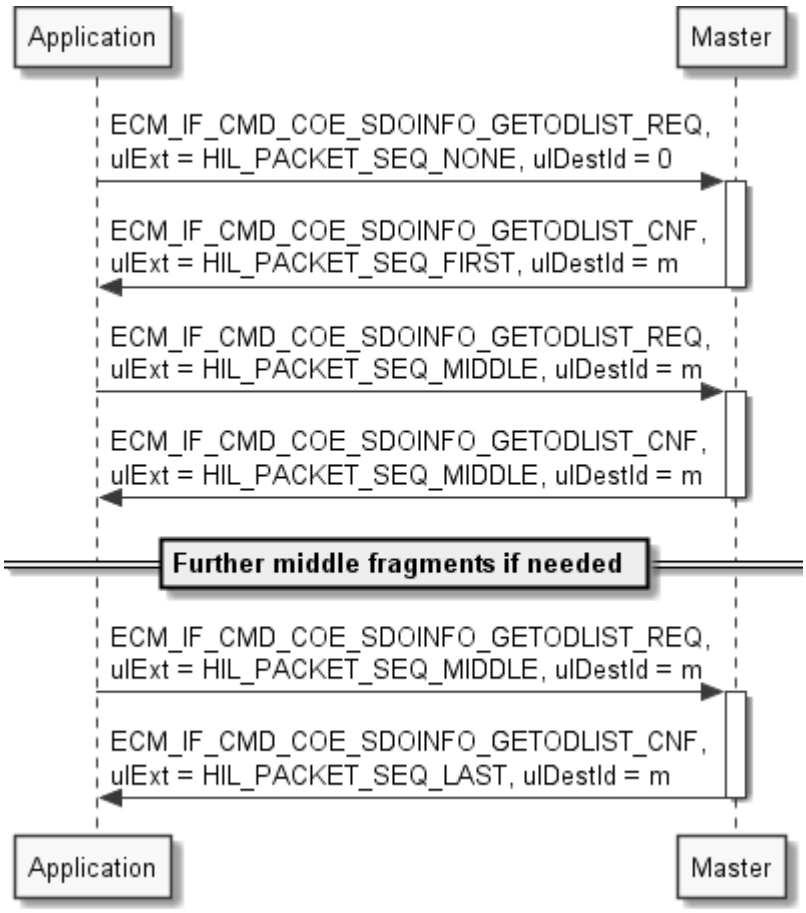


Figure 14: Multiple fragment handling of GetOdList SDOINFO service

4.7.3.2 Fragmentation of SDOINFO.GetObjDesc

Flow charts are presented in section *SDOINFO fragmentation flowcharts* (page 94).

When the data fit into a single fragment, the following sequence is used:

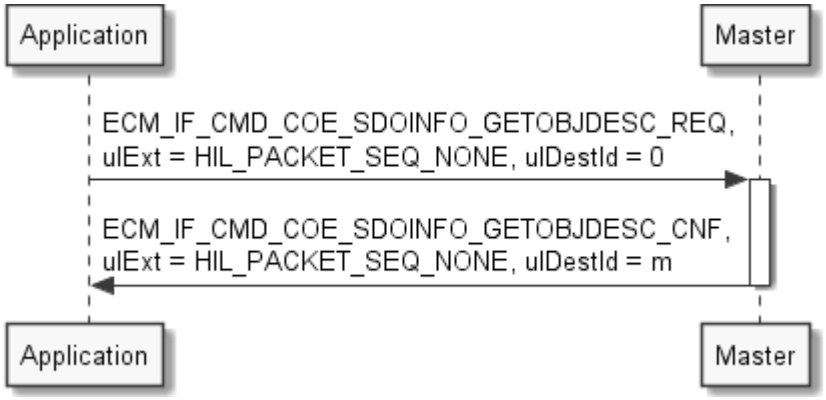


Figure 15: Single fragment handling of GetObjDesc SDOINFO Service

When two or more fragments are required for the transfer, the following sequence diagrams apply:

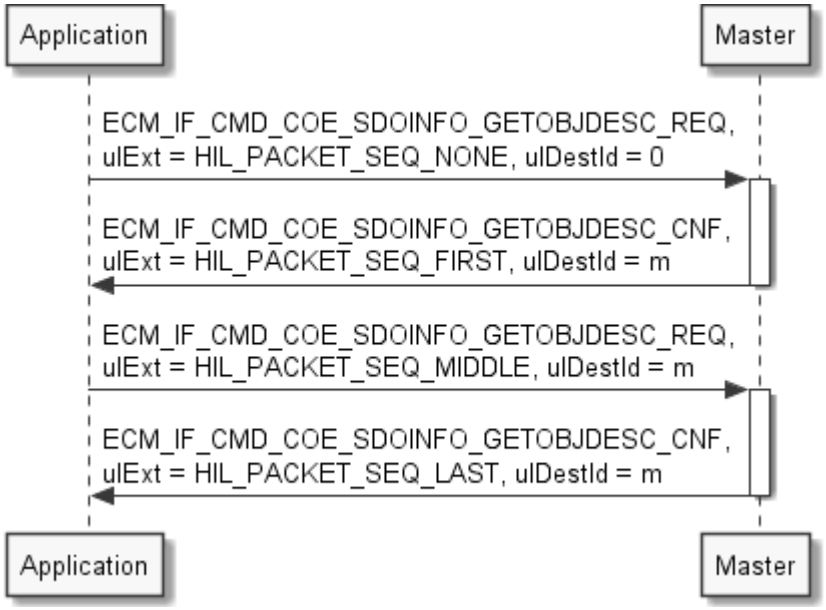


Figure 16: Two fragment handling of GetObjDesc SDOINFO Service

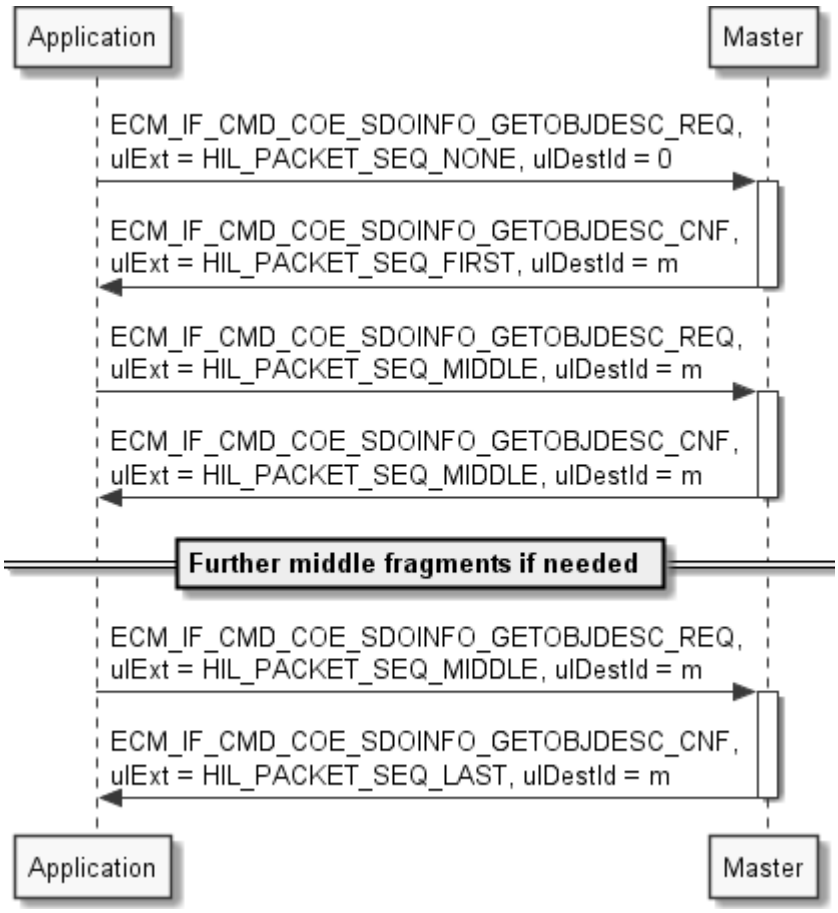


Figure 17: Multiple fragment handling of GetObjDesc SDOINFO Service

4.7.3.3 Fragmentation of SDOINFO.GetEntryDesc

Flow charts are presented in section *SDOINFO fragmentation flowcharts* (page 94).

When the data fit into a single fragment, the following sequence is used:

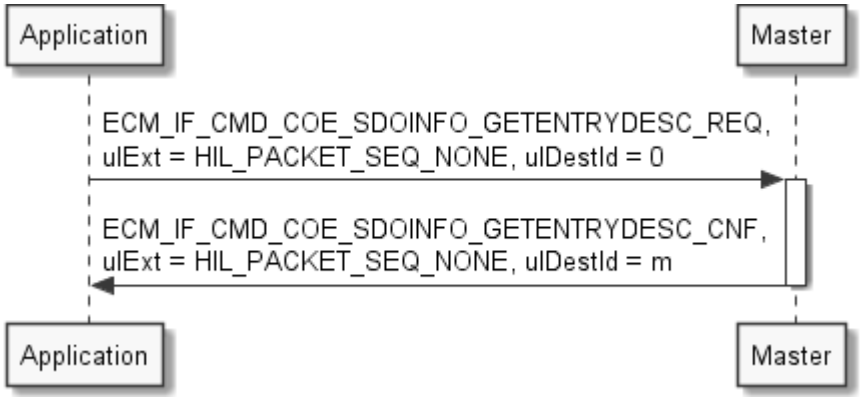


Figure 18: Single fragment handling of GetEntryDesc SDOINFO Service

When two or more fragments are required for the transfer, the following sequence diagrams apply:

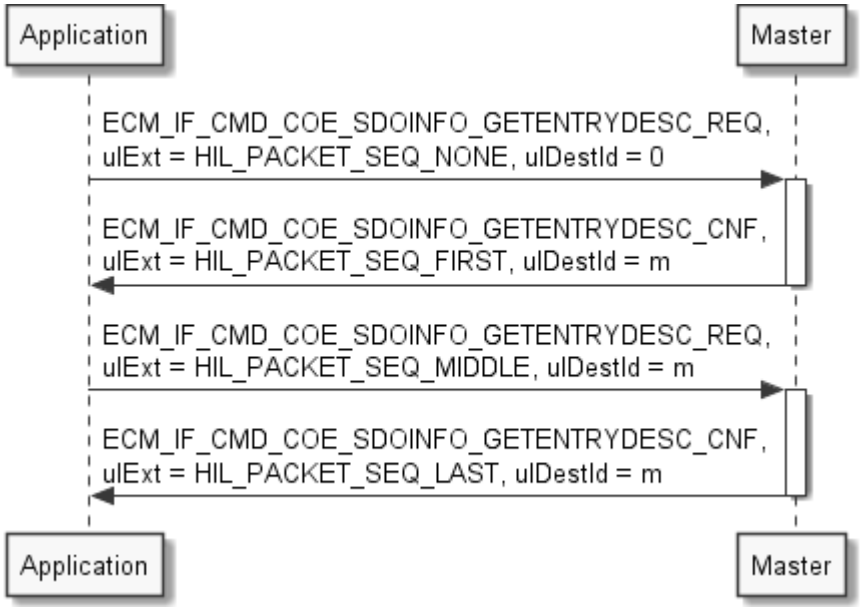


Figure 19: Two fragment handling of GetEntryDesc SDOINFO Service

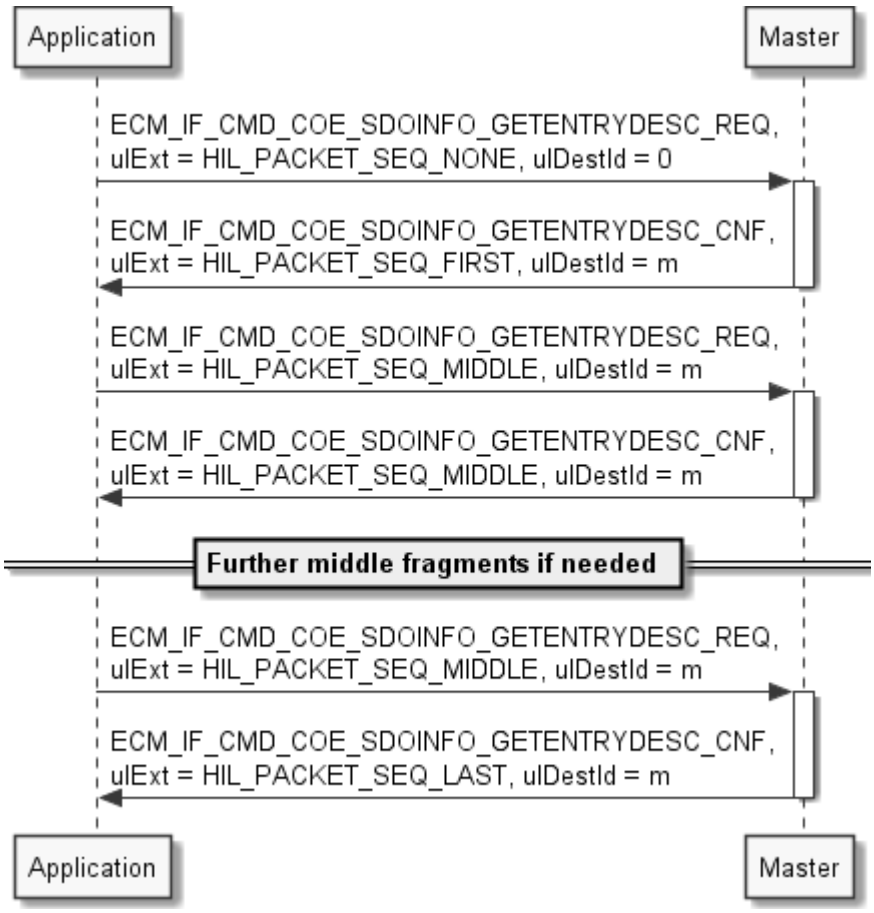


Figure 20: Multiple fragment handling of GetEntryDesc SDOINFO Service

4.7.3.4 Get object list (OdList)

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

`ulDestId` has to be handled as follows:

- The first fragment has `ulDestId` == 0
- The stack returns the first fragment confirmation with `ulDestId` != 0
- This returned `ulDestId` has to be provided to all subsequent fragments

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_COE_SDOINFO_GETODLIST_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_COE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_COE_TRANSPORT_TYPE_AOE is selected */
  uint16_t usListType;
  uint32_t ulTimeoutMs;
  uint32_t ulMaxTotalBytes;
} ECM_IF_COE_SDOINFO_GETODLIST_REQ_DATA_T;

enum ECM_IF_COE_SDOINFO_GETODLIST_LIST_TYPE_Etag
{
  ECM_IF_COE_SDOINFO_GETODLIST_LIST_TYPE_COUNTS = 0,
  ECM_IF_COE_SDOINFO_GETODLIST_LIST_TYPE_ALL = 1,
  ECM_IF_COE_SDOINFO_GETODLIST_LIST_TYPE_RXPDOMAPPABLE = 2,
  ECM_IF_COE_SDOINFO_GETODLIST_LIST_TYPE_TXPDOMAPPABLE = 3,
  ECM_IF_COE_SDOINFO_GETODLIST_LIST_TYPE_BACKUP = 4,
  ECM_IF_COE_SDOINFO_GETODLIST_LIST_TYPE_SETTINGS = 5
};

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDOINFO_GETODLIST_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDOINFO_GETODLIST_REQ_DATA_T tData;
} ECM_IF_COE_SDOINFO_GETODLIST_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	16	Packet Data Length in bytes
ulCmd	UINT32	0x9A04	`ECM_IF_CMD_COE_SDOINFO_GETODLIST_REQ`
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0, 1	Transport type 0: CoE transport (`ECM_IF_COE_TRANSPORT_COE`), 1: AoE transport (`ECM_IF_COE_TRANSPORT_AOE`)
usAoEPort	UINT16	0…65535	AoEPort (only used if usTransportType = 1)
usListType	UINT16	0 … 5	See table below
ulTimeoutMs	UINT32		Timeout in ms
ulMaxTotalBytes	UINT32	0 ... 2^{32}-1	Maximum total data bytes to be requested

Table 75: `ECM_IF_CMD_COE_SDOINFO_GETODLIST_REQ` – Get object list request

Timeout value ulTimeoutMs

It is recommended to set this value at least to 1000 ms. However, some slaves require even higher timeout values due to their internal functionality.

Definition of usListType

Value	Definition / description
0x0000	ECM_IF_COE_SDOINFO_GETODLIST_TYPE_COUNTS Retrieve the counts of lists 1 to 5
0x0001	ECM_IF_COE_SDOINFO_GETODLIST_TYPE_ALL Retrieve a list of all existing objects
0x0002	ECM_IF_COE_SDOINFO_GETODLIST_TYPE_RXPDOMAPPABLE Retrieve a list of all RxPDO objects which can be mapped
0x0003	ECM_IF_COE_SDOINFO_GETODLIST_TYPE_TXPDOMAPPABLE Retrieve a list of all TxPDO objects which can be mapped
0x0004	ECM_IF_COE_SDOINFO_GETODLIST_TYPE_BACKUP Retrieve a list of all objects necessary for backup
0x0005	ECM_IF_COE_SDOINFO_GETODLIST_TYPE_SETTINGS Retrieve a list of all objects used during startup

Table 76: Possible values of usListType

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_COE_SDOINFO_GETODLIST_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType;
  uint16_t usAoEPort;
  uint16_t usListType;
  uint32_t ulTimeoutMs;
  uint32_t ulTotalBytes; /* summed length of all abData confirmation fragments */
  uint16_t ausObjectIDs[512]; /* actual byte length is given by ulLen -
offsetof(ECM_IF_COE_SDOINFO_GETODLIST_CNF_DATA_T, abData) */
} ECM_IF_COE_SDOINFO_GETODLIST_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDOINFO_GETODLIST_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDOINFO_GETODLIST_CNF_DATA_T tData;
} ECM_IF_COE_SDOINFO_GETODLIST_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	16 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9A05	`ECM_IF_CMD_COE_SDOINFO_GETODLIST_CNF`
Data			
usStationAddress	UINT16	Valid address	Value from request
usTransportType	UINT16	0, 1	Value from request
usAoEPort	UINT16		Value from request
usListType	UINT16	0 … 5	Value from request
ulTimeoutMs	UINT32		Value from request
ulTotalBytes	UINT32		Summed length of all `abData` of all fragments
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as `ulLen - 16`

Table 77: `ECM_IF_CMD_COE_SDOINFO_GETODLIST_CNF` – Get object list confirmation

4.7.3.5 Get object description (ObjDesc)

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 4.1.3)
- During normal operation, use *Fixed station address* (page 4.1.2)

`ulDestId` has to be handled as follows:

- The first fragment has `ulDestId` == 0
- The stack returns the first fragment confirmation with `ulDestId` != 0
- This returned `ulDestId` has to be provided to all subsequent fragments

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_COE_SDOINFO_GETOBJDESC_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_COE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_COE_TRANSPORT_TYPE_AOE is selected */
  uint16_t usObjIndex;
  uint32_t ulTimeoutMs;
  uint32_t ulMaxTotalBytes;
} ECM_IF_COE_SDOINFO_GETOBJDESC_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDOINFO_GETOBJDESC_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDOINFO_GETOBJDESC_REQ_DATA_T tData;
} ECM_IF_COE_SDOINFO_GETOBJDESC_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	16	Packet Data Length in bytes
ulCmd	UINT32	0x9A06	`ECM_IF_CMD_COE_SDOINFO_GETOBJDESC_REQ`
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: CoE transport (`ECM_IF_COE_TRANSPORT_COE`), 1: AoE transport (`ECM_IF_COE_TRANSPORT_AOE`)
usAoEPort	UINT16	0…65535	AoEPort (only used if usTransportType = 1)
usObjIndex	UINT16	0…65535	Index of object
ulTimeoutMs	UINT32		Timeout in ms
ulMaxTotalBytes	UINT32	0 ... 2^{32}-1	Maximum total data bytes to be requested

Table 78: `ECM_IF_CMD_COE_SDOINFO_GETOBJDESC_REQ` – Get object description request

Timeout value ulTimeoutMs

It is recommended to set this value at least to 1000 ms. However, some slaves require even higher timeout values due to their internal functionality.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_COE_SDOINFO_GETOBJDESC_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType;
  uint16_t usAoEPort;
  uint16_t usObjIndex;
  uint32_t ulTimeoutMs;
  uint32_t ulTotalBytes; /* summed length of all abData confirmation fragments */
  uint8_t abData[1024]; /* actual length is given by ulLen -
offsetof(ECM_IF_COE_SDOINFO_GETOBJDESC_CNF_DATA_T, abData) */
} ECM_IF_COE_SDOINFO_GETOBJDESC_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_COE_SDOINFO_GETOBJDESC_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDOINFO_GETOBJDESC_CNF_DATA_T tData;
} ECM_IF_COE_SDOINFO_GETOBJDESC_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	16 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9A07	`ECM_IF_CMD_COE_SDOINFO_GETOBJDESC_CNF`
Data			
usStationAddress	UINT16	Valid address	Value from request
usTransportType	UINT16	0,1	Value from request
usAoEPort	UINT16		Value from request
usObjIndex	UINT16	0…65535	Value from request
ulTimeoutMs	UINT32		Value from request
ulTotalBytes	UINT32		Summed length of all `abData` of all fragments
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as `ulLen - 18`

Table 79: `ECM_IF_CMD_COE_SDOINFO_GETOBJDESC_CNF` – Get object description confirmation

4.7.3.6 Get entry description (EntryDesc)

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

`ulDestId` has to be handled as follows:

- The first fragment has `ulDestId` == 0
- The stack returns the first fragment confirmation with `ulDestId` != 0
- This returned `ulDestId` has to be provided to all subsequent fragments

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_COE_SDOINFO_GETENTRYDESC_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_COE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_COE_TRANSPORT_TYPE_AOE is selected */
  uint16_t usObjIndex;
  uint8_t bSubIndex;
  uint8_t bRequestedValueInfo;
  uint32_t ulTimeoutMs;
  uint32_t ulMaxTotalBytes;
} ECM_IF_COE_SDOINFO_GETENTRYDESC_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_COE_SDOINFO_GETENTRYDESC_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDOINFO_GETENTRYDESC_REQ_DATA_T tData;
} ECM_IF_COE_SDOINFO_GETENTRYDESC_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	16	Packet Data Length in bytes
ulCmd	UINT32	0x9A08	`ECM_IF_CMD_COE_SDOINFO_GETENTRYDESC_REQ`
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: CoE transport (`ECM_IF_COE_TRANSPORT_COE`), 1: AoE transport (`ECM_IF_COE_TRANSPORT_AOE`)
usAoEPort	UINT16	0…65535	AoEPort (only used if usTransportType = 1)
usObjIndex	UINT16	0…65535	Index of subobject
bSubIndex	UINT8	0…255	Subindex of subobject
bRequestedValueInfo	UINT8		
ulTimeoutMs	UINT32		Timeout in ms
ulMaxTotalBytes	UINT32	0 ... 2^{32}-1	Maximum total data bytes to be requested

Table 80: `ECM_IF_CMD_COE_SDOINFO_GETENTRYDESC_REQ` – Get entry description request

Timeout value ulTimeoutMs

It is recommended to set this value at least to 1000 ms. However, some slaves require even higher timeout values due to their internal functionality.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_COE_SDOINFO_GETENTRYDESC_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType;
  uint16_t usAoEPort;
  uint16_t usObjIndex;
  uint8_t bSubIndex;
  uint8_t bValueInfo;
  uint32_t ulTimeoutMs;
  uint32_t ulTotalBytes; /* summed length of all abData confirmation fragments */
  uint8_t abData[1024]; /* actual length is given by ulLen -
offsetof(ECM_IF_COE_SDOINFO_GETENTRYDESC_CNF_DATA_T, abData) */
} ECM_IF_COE_SDOINFO_GETENTRYDESC_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_COE_SDOINFO_GETENTRYDESC_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_COE_SDOINFO_GETENTRYDESC_CNF_DATA_T tData;
} ECM_IF_COE_SDOINFO_GETENTRYDESC_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	18 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9A09	ECM_IF_COE_SDOINFO_GETENTRYDESC_CNF
Data			
usStationAddress	UINT16	Valid address	Value from request
usTransportType	UINT16	0,1	Value from request
usAoEPort	UINT16		Value from request
usObjIndex	UINT16	0…65535	Value from request
bSubIndex	UINT8	0…255	Value from request
bValueInfo	UINT8		Returned value info
ulTimeoutMs	UINT32		Value from request
ulTotalBytes	UINT32		Summed length of all abData of all fragments
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as ulLen - 18

Table 81: ECM_IF_CMD_COE_SDOINFO_GETENTRYDESC_CNF – Get entry description confirmation

Bit mask for `ulAccessBitMask` and `ulValueInfo`

Bit No.	Definition / description
6	`MSK_ECM_IF_COE_SDOINFO_GETENTRYDESC_VALUE_INFO_FLAGS_MAXIMUM_VALUE` In request, this bit defines whether Maximum Value is to be requested by master. In confirmation, this bit defines whether Maximum Value is available and has been requested.
5	`MSK_ECM_IF_COE_SDOINFO_GETENTRYDESC_VALUE_INFO_FLAGS_MINIMUM_VALUE` In request, this bit defines whether Minimum Value is to be requested by master. In confirmation, this bit defines whether Minimum Value is available and has been requested.
4	`MSK_ECM_IF_COE_SDOINFO_GETENTRYDESC_VALUE_INFO_FLAGS_DEFAULT_VALUE` In request, this bit defines whether Default Value is to be requested by master. In confirmation, this bit defines whether Default Value is available and has been requested.
3	`MSK_ECM_IF_COE_SDOINFO_GETENTRYDESC_VALUE_INFO_FLAGS_UNIT_TYPE` In request, this bit defines whether Unit Type is to be requested by master. In confirmation, this bit defines whether Unit type is available and has been requested.

Table 82: Meaning of `bRequestedValueInfo` and `bValueInfo`

4.7.3.7 SDOINFO fragmentation flowcharts

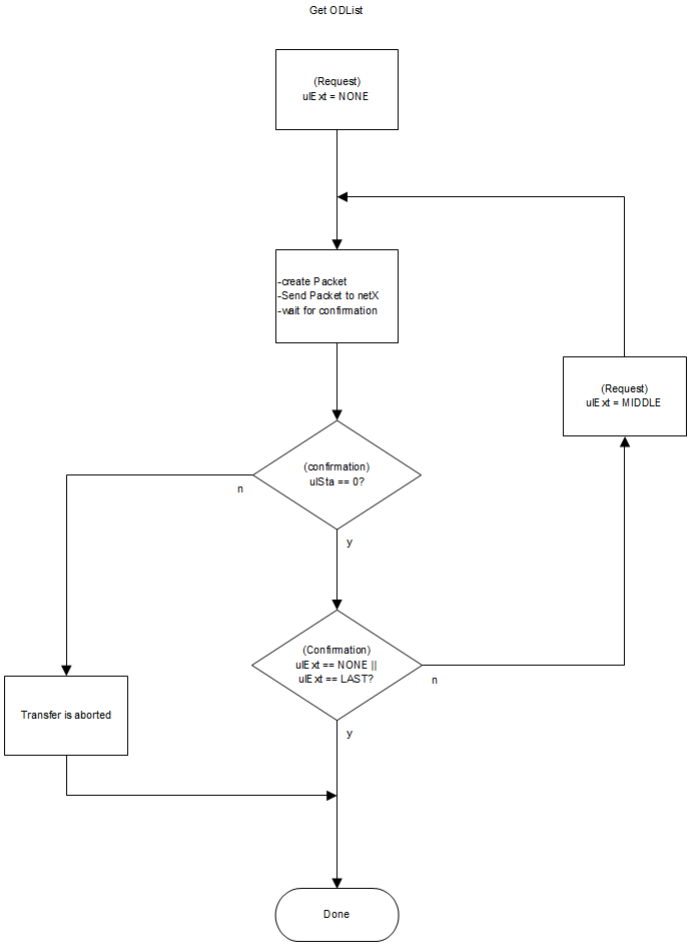


Figure 21: Flowchart for GetOdList fragmentation

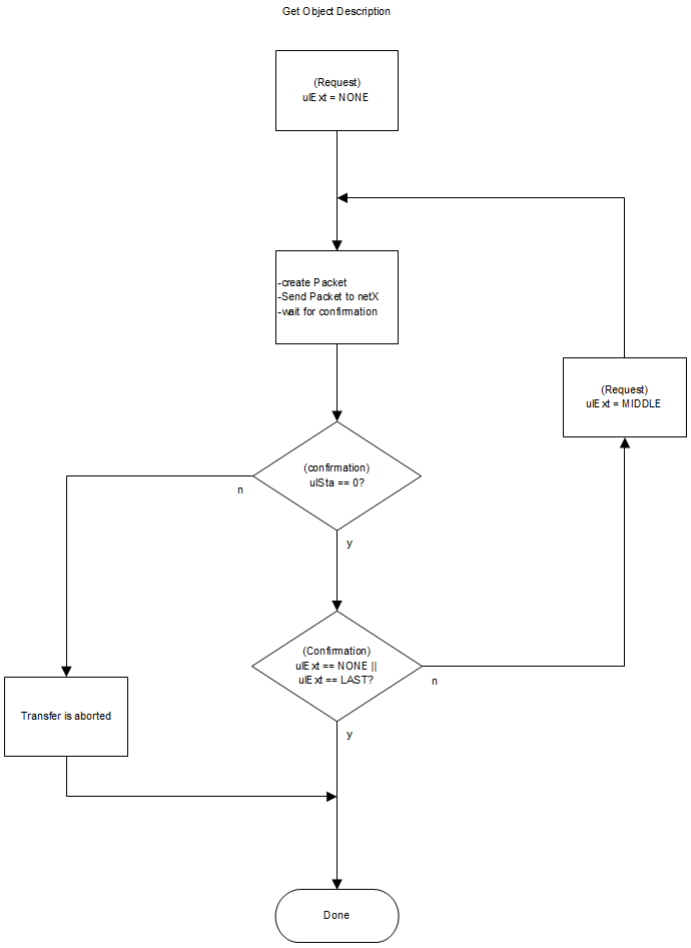


Figure 22: Flowchart for GetObjDesc fragmentation

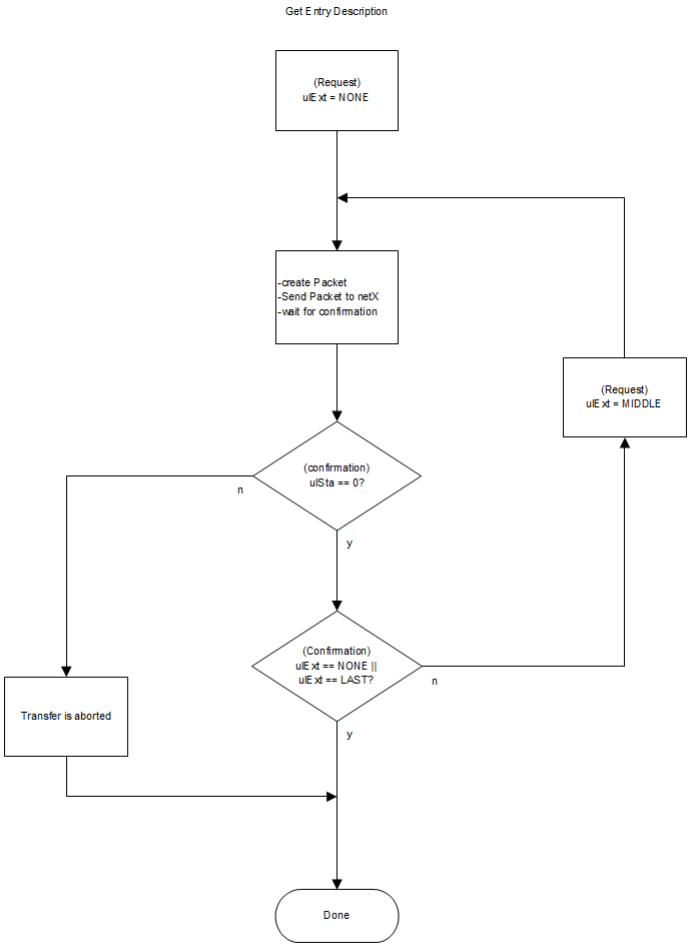


Figure 23: Flowchart for GetEntryDesc fragmentation

4.7.4 SDO access (Legacy)

4.7.4.1 Download/write SDO (Legacy)

This packet allows writing to an object/subobject.

Note: This packet is provided for applications migrating from ECM V3.X. This packet does not support fragmentation.

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

Packet structure reference

```
#define ETHERCAT_MASTER_COE_MAX_SDO_DOWNLOAD_DATA (HIL_MAX_DATA_SIZE - (sizeof(uint32_t)
* 4))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SDO_DOWNLOAD_REQ_DATA_Ttag
{
  uint32_t ulNodeId;
  uint32_t ulIndex;
  uint32_t ulSubIndex;
  uint32_t ulDataCnt;
  uint8_t abSdoData[ETHERCAT_MASTER_COE_MAX_SDO_DOWNLOAD_DATA];
} ETHERCAT_MASTER_PACKET_SDO_DOWNLOAD_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SDO_DOWNLOAD_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_SDO_DOWNLOAD_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_SDO_DOWNLOAD_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	16 + n	Packet Data Length in bytes
ulCmd	UINT32	0x650008	ETHERCAT_MASTER_CMD_SDO_DOWNLOAD_REQ
Data			
ulNodeId	UINT32		During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
ulIndex	UINT32	0 … 0xFFFF	Index
ulSubIndex	UINT32	0 … 255	Subindex
ulDataCnt	UINT32		Data count
abSdoData[n]	UINT8[]		SDO Data to be written Actual data length is defined as `ulLen - 16`

Table 83: ETHERCAT_MASTER_CMD_SDO_DOWNLOAD_REQ – Download/write SDO request (Legacy)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SDO_DOWNLOAD_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ETHERCAT_MASTER_PACKET_SDO_DOWNLOAD_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x650009	ETHERCAT_MASTER_CMD_SDO_DOWNLOAD_CNF

Table 84: ETHERCAT_MASTER_CMD_SDO_DOWNLOAD_CNF – Download/write SDO confirmation (Legacy)

4.7.4.2 Upload/read SDO (Legacy)

This packet allows reading to an object/subobject.

Note: This packet is provided for applications migrating from ECM V3.X. This packet does not support fragmentation.

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SDO_UPLOAD_REQ_DATA_Ttag
{
  uint32_t ulNodeId;
  uint32_t ulIndex;
  uint32_t ulSubIndex;
} ETHERCAT_MASTER_PACKET_SDO_UPLOAD_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SDO_UPLOAD_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_SDO_UPLOAD_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_SDO_UPLOAD_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	12	Packet Data Length in bytes
ulCmd	UINT32	0x650006	ETHERCAT_MASTER_CMD_SDO_UPLOAD_REQ
Data			
ulNodeId	UINT32		During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
ulIndex	UINT32	0…0xFFFF	Index
ulSubIndex	UINT32	0…255	Subindex

Table 85: ETHERCAT_MASTER_CMD_SDO_UPLOAD_REQ – Upload/read SDO request (Legacy)

Packet structure reference

```
#define ETHERCAT_MASTER_COE_GET_SDO_UPLOAD_CNF_LEN_ON_ERROR (12)
#define ETHERCAT_MASTER_COE_MAX_SDO_UPLOAD_DATA (HIL_MAX_DATA_SIZE - (sizeof(uint32_t) *
4))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SDO_UPLOAD_CNF_DATA_Ttag
{
  uint32_t ulNodeId;
  uint32_t ulIndex;
  uint32_t ulSubIndex;
  uint32_t ulDataCnt;
  uint8_t  abSdoData[ETHERCAT_MASTER_COE_MAX_SDO_UPLOAD_DATA];
} ETHERCAT_MASTER_PACKET_SDO_UPLOAD_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SDO_UPLOAD_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_SDO_UPLOAD_CNF_DATA_T tData;
} ETHERCAT_MASTER_PACKET_SDO_UPLOAD_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	16 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x650007	ETHERCAT_MASTER_CMD_SDO_UPLOAD_CNF
Data			
ulNodeId	UINT32		Value from request
ulIndex	UINT32		Value from request
ulSubIndex	UINT32		Value from request
ulDataCnt	UINT32		Length of data read Same value as n
abSdoData[n]	UINT8[]		Data being read Actual length is ulDataCnt

Table 86: ETHERCAT_MASTER_CMD_SDO_UPLOAD_CNF – Upload/read SDO confirmation (Legacy)

4.7.5 SDOINFO access (Legacy)

4.7.5.1 Get object list (OdList) (Legacy)

Note: This packet is provided for applications migrating from ECM V3.X. This packet does not support fragmentation.

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ODLIST_REQ_DATA_Ttag
{
  uint32_t ulNodeId;
  uint32_t ulListType;
} ETHERCAT_MASTER_PACKET_GET_ODLIST_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ODLIST_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_ODLIST_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_ODLIST_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	8	Packet Data Length in bytes
ulCmd	UINT32	0x65000A	ETHERCAT_MASTER_CMD_GET_ODLIST_REQ
Data			
ulNodeId	UINT32		During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
ulListType	UINT32	1 … 5	List type, see *Table 88: Possible values of ulListType* below

Table 87: ETHERCAT_MASTER_CMD_GET_ODLIST_REQ – Get object list request (Legacy)

Definition of ulListType

Value	Definition / description
0x0001	ETHERCAT_MASTER_COE_GET_ODLIST_TYPE_ALL Retrieve list of all existing objects
0x0002	ETHERCAT_MASTER_COE_GET_ODLIST_TYPE_RXPDOMAP Retrieve list of all RxPDO objects which can be mapped
0x0003	ETHERCAT_MASTER_COE_GET_ODLIST_TYPE_TXPDOMAP Retrieve list of all TxPDO objects which can be mapped
0x0004	ETHERCAT_MASTER_COE_GET_ODLIST_TYPE_STORE Retrieve list of all objects necessary for backup
0x0005	ETHERCAT_MASTER_COE_GET_ODLIST_TYPE_STARTUP Retrieve list of all objects used during startup

Table 88: Possible values of ulListType

Packet structure reference

```
#define ETHERCAT_MASTER_COE_GET_ODLIST_CNF_LEN_ON_ERROR (8)
#define ETHERCAT_MASTER_COE_GET_ODLIST_DATA ((HIL_MAX_DATA_SIZE - (sizeof(uint32_t) * 3))
/ sizeof(uint16_t))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ODLIST_CNF_DATA_Ttag
{
  uint32_t ulNodeId;
  uint32_t ulListType;
  uint32_t ulDataCnt;
  uint16_t ausObjectList[ETHERCAT_MASTER_COE_GET_ODLIST_DATA];
} ETHERCAT_MASTER_PACKET_GET_ODLIST_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ODLIST_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_ODLIST_CNF_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_ODLIST_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	12 + n * 2 8 on error	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x65000B	ETHERCAT_MASTER_CMD_GET_ODLIST_CNF
Data			
ulNodeId	UINT32		Value from request
ulListType	UINT32		Value from request
ulDataCnt	UINT32		Data count
ausObjectList[n]	UINT16[]		List of object indices

Table 89: ETHERCAT_MASTER_CMD_GET_ODLIST_CNF – Get object list confirmation (Legacy)

4.7.5.2 Get object description (ObjDesc) (Legacy)

Note: This packet is provided for applications migrating from ECM V3.X. This packet does not support fragmentation.

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_REQ_DATA_Ttag
{
  uint32_t ulNodeId;
  uint32_t ulIndex;
} ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	16	Packet Data Length in bytes
ulCmd	UINT32	0x650018	ETHERCAT_MASTER_CMD_GET_OBJECTDESC_REQ
Data			
ulNodeId	UINT32		During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
ulIndex	UINT32	0 … 0xFFFF	Index of object

Table 90: ETHERCAT_MASTER_CMD_GET_OBJECTDESC_REQ – Get object description request (Legacy)

Packet structure reference

```
#define ETHERCAT_MASTER_COE_GET_OBJECTDESC_CNF_LEN_ON_ERROR (8)
#define ETHERCAT_MASTER_COE_GET_OBJECTDESC_NAME_LEN (HIL_MAX_DATA_SIZE -
(sizeof(uint32_t) * 7))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_CNF_DATA_Ttag
{
  uint32_t ulNodeId;
  uint32_t ulIndex;/* Index in the object dictionary */
  uint32_t ulDataType; /* Data type of the object */
  uint32_t ulObjCode; /* Object code */
  uint32_t ulObjCategory; /* Object category */
  uint32_t ulMaxNumSubIndex; /* Maximum sub index number */
  uint32_t ulObNameLen; /* Length of the object name */
  uint8_t abObjName[ETHERCAT_MASTER_COE_GET_OBJECTDESC_NAME_LEN]; /* Object name (not
NULL terminated!) */
} ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_CNF_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_OBJECTDESC_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	28 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x650019	ETHERCAT_MASTER_CMD_GET_OBJECTDESC_CNF
Data			
ulNodeId	UINT32		Value from request
ulIndex	UINT32	0 … 0xFFFF	Value from request
ulDataType	UINT32	0 … 0x0FFF	Data type of object
ulObjCode	UINT32		Object code of object
ulObjCategory	UINT32		Object category
ulMaxNumSubIndex	UINT32	0 … 255	Maximum number of subindices
ulObNameLen	UINT32		Length of name of object
abObjName[n]	UINT8[]		Name of object Actual length is given in ulObNameLen

Table 91: ETHERCAT_MASTER_CMD_GET_OBJECTDESC_CNF – Get object description confirmation (Legacy)

4.7.5.3 Get entry description (EntryDesc) (Legacy)

Note: This packet is provided for applications migrating from ECM V3.X. This packet does not support fragmentation.

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_REQ_DATA_Ttag
{
  uint32_t ulNodeId;
  uint32_t ulIndex;
  uint32_t ulSubIndex;
  uint32_t ulAccessBitMask;
} ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	16	Packet Data Length in bytes
ulCmd	UINT32	0x65001A	`ETHERCAT_MASTER_CMD_GET_ENTRYDESC_REQ`
Data			
ulNodeId	UINT32		During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
ulIndex	UINT32	0 … 0xFFFF	Index of subobject
ulSubIndex	UINT32	0 … 0xFF	Subindex of subobject
ulAccessBitMask	UINT32		Access bit mask See *Table 93: Parameter `ulAccessBitMask`*

Table 92: `ETHERCAT_MASTER_CMD_GET_ENTRYDESC_REQ` – Get entry description request (Legacy)

Meaning of `ulAccessBitMask`

Bits	Name (Bit mask)	Description
31 ... 7	Reserved	Reserved for future use
6	ETHERCAT_MASTER_COE_ENTRY_MAXVALUE (0x00000040)	Maximum value
5	ETHERCAT_MASTER_COE_ENTRY_MINVALUE (0x00000020)	Minimum value
4	ETHERCAT_MASTER_COE_ENTRY_DEFAULTVALUE (0x00000010)	Default value
3	ETHERCAT_MASTER_COE_ENTRY_UNITTYPE (0x00000008)	Unit
2	ETHERCAT_MASTER_COE_ENTRY_PDOMAPPING (0x00000004)	Information whether the object can be mapped to PDO It is always requested. Bit definition is provided for legacy use.
1	ETHERCAT_MASTER_COE_ENTRY_OBJCATEGORY (0x00000002)	Object category It is always requested. Bit definition is provided for legacy use.
0	ETHERCAT_MASTER_COE_ENTRY_OBJACCESS (0x00000001)	Object access rights It is always requested. Bit definition is provided for legacy use.

Table 93: Parameter `ulAccessBitMask`

Packet structure reference

```
#define ETHERCAT_MASTER_COE_GET_ENTRYDESC_MAX_DATA (HIL_MAX_DATA_SIZE - (sizeof(uint32_t)
* 11))
#define ETHERCAT_MASTER_COE_GET_ENTRYDESC_CNF_LEN_ON_ERROR (12)

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_CNF_DATA_Ttag
{
  uint32_t ulNodeId;
  uint32_t ulIndex;/* Index in the object dictionary */
  uint32_t ulSubIndex;
  uint32_t ulValueInfo; /* Bit mask to define which information is available */
  uint32_t ulDataType; /* Object data type */
  uint32_t ulBitLen; /* Object size (number of bits) */
  uint32_t ulObAccess; /* Access rights */
  uint32_t fRxPdoMapping; /* Is the object PDO-mappable? */
  uint32_t fTxPdoMapping; /* Can the PDO be changed */
  uint32_t ulUnitType; /* Unit*/
  uint32_t ulDataLen; /* Size of the remaining object data */
  uint8_t  abObjData[ETHERCAT_MASTER_COE_GET_ENTRYDESC_MAX_DATA]; /* Remaining object
data (see EtherCAT specification) */
} ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_CNF_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_ENTRYDESC_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	44 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x65001B	ETHERCAT_MASTER_CMD_GET_ENTRYDESC_CNF
Data			
ulNodeId	UINT32		Value from request
ulIndex	UINT32		Value from request
ulSubIndex	UINT32		Value from request
ulValueInfo	UINT32		Bits 0 to 2 are always set for legacy applications
ulDataType	UINT32	0 … 0x0FFF	Data type of sub index
ulBitLen	UINT32	0 … 0xFFFF	Bit length of subindex If value is 0xFFFF, the actual bit length has to be determined by read
ulObAccess	UINT32		Access bit mask See *Table 93: Parameter ulAccessBitMask*
fRxPdoMapping	BOOL		RxPDO Mapping
fTxPdoMapping	BOOL		TxPDO Mapping
ulUnitType	UINT32		Unit Type according ETG.1004
ulDataLen	UINT32		Length of data in abObjData
abObjData[n]	UINT8[]		Remaining object data Actual length is given by ulDataLen

Table 94: ETHERCAT_MASTER_CMD_GET_ENTRYDESC_CNF – Get entry description confirmation (Legacy)

Bit mask for ulAccessBitMask and ulValueInfo

Bit No.	Definition / description
6	ETHERCAT_MASTER_COE_ENTRY_MAXVALUE In request, this bit defines whether Maximum Value is to be requested by master. In confirmation, this bit defines whether Maximum Value is available and has been requested.
5	ETHERCAT_MASTER_COE_ENTRY_MINVALUE In request, this bit defines whether Minimum Value is to be requested by master. In confirmation, this bit defines whether Minimum Value is available and has been requested.
4	ETHERCAT_MASTER_COE_ENTRY_DEFAULTVALUE In request, this bit defines whether Default Value is to be requested by master. In confirmation, this bit defines whether Default Value is available and has been requested.
3	ETHERCAT_MASTER_COE_ENTRY_UNITTYPE In request, this bit defines whether Unit type is to be requested by master. In confirmation, this bit defines whether unit type is available and has been requested.
2	ETHERCAT_MASTER_COE_ENTRY_PDOMAPPING PDO Mapping flags are always requested and therefore this bit is always set to 1 in response.
1	ETHERCAT_MASTER_COE_ENTRY_OBJCATEGORY Object category is always requested and therefore this bit is always set to 1 in response.
0	ETHERCAT_MASTER_COE_ENTRY_OBJACCESS Object access is always requested and therefore this bit is always set to 1 in response.

Table 95: Meaning of ulAccessBitMask and ulValueInfo

4.8 FoE services

FoE services are implemented starting with V4.3.

4.8.1 Slave state accessibility

FoE access is possible in the following slave states if supported by slave:

- BOOT (if supported by slave)
- PREOP
- SAFEOP
- OP

Slaves may limit access depending on slave state to certain files.

4.8.2 Fragmentation of write file (FoE)

Flow charts are presented in section *FoE fragmentation flowcharts* (page 118).

When the data fits into a single fragment, the following sequence is used:

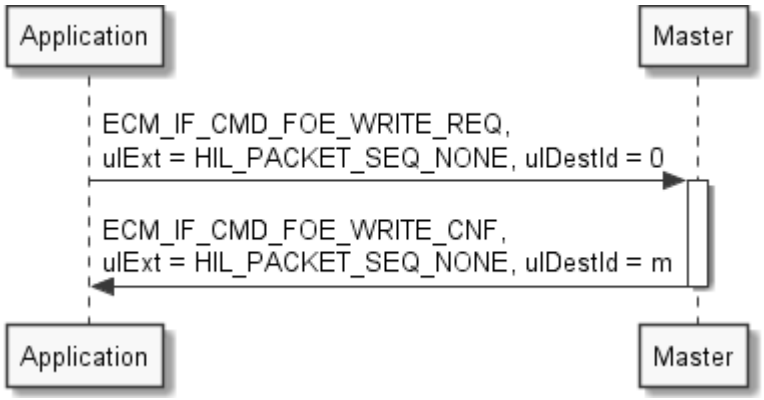


Figure 24: Single fragment handling of write file service

When two or more fragments are required for the transfer, the following sequence diagrams apply:

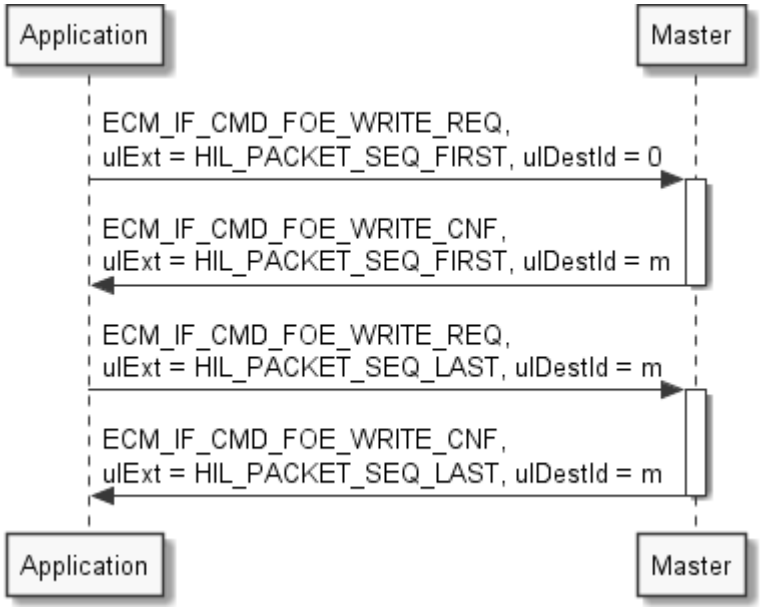


Figure 25: Two Fragment handling of write file service

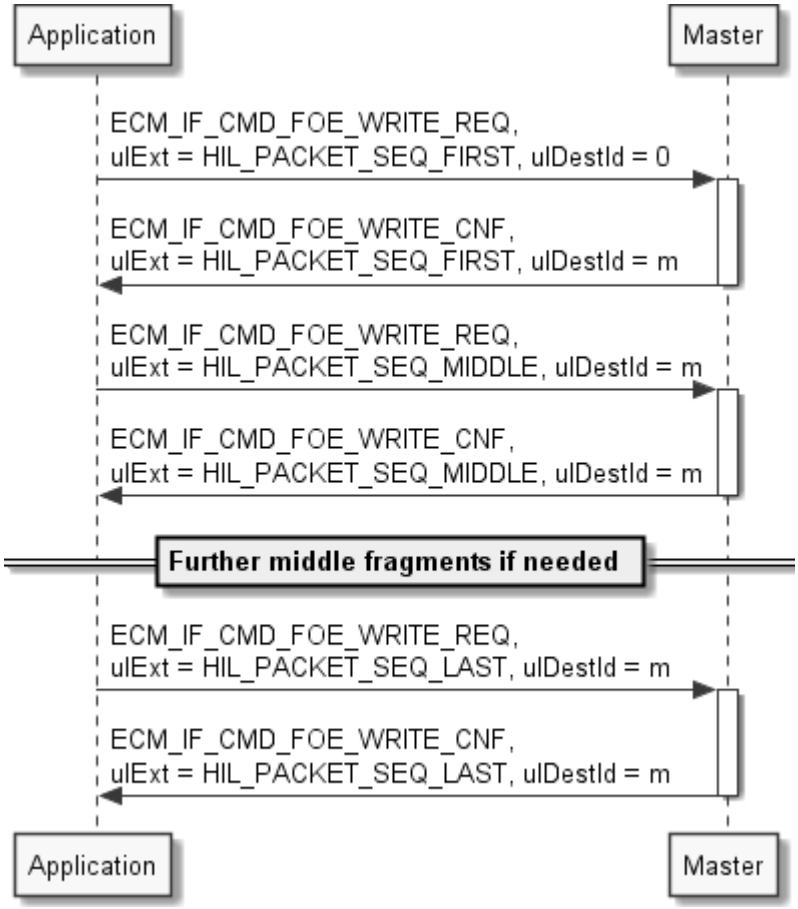


Figure 26: Multiple fragment handling of write file service

4.8.3 Fragmentation of read IDN (SoE)

Flow charts are presented in section *FoE fragmentation flowcharts* (page 118).

When the data fit into a single fragment, the following sequence is used:

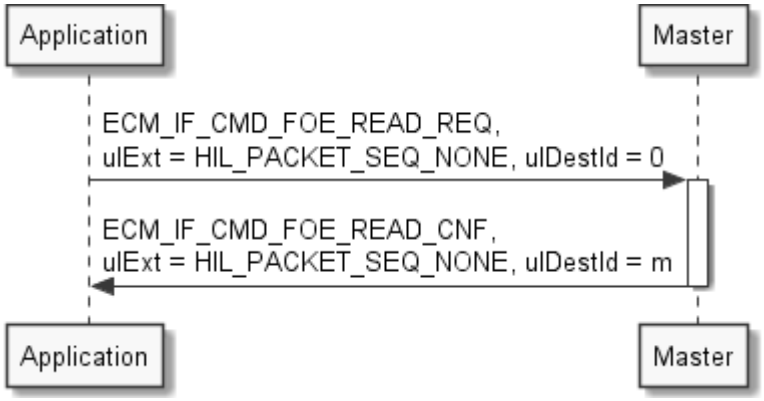


Figure 27: Single fragment handling of read file service

When two or more fragments are required for the transfer, the following sequence diagrams apply:

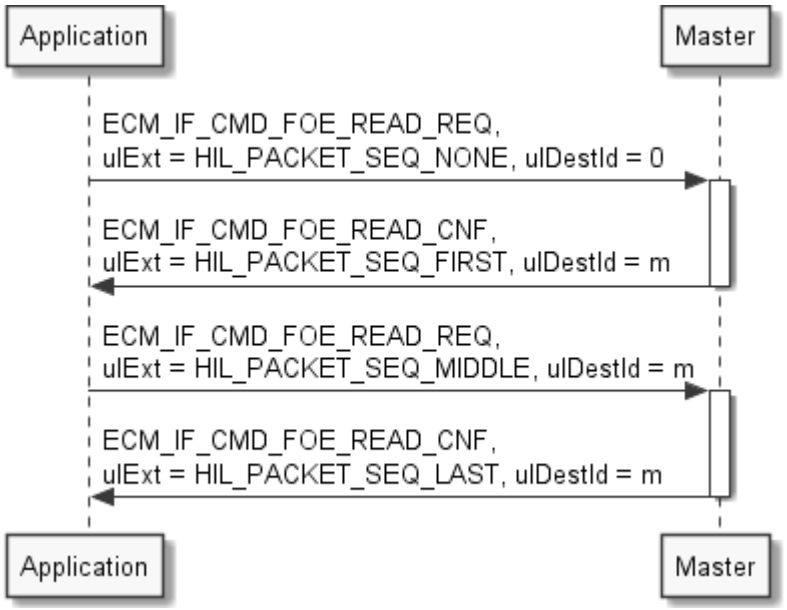


Figure 28: Two fragment handling of read file service

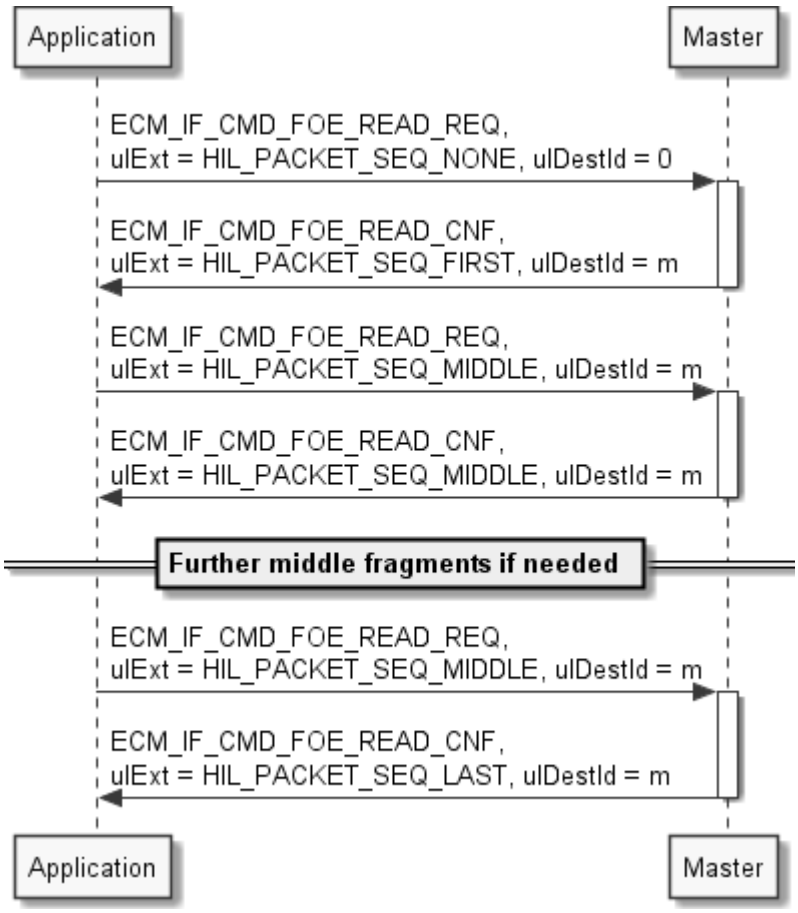


Figure 29: Multiple fragment handling of read file service

4.8.4 Packets

4.8.4.1 Write file (FoE)

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

`ulDestId` has to be handled as follows:

- First fragment has `ulDestId` == 0
- Stack returns first fragment confirmation with `ulDestId` != 0
- This `ulDestId` has to be provided to all subsequent fragments

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_WRITE_REQ_DATA_FIRST_Ttag
{
  /* structure for first fragment */
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_FOE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_FOE_TRANSPORT_TYPE_AOE is selected */
  uint32_t ulTotalBytes; /* has to be set to summed length of all abData of all fragments
*/
  uint32_t ulTimeoutMs;
  uint32_t ulPassword;
  uint32_t ulFileNameBytes; /* number of bytes used for file name including its NUL
terminator */
  uint8_t abData[1024]; /* [0 - (ulFileNameBytes - 1)] is a NUL-terminated filename */
} ECM_IF_FOE_WRITE_REQ_DATA_FIRST_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_WRITE_REQ_DATA_SEG_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_FOE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_FOE_TRANSPORT_TYPE_AOE is selected */
  uint32_t ulTotalBytes; /* has to be set to summed length of all abData of all fragments
*/
  uint32_t ulTimeoutMs;
  uint8_t abData[1024];
} ECM_IF_FOE_WRITE_REQ_DATA_SEG_T;

typedef __HIL_PACKED_PRE union __HIL_PACKED_POST ECM_IF_FOE_WRITE_REQ_DATA_Ttag
{
  ECM_IF_FOE_WRITE_REQ_DATA_FIRST_T tFirst;
  ECM_IF_FOE_WRITE_REQ_DATA_SEG_T tSeg;
} ECM_IF_FOE_WRITE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_WRITE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_FOE_WRITE_REQ_DATA_T tData;
} ECM_IF_FOE_WRITE_REQ_T;
```

Packet description (First segment)

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	22 + n	Packet Data Length in bytes
ulCmd	UINT32	0x9900	ECM_IF_CMD_FOE_WRITE_REQ
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: FoE transport (ECM_IF_FOE_TRANSPORT_FOE), 1: AoE transport (ECM_IF_FOE_TRANSPORT_AOE)
usAoEPort	UINT16	0 … 65535	AoEPort (only used if usTransportType = 1)
ulTotalBytes	UINT32	0 ... 2^{32}-1	Summed length of all abData of all fragments
ulTimeoutMs	UINT32		Timeout in ms
ulPassword	UINT32	0 ... 2^{32}-1	FoE password to be used
ulFileNameBytes	UINT32		Number of bytes used for file name including its NUL terminator
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as ulLen – 22 The first segment contains the NUL-terminated file name. Its length is given by ulFileNameBytes.

Table 96: ECM_IF_CMD_FOE_WRITE_REQ – Write file request (FoE, First segment)

Packet description (Following segments)

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	18 + n	Packet Data Length in bytes
ulCmd	UINT32	0x9900	ECM_IF_CMD_FOE_WRITE_REQ
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: FoE transport (ECM_IF_FOE_TRANSPORT_FOE), 1: AoE transport (ECM_IF_FOE_TRANSPORT_AOE)
usAoEPort	UINT16	0 … 65535	AoEPort (only used if usTransportType = 1)
ulTotalBytes	UINT32	0 ... 2^{32}-1	Summed length of all abData of all fragments
ulTimeoutMs	UINT32		Timeout in ms
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as ulLen - 18

Table 97: ECM_IF_CMD_FOE_WRITE_REQ – Write file request (FoE, Following segment)

Timeout value ulTimeoutMs

It is recommended to use at least 1000 ms as value. However, slaves exist that need higher timeout value due to their way of functioning.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_WRITE_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType;
  uint16_t usAoEPort;
  uint32_t ulTotalBytes;
  uint32_t ulTimeoutMs;
  uint8_t abErrorText[ECM_IF_FOE_MAX_ERROR_TEXT_BYTE_LEN]; /* NUL-terminated error text,
valid when ulSta != 0 */
} ECM_IF_FOE_WRITE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_WRITE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_FOE_WRITE_CNF_DATA_T tData;
} ECM_IF_FOE_WRITE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	14 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9901	ECM_IF_CMD_FOE_WRITE_CNF
Data			
usStationAddress	UINT16	Valid address	Value from request
usTransportType	UINT16	0,1	Value from request
usAoEPort	UINT16		Value from request
ulTotalBytes	UINT32		Value from request
ulTimeoutMs	UINT32		Value from request
abErrorText	UINT8[n]		Optionally a NUL-terminated string of n bytes produced by slave when ulSta != 0

Table 98: ECM_IF_CMD_FOE_WRITE_CNF – Write file confirmation

4.8.4.2 Read file (FoE)

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

`ulDestId` has to be handled as follows:

- First fragment has `ulDestId` == 0
- Stack returns first fragment confirmation with `ulDestId` != 0
- This `ulDestId` has to be provided to all subsequent fragments

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_READ_REQ_DATA_FIRST_Ttag
{
  /* structure for first fragment */
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_FOE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_FOE_TRANSPORT_TYPE_AOE is selected */
  uint32_t ulTimeoutMs;
  uint32_t ulMaxTotalBytes;
  uint32_t ulPassword;
  uint8_t abFileName[1024]; /* byte size defined by tHead.ulLen -
offsetof(ECM_IF_FOE_READ_REQ_DATA_FIRST_T, tData) */
} ECM_IF_FOE_READ_REQ_DATA_FIRST_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_READ_REQ_DATA_SEG_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_FOE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_FOE_TRANSPORT_TYPE_AOE is selected */
  uint32_t ulTimeoutMs;
  uint32_t ulMaxTotalBytes;
} ECM_IF_FOE_READ_REQ_DATA_SEG_T;

typedef __HIL_PACKED_PRE union __HIL_PACKED_POST ECM_IF_FOE_READ_REQ_DATA_Ttag
{
  ECM_IF_FOE_READ_REQ_DATA_FIRST_T tFirst;
  ECM_IF_FOE_READ_REQ_DATA_SEG_T tSeg;
}ECM_IF_FOE_READ_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_READ_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_FOE_READ_REQ_DATA_T tData;
} ECM_IF_FOE_READ_REQ_T;
```

Packet description (First segment)

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	18 + n	Packet Data Length in bytes
ulCmd	UINT32	0x9902	ECM_IF_CMD_FOE_READ_REQ
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: FoE transport (ECM_IF_FOE_TRANSPORT_FOE), 1: AoE transport (ECM_IF_FOE_TRANSPORT_AOE)
usAoEPort	UINT16	0 … 65535	AoEPort (only used if usTransportType = 1)
ulTimeoutMs	UINT32		Timeout in ms
ulMaxTotalBytes	UINT32	0 ... 2^{32}-1	Maximum total data bytes to be requested ulMaxTotalBytes is used to limit the size of a file. A received file that is larger than ulMaxTotalBytes will trigger a transfer error.
ulPassword	UINT32	0 ... 2^{32}-1	See table below
abFileName	UINT8[n]	0 ... 2^{32}-1	Name of file to be requested (NUL-terminated)

Table 99: ECM_IF_CMD_FOE_READ_REQ – Read file request (FoE, First segment)

Packet description (Following segments)

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	18	Packet Data Length in bytes
ulCmd	UINT32	0x9902	ECM_IF_CMD_FOE_READ_REQ
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: FoE transport (ECM_IF_FOE_TRANSPORT_FOE), 1: AoE transport (ECM_IF_FOE_TRANSPORT_AOE)
usAoEPort	UINT16	0 … 65535	AoEPort (only used if usTransportType = 1)
ulTimeoutMs	UINT32		Timeout in ms
ulMaxTotalBytes	UINT32	0 ... 2^{32}-1	Maximum total data bytes to be requested ulMaxTotalBytes is used to limit the size of a file. A received file that is larger than ulMaxTotalBytes will trigger a transfer error.

Table 100: ECM_IF_CMD_FOE_READ_REQ – Read file request (FoE, Following segments)

Timeout value ulTimeoutMs

It is recommended to use at least 1000 ms as value. However, slaves exist that need higher timeout value due to their way of functioning.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_READ_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType;
  uint16_t usAoEPort;
  uint32_t ulTimeoutMs;
  uint32_t ulTotalBytes; /* summed length of all abData confirmation fragments */
  uint8_t abData[1024]; /* actual length is given by ulLen -
offsetof(ECM_IF_FOE_READ_CNF_DATA_T, abData) */
  /* in case of ulSta != 0, abData contains a NUL-terminated error string */
} ECM_IF_FOE_READ_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_FOE_READ_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_FOE_READ_CNF_DATA_T tData;
} ECM_IF_FOE_READ_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	14 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9903	ECM_IF_CMD_FOE_READ_CNF
Data			
usStationAddress	UINT16	Valid address	Value from request
usTransportType	UINT16	0, 1	Value from request
usAoEPort	UINT16		Value from request
ulTimeoutMs	UINT32		Timeout in ms
ulTotalBytes	UINT32		Summed length of all abData of all fragments
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as ulLen – 14 In case of ulSta != 0, it optionally contains a NUL-terminated error message.

Table 101: ECM_IF_CMD_FOE_READ_CNF – Read file confirmation (FoE)

4.8.5 FoE fragmentation flowcharts

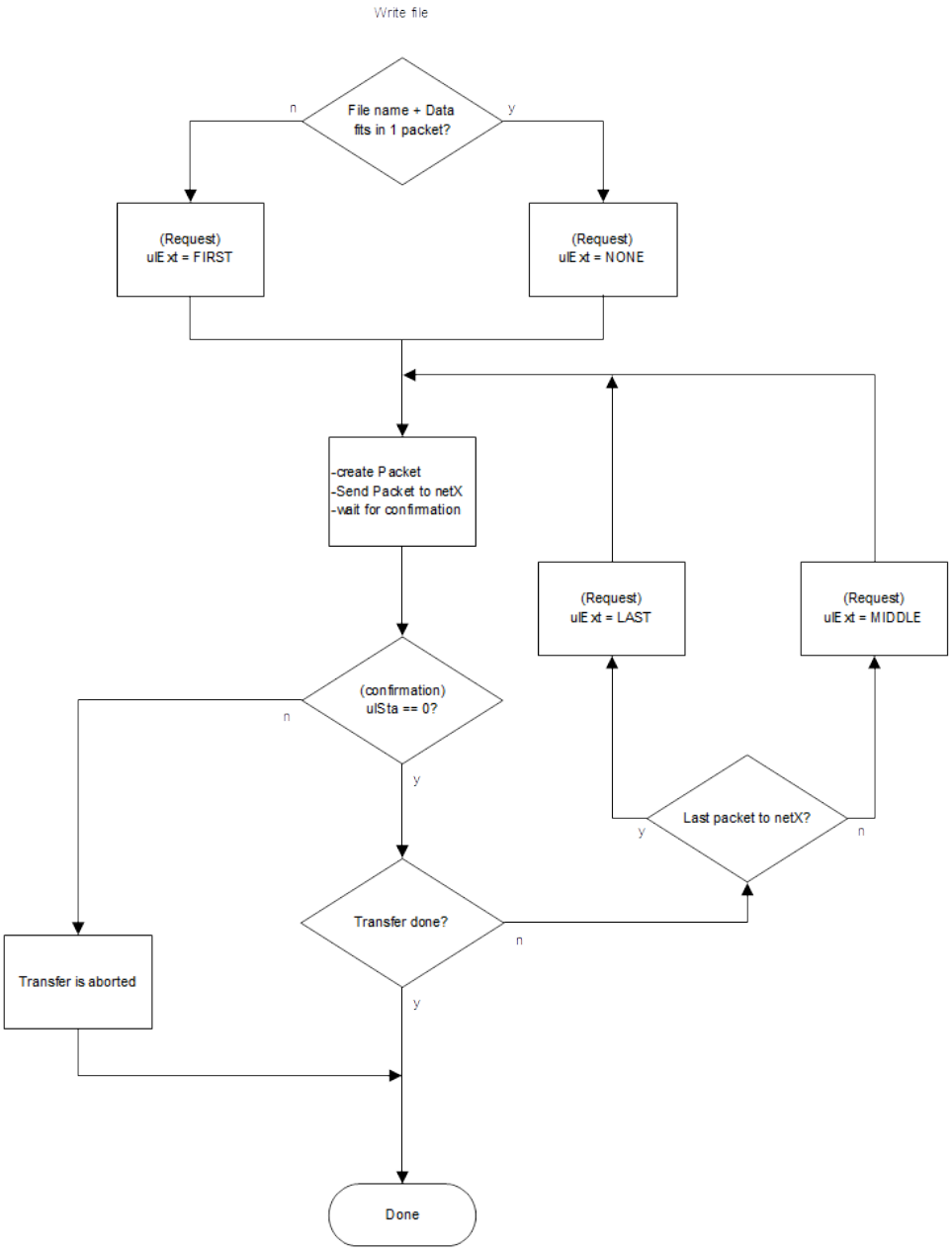


Figure 30: Flowchart for write file service fragmentation

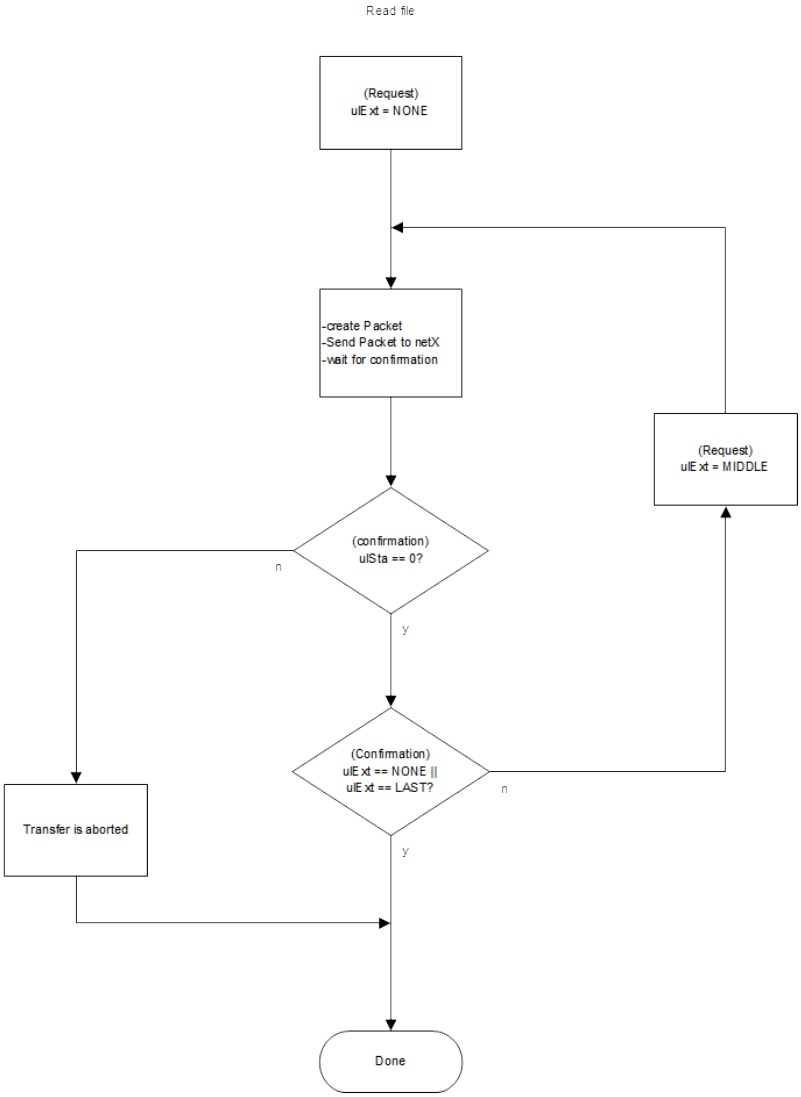


Figure 31: Flowchart for read file fragmentation

4.9 SoE services

4.9.1 Slave state accessibility

SoE access is possible in the following slave states if supported by slave:

- PREOP
- SAFEOP
- OP

Slaves may limit access depending on slave state to certain objects.

4.9.2 Fragmentation of write IDN (SoE)

Flow charts are presented in section *SoE fragmentation flowcharts* (page 130).

When the data fits into a single fragment, the following sequence is used:

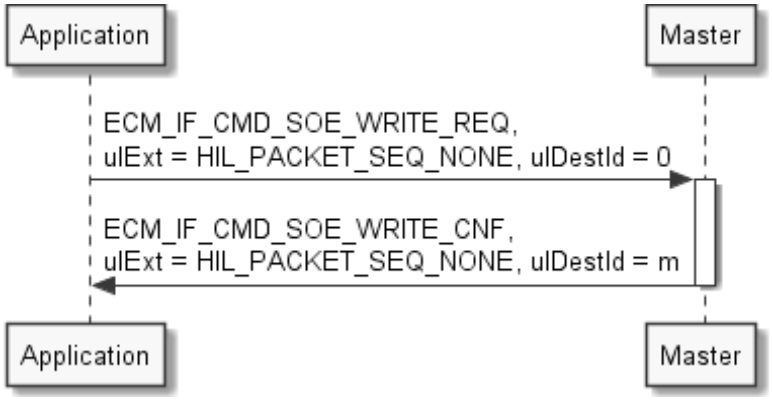


Figure 32: Single fragment handling of write IDN service

When two or more fragments are required for the transfer, the following sequence diagrams apply:

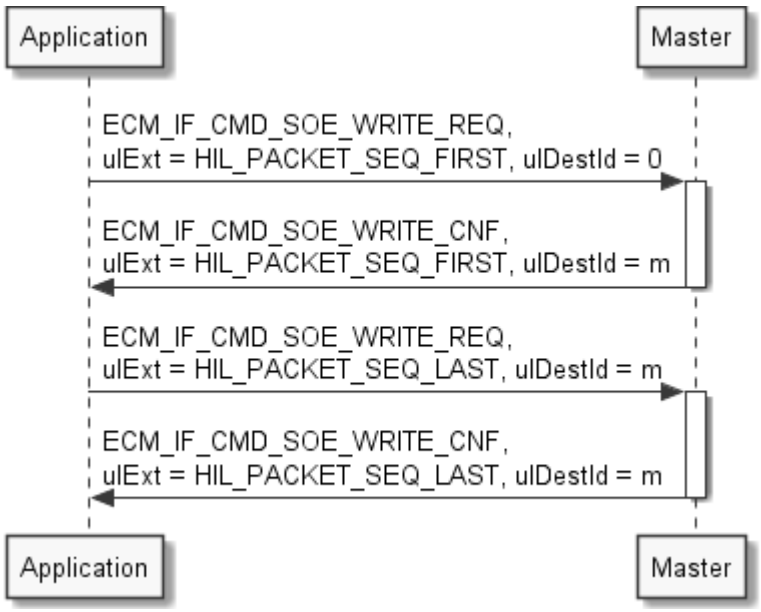


Figure 33: Two Fragment handling of write IDN service

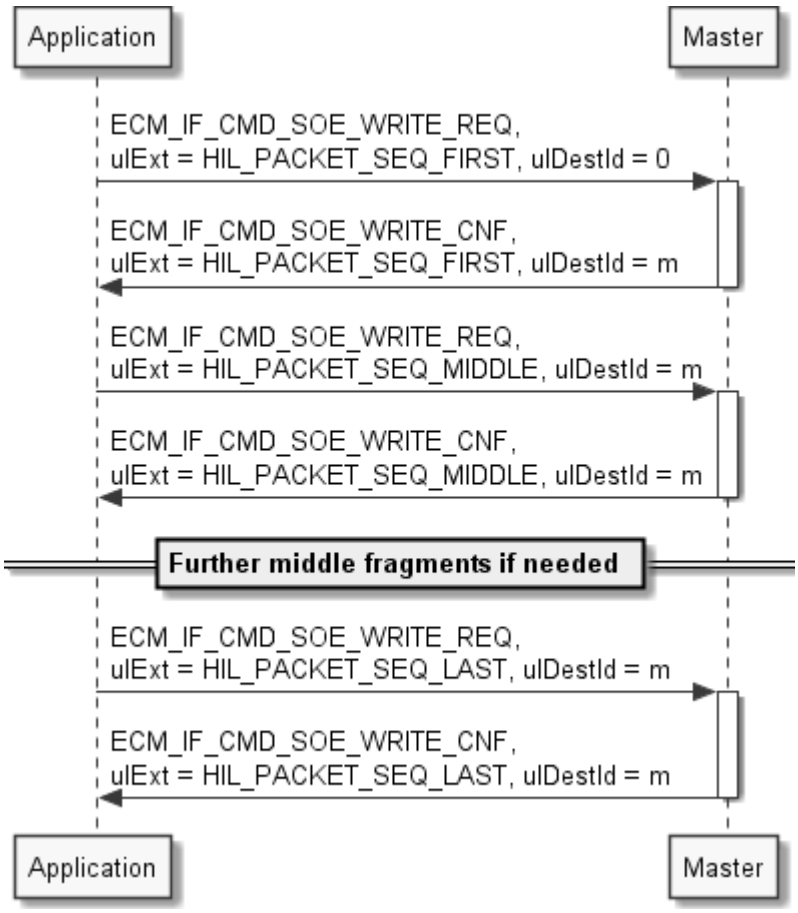


Figure 34: Multiple fragment handling of write IDN service

4.9.3 Fragmentation of read IDN (SoE)

Flow charts are presented in section *SoE fragmentation flowcharts* (page 130).

When the data fit into a single fragment, the following sequence is used:

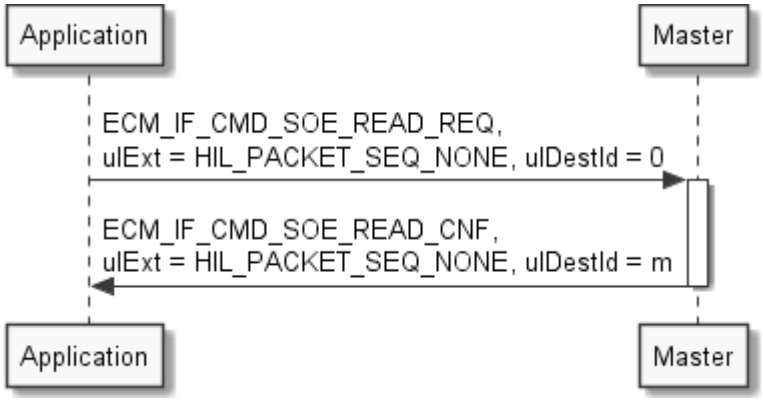


Figure 35: Single fragment handling of read IDN service

When two or more fragments are required for the transfer, the following sequence diagrams apply:

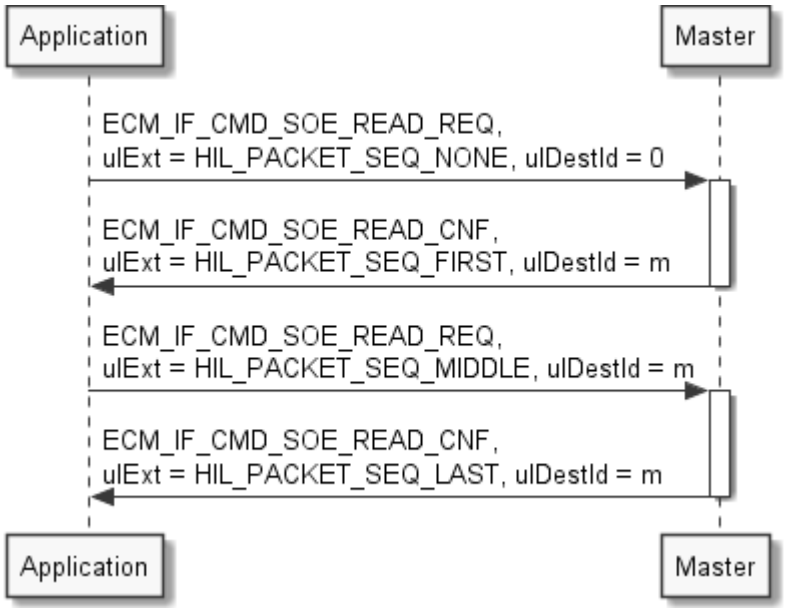


Figure 36: Two fragment handling of read IDN service

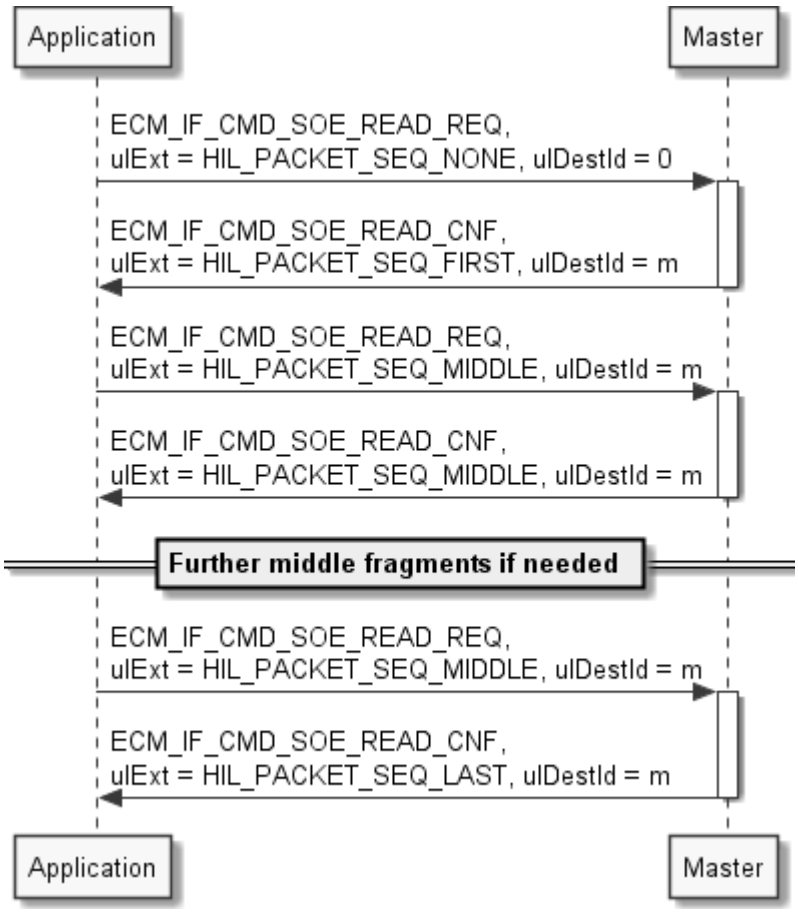


Figure 37: Multiple fragment handling of read IDN service

4.9.4 Packets

4.9.4.1 Write IDN (SoE)

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

`ulDestId` has to be handled as follows:

- First fragment has `ulDestId` == 0
- Stack returns first fragment confirmation with `ulDestId` != 0
- This `ulDestId` has to be provided to all subsequent fragments

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SOE_WRITE_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_SOE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_SOE_TRANSPORT_TYPE_AOE is selected */
  uint16_t usIDN;
  uint32_t ulTotalBytes; /* has to be set to summed length of all abData of all fragments
*/
  uint32_t ulTimeoutMs;
  uint8_t bDriveNo;
  uint8_t bElementFlags; /* see ECM_IF_SOE_ELEMENT_FLAGS_E */
  uint8_t abData[1024];
} ECM_IF_SOE_WRITE_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SOE_WRITE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SOE_WRITE_REQ_DATA_T tData;
} ECM_IF_SOE_WRITE_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	18 + n	Packet Data Length in bytes
ulCmd	UINT32	0x9B00	`ECM_IF_CMD_SOE_WRITE_REQ`
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: SoE transport (`ECM_IF_SOE_TRANSPORT_SOE`), 1: AoE transport (`ECM_IF_SOE_TRANSPORT_AOE`)
usAoEPort	UINT16	0 … 65535	AoEPort (only used if `usTransportType` = 1)
usIDN	UINT16	Valid IDN	IDN number
ulTotalBytes	UINT32	0 ... 2^{32}-1	Summed length of all `abData` of all fragments
ulTimeoutMs	UINT32		Timeout in ms
bDriveNo	UINT8	0 … 7	Drive number
bElementFlags	UINT8	0 … 255	See *Table 103: Meaning of `bElementFlags`* below
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as `ulLen - 18`

Table 102: `ECM_IF_CMD_SOE_WRITE_REQ` – Write IDN request

Timeout value ulTimeoutMs

It is recommended to use at least 1000 ms as value. However, slaves exist that need higher timeout value due to their way of functioning.

Bit mask for `bElementFlags`

Bit No.	Definition / description
7	`RESERVED` Reserved, set to 0.
6	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_VALUE` Value of IDN is requested
5	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_MAX` Maximum value of IDN is requested
4	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_MIN` Minimum value of IDN is requested
3	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_UNIT` Unit string of IDN is requested
2	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_ATTRIBUTE` Attribute of IDN is requested
1	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_NAME` Name string of IDN is requested
0	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_DATASTATE` Data State of IDN is requested

Table 103: Meaning of `bElementFlags`

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SOE_WRITE_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType;
  uint16_t usAoEPort;
  uint16_t usIDN;
  uint32_t ulTotalBytes;
  uint32_t ulTimeoutMs;
  uint8_t bDriveNo;
  uint8_t bElementFlags;
} ECM_IF_SOE_WRITE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SOE_WRITE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SOE_WRITE_CNF_DATA_T tData;
} ECM_IF_SOE_WRITE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	18	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9B01	ECM_IF_CMD_SOE_WRITE_CNF
Data			
usStationAddress	UINT16	Valid address	Value from request
usTransportType	UINT16	0,1	Value from request
usAoEPort	UINT16		Value from request
usIDN	UINT16	Valid IDN	Value from request
ulTotalBytes	UINT32		Value from request
ulTimeoutMs	UINT32		Value from request
bDriveNo	UINT8	0 … 7	Value from request
bElementFlags	UINT8	0 … 255	Value from request

Table 104: ECM_IF_CMD_SOE_WRITE_CNF – Write IDN confirmation

4.9.4.2 Read IDN (SoE)

The following addressing schemes are used:

- During generic bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

`ulDestId` has to be handled as follows:

- First fragment has `ulDestId` == 0
- Stack returns first fragment confirmation with `ulDestId` != 0
- This `ulDestId` has to be provided to all subsequent fragments

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SOE_READ_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType; /* see ECM_IF_SOE_TRANSPORT_TYPE_E */
  uint16_t usAoEPort; /* used when ECM_IF_SOE_TRANSPORT_TYPE_AOE is selected */
  uint16_t usIDN;
  uint32_t ulTimeoutMs;
  uint8_t bDriveNo;
  uint8_t bElementFlags; /* see ECM_IF_SOE_ELEMENT_FLAGS_E */
  uint32_t ulMaxTotalBytes;
} ECM_IF_SOE_READ_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SOE_READ_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SOE_READ_REQ_DATA_T tData;
} ECM_IF_SOE_READ_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	18	Packet Data Length in bytes
ulCmd	UINT32	0x9B02	`ECM_IF_CMD_SOE_READ_REQ`
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usTransportType	UINT16	0,1	Transport type 0: SoE transport (`ECM_IF_SOE_TRANSPORT_SOE`), 1: AoE transport (`ECM_IF_SOE_TRANSPORT_AOE`)
usAoEPort	UINT16	0 … 65535	AoEPort (only used if `usTransportType` = 1)
usIDN	UINT16	Valid IDN	IDN number
ulTimeoutMs	UINT32		Timeout in ms
bDriveNo	UINT8	0 … 7	Drive number
bElementFlags	UINT8	0 … 255	See table below
ulMaxTotalBytes	UINT32	0 ... 2^{32}-1	Maximum total data bytes to be requested

Table 105: `ECM_IF_CMD_SOE_READ_REQ` – Read IDN request

Timeout value ulTimeoutMs

It is recommended to use at least 1000 ms as value. However, slaves exist that need higher timeout value due to their way of functioning.

Bit mask for `bElementFlags`

Bit No.	Definition / description
7	`RESERVED` Reserved, set to 0.
6	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_VALUE` Value of IDN is requested
5	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_MAX` Maximum value of IDN is requested
4	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_MIN` Minimum value of IDN is requested
3	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_UNIT` Unit string of IDN is requested
2	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_ATTRIBUTE` Attribute of IDN is requested
1	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_NAME` Name string of IDN is requested
0	`MSK_ECM_IF_SOE_ELEMENT_FLAGS_DATASTATE` Data State of IDN is requested

Table 106: Meaning of `bElementFlags`

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SOE_READ_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usTransportType;
  uint16_t usAoEPort;
  uint16_t usIDN;
  uint32_t ulTimeoutMs;
  uint8_t bDriveNo;
  uint8_t bElementFlags;
  uint32_t ulTotalBytes; /* summed length of all abData confirmation fragments */
  uint8_t abData[1024]; /* actual length is given by ulLen -
offsetof(ECM_IF_SOE_READ_CNF_DATA_T, abData) */
} ECM_IF_SOE_READ_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SOE_READ_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SOE_READ_CNF_DATA_T tData;
} ECM_IF_SOE_READ_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	18 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9B03	ECM_IF_CMD_SOE_READ_CNF
Data			
usStationAddress	UINT16	Valid address	Value from request
usTransportType	UINT16	0,1	Value from request
usAoEPort	UINT16		Value from request
usIDN	UINT16	Valid IDN	Value from request
ulTimeoutMs	UINT32		Timeout in ms
bDriveNo	UINT8	0 … 7	Value from request
bElementFlags	UINT8	0 … 255	Value from request
ulTotalBytes	UINT32		Summed length of all abData of all fragments
abData[n]	UINT8[]		Data of a fragment. Actual byte length is given as ulLen - 18

Table 107: ECM_IF_CMD_SOE_READ_CNF – Read IDN confirmation

4.9.5 SoE fragmentation flowcharts

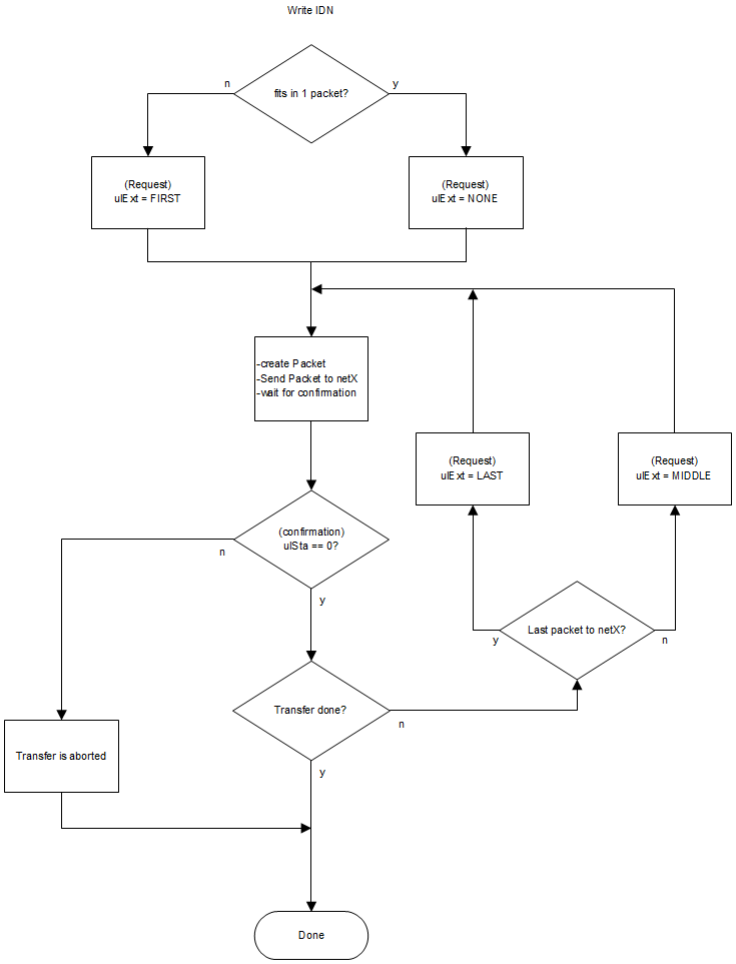


Figure 38: Flowchart for write IDN service fragmentation

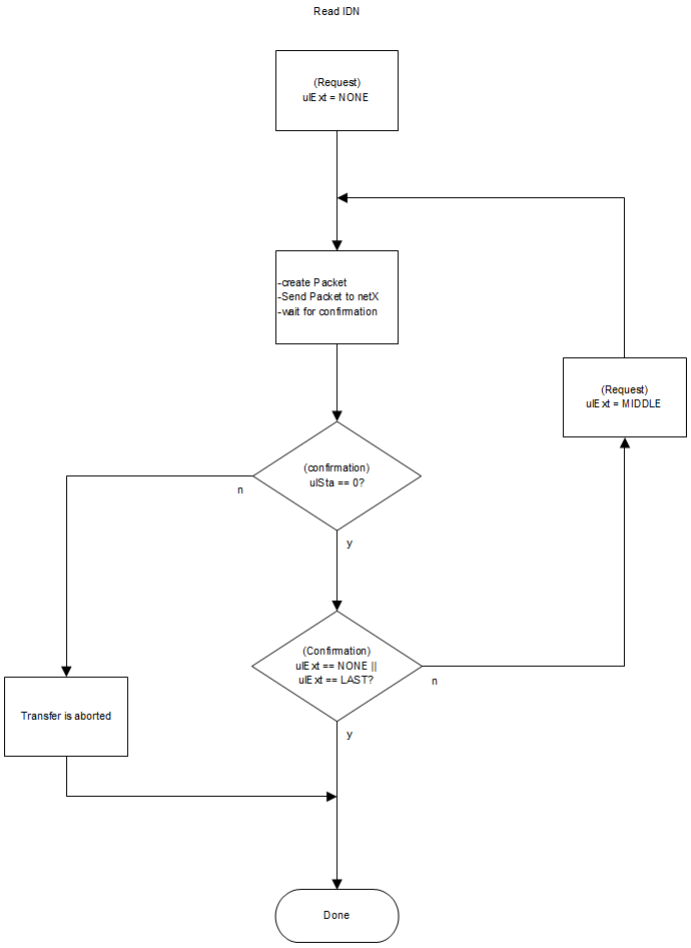


Figure 39: Flowchart for read IDN fragmentation

4.10 Distributed Clocks diagnostics

4.10.1 Packets

4.10.1.1 Get DC deviation information

This packet provides access to DC diagnostics. This includes maximum values that are recorded master-internally every ARMW/FRMW cycle.

The update rate of `ulDcSlaveBrdDeviationSignMag` depends on whether a BRD for DCSystimeDiff is configured cyclically. If it is not cyclically configured, it is requested acyclically based on calling this packet. If it is cyclically configured, it is updated every cycle which contains the actual BRD.

The maximum values can be reset with the packet *Reset DC max deviations information* (page 135).

In addition, the master resets those maximum values on resynchronization of DC slaves.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_DC_DEVIATION_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_GET_DC_DEVIATION_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E60	`ECM_IF_CMD_GET_DC_DEVIATION_REQ`

Table 108: `ECM_IF_CMD_GET_DC_DEVIATION_REQ` – Get DC deviation information request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_DC_DEVIATION_CNF_DATA_Ttag
{
  uint32_t ulDcSlaveBrdDeviationSignMag;
  uint32_t ulDcBusDeviationSignMag;
  uint32_t ulDcLocalSysTimeDeviationSignMag;
  uint32_t ulDcStatusFlags;

  /* max values */
  uint32_t ulDcSlaveBrdDeviationMaxMag;
  uint32_t ulDcBusDeviationPosMaxMag;
  uint32_t ulDcBusDeviationNegMaxMag;
  uint32_t ulDcLocalSysTimeDeviationPosMaxMag;
  uint32_t ulDcLocalSysTimeDeviationNegMaxMag;
} ECM_IF_GET_DC_DEVIATION_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_DC_DEVIATION_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_DC_DEVIATION_CNF_DATA_T tData;
} ECM_IF_GET_DC_DEVIATION_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	36	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E61	ECM_IF_CMD_GET_DC_DEVIATION_CNF
Data			
ulDcSlaveBrdDeviationSignMag	UINT32		Bitwise OR-related value of SysTimeDifference registers of all connected DC capable slaves
ulDcBusDeviationSignMag	UINT32		Deviation of Bus Cycle generator towards bus cycle For definition of Sign/Magnitude values see table below.
ulDcLocalSysTimeDeviationSignMag	UINT32		Deviation of master SysTime unit in relation to Bus Cycle For definition of Sign/Magnitude values see table below
ulDcStatusFlags	UINT32		DC Status flags See *Table 110: Meaning of ulDcStatusFlags* below
ulDcSlaveBrdDeviationMaxMag	UINT32		Max deviation seen in ulDcSlaveBrdDeviationSignMag
ulDcBusDeviationPosMaxMag	UINT32		Max. positive deviation seen in ulDcBusDeviationSignMag
ulDcBusDeviationNegMaxMag	UINT32		Max. negative deviation seen in ulDcBusDeviationSignMag
ulDcLocalSysTimeDeviationPosMaxMag	UINT32		Max. positive deviation seen in ulDcLocalSysTimeDeviationSignMag
ulDcLocalSysTimeDeviationNegMaxMag	UINT32		Max. negative deviation seen in ulDcLocalSysTimeDeviationSignMag

Table 109: ECM_IF_CMD_GET_DC_DEVIATION_CNF – Get DC deviation information confirmation

Bit mask for ulDcStatusFlags

Bit No.	Definition / description
31-7	RESERVED Reserved, set to 0.
6	ECM_IF_DC_CONTROL_STATUS_STOPPED_DC_ALL_PORTS_RX_STATUS_TIMEOUT If this bit is set, the sending of ARMW/FRMW has been stopped due to no frames being received.
5	ECM_IF_DC_CONTROL_STATUS_STOPPED_DL_STATUS_IRQ If this bit is set, the sending of ARMW/FRMW has been stopped due to a slave signaling a DL status change.
4	ECM_IF_DC_CONTROL_STATUS_STOPPED_EXPECTED_BRD_ALSTATUS_WKC_RED If this bit is set, the sending of ARMW/FRMW has been stopped due to unexpected working counter on redundancy channel.
3	ECM_IF_DC_CONTROL_STATUS_STOPPED_EXPECTED_BRD_ALSTATUS_WKC_MAIN If this bit is set, the sending of ARMW/FRMW has been stopped due to unexpected working counter on main channel.
2	ECM_IF_DC_CONTROL_STATUS_STOPPED_EXPECTED_DC_RX_STATUS_RED If this bit is set, the sending of ARMW/FRMW has been stopped due to unexpected received frame source (main,red) on redundancy port.
1	ECM_IF_DC_CONTROL_STATUS_STOPPED_EXPECTED_DC_RX_STATUS_MAIN If this bit is set, the sending of ARMW/FRMW has been stopped due to unexpected received frame source (main,red) on main port.
0	ECM_IF_DC_CONTROL_STATUS_ACTIVE If this bit is set, the sending of ARMW/FRMW status is active.

Table 110: Meaning of ulDcStatusFlags

Definition of sign/magnitude values

Bit No.	Definition / description
31	Sign of difference If set, the magnitude has to be considered as negative value.
30-0	Magnitude of difference in ns (unsigned int)

Table 111: Meaning of sign/magnitude values

4.10.1.2 Reset DC max deviations information

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_RESET_DC_MAX_DEVIATIONS_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_RESET_DC_MAX_DEVIATIONS_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E64	ECM_IF_CMD_RESET_DC_MAX_DEVIATIONS_REQ

Table 112: ECM_IF_CMD_RESET_DC_MAX_DEVIATIONS_REQ – Reset DC max deviations request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_RESET_DC_MAX_DEVIATIONS_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_RESET_DC_MAX_DEVIATIONS_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	36	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E65	ECM_IF_CMD_RESET_DC_MAX_DEVIATIONS_CNF

Table 113: ECM_IF_CMD_RESET_DC_MAX_DEVIATIONS_CNF – Reset DC max deviations confirmation

4.10.1.3 Get slave DC info

This packet allows reading the master known DC status of a given slave.

The following addressing schemes are used:

- *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_SLAVE_DC_INFO_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
} ECM_IF_GET_SLAVE_DC_INFO_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_SLAVE_DC_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_SLAVE_DC_INFO_REQ_DATA_T tData;
} ECM_IF_GET_SLAVE_DC_INFO_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	2	Packet Data Length in bytes
ulCmd	UINT32	0x9E62	ECM_IF_CMD_GET_SLAVE_DC_INFO_REQ
Data			
usStationAddress	UINT16	Valid address	Use *Fixed station address* (page 18)

Table 114: ECM_IF_CMD_GET_SLAVE_DC_INFO_REQ – Get slave DC information request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_SLAVE_DC_INFO_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usFlags;
  uint32_t ulDcSystimeDelayNs;
  uint64_t ullDcSystimeOffsetNs;
  uint64_t ullDcSyncShiftTimeNs;
  uint32_t ulDcCyc0Time;
  uint32_t ulDcCyc1Time;
  uint64_t ullRxLatchTime0Ns;
  uint32_t ulRxLatchTime1Ns;
  uint32_t ulRxLatchTime2Ns;
  uint32_t ulRxLatchTime3Ns;
  uint32_t ulPort1SumDelayNs;
  uint32_t ulPort2SumDelayNs;
  uint32_t ulPort3SumDelayNs;
  uint32_t ulTotalSumDelayNs;
  uint64_t ullDcSync0StartingDelayTimeNs;
  uint64_t ullDcResyncSystimeOffsetNs;
} ECM_IF_GET_SLAVE_DC_INFO_CNF_DATA_T;

/* usFlags */
enum ECM_IF_GET_SLAVE_DC_INFO_FLAGS_Etag
{
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_IN_TOPOLOGY = 0x0001,
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_TIME_CONFIGURED = 0x0002, /* DC synchronization
(DcSystimeOffset, DcSystimeDelay) setup done */
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_SYNC_CONFIGURED = 0x0004,
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_IS_64BIT = 0x0008,
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_IS_SUPPORTED = 0x0010,
```

```
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_ACTIVATE = 0x0100,
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_ACTIVATE_SYNC0 = 0x0200,
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_ACTIVATE_SYNC1 = 0x0400,
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_RX_TIMESTAMP_LATCH_SUPPORTED = 0x0800,

  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_PORT0_EXISTS = 0x1000,
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_PORT1_EXISTS = 0x2000,
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_PORT2_EXISTS = 0x4000,
  ECM_IF_GET_SLAVE_DC_INFO_FLAGS_PORT3_EXISTS = 0x8000
};

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_SLAVE_DC_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_SLAVE_DC_INFO_CNF_DATA_T tData;
} ECM_IF_GET_SLAVE_DC_INFO_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	76	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E63	ECM_IF_CMD_GET_SLAVE_DC_INFO_CNF
Data			
usStationAddress	UINT16		Station Address (Value from request)
usFlags	UINT16		DC Info flags See *Table 116: Meaning of usFlags* below
ulDcSystimeDelayNs	UINT32		DC Systime Delay configured to slave in ns
ullDcSystimeOffsetNs	UINT64		DC Systime Offset configured to slave in ns
ullDcSyncShiftTimeNs	UINT64		Configured DC Sync0 shift time based on bus cycle reference in ns
ulDcCyc0Time	UINT32		DC Cyc0Time as written to register
ulDcCyc1Time	UINT32		DC Cyc1Time as written to register
ullRxLatchTime0Ns	UINT64		Last latched local time of DC slave for Port 0
ulRxLatchTime1Ns	UINT32		Last latched local time of DC slave for Port 1
ulRxLatchTime2Ns	UINT32		Last latched local time of DC slave for Port 2
ulRxLatchTime3Ns	UINT32		Last latched local time of DC slave for Port 3
ulPort1SumDelayNs	UINT32		Total summed delay of all slaves connected to Port 1 of slave in ns
ulPort2SumDelayNs	UINT32		Total summed delay of all slaves connected to Port 2 of slave in ns
ulPort3SumDelayNs	UINT32		Total summed delay of all slaves connected to Port 3 of slave in ns
ulTotalSumDelayNs	UINT32		Total summed delay of all slaves connected to Port1, 2 and 3
ullDcSync0StartingDelayTimeNs	UINT64		Last used delay for starting Sync signal generation on slave

Table 115: ECM_IF_CMD_GET_SLAVE_DC_INFO_CNF – Get save DC deviation information confirmation

Bit mask for usFlags

Bit No.	Definition / description
15	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_PORT3_EXISTS Slave has a port 3
14	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_PORT2_EXISTS Slave has a port 2
13	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_PORT1_EXISTS Slave has a port 1
12	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_PORT0_EXISTS Slave has a port 0
11	Reserved
10	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_ACTIVATE_SYNC1 DC Sync1 is configured to be activated
9	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_ACTIVATE_SYNC0 DC Sync0 is configured to be activated
8	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_ACTIVATE DC Sync Unit is configured to be activated
4	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_IS_SUPPORTED Slave supports DC in general (either 32bit or 64bit)
3	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_IS_64BIT Slave supports 64bit DC
2	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_SYNC_CONFIGURED DC Sync Unit has been configured with parameters
1	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_DC_TIME_CONFIGURED DC SysTime has been configured on slave
0	ECM_IF_GET_SLAVE_DC_INFO_FLAGS_IN_TOPOLOGY Slave is connected and actively participating in topology

Table 116: Meaning of usFlags

4.10.2 Legacy packets

4.10.2.1 Get DC deviation (Legacy)

Note: This packet is provided for applications migrating from ECM V3.X.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_DC_DEVIATION_REQ_Ttag
{
  /* the request needs no data part */
  HIL_PACKET_HEADER_T                                tHead;  /** packet header. */
} ETHERCAT_MASTER_PACKET_GET_DC_DEVIATION_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x65001E	ETHERCAT_MASTER_CMD_GET_DC_DEVIATION_REQ

Table 117: ETHERCAT_MASTER_CMD_GET_DC_DEVIATION_REQ – Get DC deviation request (Legacy)

Packet structure reference

```
#define  ETHERCAT_MASTER_GET_DC_DEVIATION_NUMOFSLAVES ((HIL_MAX_DATA_SIZE -
sizeof(uint32_t))/sizeof(uint32_t))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_DC_DEVIATION_CNF_DATA_Ttag
{
  uint32_t ulBroadcastDeviation;
  uint32_t aulSlaveDeviation[ETHERCAT_MASTER_GET_DC_DEVIATION_NUMOFSLAVES] ;
} ETHERCAT_MASTER_PACKET_GET_DC_DEVIATION_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_DC_DEVIATION_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_DC_DEVIATION_CNF_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_DC_DEVIATION_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	4 + 4 * n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x65001F	ETHERCAT_MASTER_CMD_GET_DC_DEVIATION_CNF
Data			
ulBroadcastDeviation	UINT32	Bit mask	Bitwise OR-related value of SysTimeDifference registers of all connected DC capable slaves
aulSlaveDeviation[n]	UINT32[]		Slave specific deviation Table is ordered according to addressing scheme described in section *Device index* (page 19) in ascending order.

Table 118: ETHERCAT_MASTER_CMD_GET_DC_DEVIATION_CNF – Get DC deviation confirmation (Legacy)

4.11 Config readout

4.11.1 Get timing information

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_TIMING_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_GET_TIMING_INFO_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E20	ECM_IF_CMD_GET_TIMING_INFO_REQ

Table 119: ECM_IF_CMD_GET_TIMING_INFO_REQ – Get timing information request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_TIMING_INFO_CNF_DATA_Ttag
{
  uint32_t ulBusCycleTimeNs;
  uint32_t ulFrameTransmitTimeNs;
  uint32_t ulExpectedBusDelayNs;
  uint32_t ulExpectedRxEndTimeNs; /* from start of bus cycle transmission */
  uint32_t ulExpectedTxDataTimeNs; /* from start of bus cycle transmission */
} ECM_IF_GET_TIMING_INFO_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_TIMING_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_TIMING_INFO_CNF_DATA_T tData;
} ECM_IF_GET_TIMING_INFO_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	20	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E21	ECM_IF_CMD_GET_TIMING_INFO_CNF
Data			
ulBusCycleTimeNs	UINT32	Bit mask	Bitwise OR-related value of SysTimeDifference registers of all connected DC capable slaves
ulFrameTransmitTimeNs	UINT32		Slave specific deviation (in nanoseconds) Table is ordered according to addressing scheme described in section *Device index* (page 19) in ascending order.
ulExpectedBusDelayNs	UINT32		Expected transmission time through the entire bus (in nanoseconds) i.e. delay from start of transmission to start of receive
ulExpectedRxEndTimeNs	UINT32		Time from start of bus cycle transmission until completion of receiving the bus cycle back (in nanoseconds)
ulExpectedTxDataTimeNs	UINT32		Time from start of bus cycle transmission until a new data update is expected to be signaled to stack (in nanoseconds)

Table 120: ECM_IF_CMD_GET_TIMING_INFO_CNF – Get timing information Confirmation

4.11.2 Get WcState information

The WcState bit is a means to determine the current status of process data area based on the current cyclic exchange working counters.

The EtherCAT master only maps WcState bits for cyclic telegram areas that are linked with process data and contain a compare value for its validity (e.g. <Cnt> tag in ENI).

4.11.2.1 Meaning of the WcState bit

The WcState bit indicates whether a related process data area is invalid.

Value	Definition / description
0	Related process data is valid
1	Related process data is invalid

Table 121: Possible values of WcState bit

4.11.2.2 Ordering of entries in confirmation

The entries are in order of their appearance in the cyclic frames. The offsets within the entries specify the process data areas which are guarded by a specific WcState bit.

4.11.2.3 Get WcState information packet

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_WCSTATE_INFO_REQ_DATA_Ttag
{
  uint32_t ulEntriesStartOffset;
} ECM_IF_GET_WCSTATE_INFO_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_WCSTATE_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_WCSTATE_INFO_REQ_DATA_T tData;
} ECM_IF_GET_WCSTATE_INFO_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	4	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E22	ECM_IF_CMD_GET_WC_STATE_INFO_REQ
Data			
ulEntriesStartOffset	UINT32		Start offset at which the response data has to start Needed when ulTotalEntries in confirmation is larger than the current packet length The first message is always containing 0 here.

Table 122: ECM_IF_CMD_GET_WC_STATE_INFO_REQ – Get WcState information request

WcStateBit Entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WCSTATE_INFO_ENTRY_Ttag
{
  uint32_t ulWcStateBitPosition;
  uint16_t usTxImageStartByteOffset;
  uint16_t usRxImageStartByteOffset;
  uint16_t usImageByteLength;
  uint16_t usDirection;
} ECM_IF_WCSTATE_INFO_ENTRY_T;
```

WcStateBit Entry structure description

Variable name	Type	Meaning
ulWcStateBitPosition	UINT32	Bit position within Input process data image Examples: Value 8 defines WcState to be in byte 0 bit 0. Value 15 defines WcState to be in byte 0 bit 7.
usTxImageStartByteOffset	UINT16	Byte offset in output process data image Valid if Bit 0 of `usDirection` is set
usRxImageStartByteOffset	UINT16	Byte offset in input process data image Valid if Bit 1 of `usDirection` is set
usImageByteLength	UINT16	Byte length of related process data images
usDirection	UINT16	Bit 0: set if `usTxImageStartByteOffset` is valid Bit 1: set if `usRxImageStartByteOffset` is valid

Table 123: Structure `ECM_IF_WCSTATE_INFO_ENTRY_T`

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WCSTATE_INFO_ENTRY_Ttag
{
  uint32_t ulWcStateBitPosition;
  uint16_t usTxImageStartByteOffset;
  uint16_t usRxImageStartByteOffset;
  uint16_t usImageByteLength;
  uint16_t usDirection;
} ECM_IF_WCSTATE_INFO_ENTRY_T;

enum ECM_IF_WCSTATE_DIRECTION_Etag
{
  MSK_ECM_IF_WCSTATE_INFO_DIRECTION_TXDATA = 0x0001,
  MSK_ECM_IF_WCSTATE_INFO_DIRECTION_RXDATA = 0x0002,
};

#define ECM_IF_MAX_WCSTATE_INFO_ENTRIES ((HIL_MAX_DATA_SIZE - sizeof(uint32_t) * 2) /
sizeof(ECM_IF_WCSTATE_INFO_ENTRY_T))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_WCSTATE_INFO_CNF_DATA_Ttag
{
  uint32_t ulEntriesStartOffset;
  uint32_t ulTotalEntries; /* this number can be larger than the atEntries field can hold
*/

  /* actual number of entries given by (ptPck->tHead.ulLen -
offsetof(ECM_IF_GET_WCSTATE_INFO_CNF_DATA_T, atEntries)) /
sizeof(ECM_IF_WCSTATE_INFO_ENTRY_T) */
  ECM_IF_WCSTATE_INFO_ENTRY_T atEntries[ECM_IF_MAX_WCSTATE_INFO_ENTRIES];
} ECM_IF_GET_WCSTATE_INFO_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_WCSTATE_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_WCSTATE_INFO_CNF_DATA_T tData;
} ECM_IF_GET_WCSTATE_INFO_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	8 + 12 * n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E23	`ECM_IF_CMD_GET_WC_STATE_INFO_CNF`
Data			
ulEntriesStartOffset	UINT32		Mirrored value from request
ulTotalEntries	UINT32		Total number of entries available
atEntries	ECM_IF_WCSTATE_INFO_ENTRY_T		Entries describing the location of a WcState bit and what areas in DPM are guarded by it The actual number of entries is `(ulLen – 8) / 12`. If `ulTotalEntries` is larger than the result, more entries are available.

Table 124: `ECM_IF_CMD_GET_WC_STATE_INFO_CNF` – Get WcState information confirmation

4.11.3 Get cyclic command mapping

The EtherCAT master provides information about where the data is mapped and what kind of data is provided in that area.

4.11.3.1 Process data area types

The process data area types specify what type of data is provided by the area in process data image.

Value	Definition / description
0	VAL_ECM_IF_CYCLIC_CMD_DATATYPE_UNUSED No process data mapping for given direction
1	VAL_ECM_IF_CYCLIC_CMD_DATATYPE_PROCESS_DATA Process data is mapped in given direction
2	VAL_ECM_IF_CYCLIC_CMD_DATATYPE_DC_SYSTIME DC SysTime is mapped (only valid for receive direction)
3	VAL_ECM_IF_CYCLIC_CMD_DATATYPE_BRD_ALSTATUS Ored ALSTATUS of all slaves (only valid for receive direction)
4	VAL_ECM_IF_CYCLIC_CMD_DATATYPE_BRD_DC_SYSTIME_DIFF Ored DcSysTimeDifference register of all slaves (only valid for receive direction)
5	VAL_ECM_IF_CYCLIC_CMD_DATATYPE_WCSTATE_BITS Area of WcState Bits (only valid for receive direction)
6	VAL_ECM_IF_CYCLIC_CMD_DATATYPE_EXTSYNC_STATUS Area of ExtSync Status (only valid for receive direction)

Table 125: Possible values of `usTransmitType` and `usReceiveType`

4.11.3.2 Ordering of entries in confirmation

The entries are in order of their appearance in the cyclic frames.

4.11.3.3 Process data area type: DC SysTime

The process data area referenced by DC SysTime is either 4 or 8 bytes long. It is stored in the process data image in Little Endian format.

4.11.3.4 Process data area type: BRD ALStatus

The process data area referenced by BRD ALStatus is 2 byte long. It reflects the bitwise-ORed ALStatus register value of all slaves connected to the bus. It is stored in Little Endian format.

Bits	Description
15 … 6	Reserved
5	If set, the ALStatus-based Explicit device identification of a slave has been requested
4	ORed state of slaves. If set, at least one slave has switched to error state.
3	ORed state of slaves. If bit is set, at least one slave is in OP.
2	ORed state of slaves. If bit is set, at least one slave is in SAFEOP.
1	ORed state of slaves. If bit is set, at least one slave is in PREOP or BOOT.
0	ORed state of slaves. If bit is set, at least one slave is in INIT or BOOT.

Table 126: Data field definition of BRD ALStatus

4.11.3.5 Process data area type: BRD DcSysTimeDiff

The process data area referenced by BRD DcSysTimeDiff is 4 byte long. It reflects the bitwise-ORed values of all slaves supporting distributed clocks. It is stored in Little Endian format.

Bits	Description
31	Sign bit of DcSysTimeDiff value If set, the value is negative. When the actual sign bit is not of interest for the application, the application can simply mask out Bit 31 and use the magnitude value for knowing the deviation.
30 … 0	Magnitude of DcSysTimeDiff value

Table 127: Data field definition of BRD DcSysTimeDiff

4.11.3.6 Process data area type: WcState bits

The process data area referenced is containing WcState bits. The current layout of this area is provided by *Get WcState information* (page 141).

4.11.3.7 Process data area type: ExtSync status

The ExtSync Status process data is available since V4.4. It contains data about the current ExtSync status.

Structure Reference

```
typedef __PACKED_PRE struct ECM_EXT_SYNC_DIAG_CYCLIC_DATA_Ttag
{
  uint32_t ulExtSyncInfoFlags;
  uint16_t usExtSyncStationAddress;
  uint16_t usControlledStationAddress;
  uint64_t ullDcToExtTimeOffsetNs; /* internal DC timestamp (ns) + ullDcToExtTimeOffsetNs
=> external clock time (ns) */
  uint32_t ulDcExtErrorDiffNsSignMag;
  uint32_t ulExtSyncUpdateCount;
} __PACKED_POST ECM_EXT_SYNC_DIAG_CYLIC_DATA_T;
```

Structure Description

Element	Meaning
ulExtSyncInfoFlags	Status flags of ExtSync logic
usExtSyncStationAddress	Fixed station address of slave providing ExtSync data via PDO `0x10F4`.
usControlledStationAddress	Fixed station address of DC reference clock
ullDcToExtTimeOffsetNs	Time difference between this network and the other network providing the ExtSync base time. Relationship of timestamps: `DcToExtTimeOffsetNs = ExternalTimestampNs - InternalTimestampNs`
ulDcExtErrorDiffNsSignMag	Sign magnitude field to show the actual deviation between external synchronization reference time and own DcSysTime
ulExtSyncUpdateCount	Counter incrementing every time when ExtSync data is processed

Table 128: ExtSync Status data structure

Definition of ulExtSyncInfoFlags

Bits	Name (Bit mask)	Description
31	MSK_ECM_CYC_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_CONFIGURED (0x80000000)	Ext Sync device has been configured
30	MSK_ECM_CYC_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_ACTIVE (0x40000000)	The master is actively using External synchronization on device
29	MSK_ECM_CYC_EXT_SYNC_INFO_FLAGS_DC_TO_EXT_OFFSET_VALID (0x20000000)	ullDcToExtTimeOffsetNs is valid
28 … 24	reserved	Reserved
23 … 16	Used internally	Used internally
15	MSK_ECM_CYC_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_CONNECTED_AS_SLAVE (0x00008000)	Ext Sync Device is connected as slave
14 … 6	Reserved	Reserved for future use
5	MSK_ECM_CYC_EXT_SYNC_INFO_FLAGS_SYNC_MODE_MASTER (0x00000020)	The master is providing ExtSync to other network
4	MSK_ECM_CYC_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_NOT_CONNECTED (0x00000010)	ExtSync device is not connected when this bit is set
3	Reserved	Reserved
2	MSK_ECM_CYC_EXT_SYNC_INFO_FLAGS_IS_64BIT (0x00000004)	ExtSync timestamp size If not set, the timestamp has 32 bit. If set, the timestamp has 64 bit.
1	Reserved	Reserved
0	MSK_ECM_CYC_EXT_SYNC_INFO_FLAGS_SYNC_MODE_SLAVE (0x00000001)	The master is using ExtSync reference provided by other network

Table 129: Parameter ulExtSyncInfoFlags

Definition of ulDcExtErrorDiffNsSignMag

Bits	Description
31	Sign bit of value If set to 1, the magnitude specifies a negative value.
30 … 0	Actual magnitude of the difference in ns (unsigned int)

Table 130: Parameter ulDcExtErrorDiffNsSignMag

When the actual sign bit is not of interest for the application, the application can simply mask out Bit 31 and use the magnitude value for knowing the deviation.

4.11.3.8 Get cyclic command mapping packet

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_CYCLIC_CMD_MAPPING_REQ_DATA_Ttag
{
  uint32_t ulEntriesStartOffset;
} ECM_IF_GET_CYCLIC_CMD_MAPPING_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_CYCLIC_CMD_MAPPING_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_CYCLIC_CMD_MAPPING_REQ_DATA_T tData;
} ECM_IF_GET_CYCLIC_CMD_MAPPING_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	4	Packet Data Length in bytes
ulCmd	UINT32	0x9E24	`ECM_IF_CMD_GET_CYCLIC_CMD_MAPPING_REQ`
Data			
ulEntriesStartOffset	UINT32		Start offset at which the response data has to start Needed when `ulTotalEntries` in confirmation is larger than the actual packet length First message is always containing 0 here.

Table 131: `ECM_IF_CMD_GET_CYCLIC_CMD_MAPPING_REQ` – Get cyclic command mapping request

Cyclic command mapping entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_CYCLIC_CMD_MAPPING_ENTRY_Ttag
{
  uint16_t usTransmitType;
  uint16_t usReceiveType;
  uint16_t usTxImageStartByteOffset;
  uint16_t usRxImageStartByteOffset;
  uint16_t usImageByteLength;
  uint16_t usWkcCompareReceiveByteOffset; /* only valid if not set to 0xFFFF */
} ECM_IF_CYCLIC_CMD_MAPPING_ENTRY_T;

/* usTransmitType / usReceiveType */
enum ECM_IF_CYCLIC_CMD_DATATYPE_Etag
{
  VAL_ECM_IF_CYCLIC_CMD_DATATYPE_UNUSED = 0,
  VAL_ECM_IF_CYCLIC_CMD_DATATYPE_PROCESS_DATA = 1,
  VAL_ECM_IF_CYCLIC_CMD_DATATYPE_DC_SYSTIME = 2,
  VAL_ECM_IF_CYCLIC_CMD_DATATYPE_BRD_ALSTATUS = 3,
  VAL_ECM_IF_CYCLIC_CMD_DATATYPE_BRD_DC_SYSTIME_DIFF = 4,
  VAL_ECM_IF_CYCLIC_CMD_DATATYPE_WCSTATE_BITS = 5,
  VAL_ECM_IF_CYCLIC_CMD_DATATYPE_EXTSYNC_STATUS = 6
};

#define ECM_IF_MAX_CYCLIC_CMD_MAPPING_ENTRIES ((HIL_MAX_DATA_SIZE - sizeof(uint32_t) * 2)
/ sizeof(ECM_IF_CYCLIC_CMD_MAPPING_ENTRY_T))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_DATA_Ttag
{
  uint32_t ulEntriesStartOffset;
  uint32_t ulTotalEntries; /* this number can be larger than the atEntries field can hold
*/
```

```
  /* actual number of entries given by (ptPck->tHead.ulLen -
offsetof(ECM_IF_GET_WCSTATE_INFO_CNF_DATA_T, atEntries)) /
sizeof(ECM_IF_WCSTATE_INFO_ENTRY_T) */
  ECM_IF_CYCLIC_CMD_MAPPING_ENTRY_T atEntries[ECM_IF_MAX_CYCLIC_CMD_MAPPING_ENTRIES];
} ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_DATA_T tData;
} ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_T;
```

Cyclic command mapping entry structure description

Variable name	Type	Meaning
usTransmitType	UINT16	Specifies what kind of data is contained in transmit process data image. See *Table 125: Possible values of* `usTransmitType` and `usReceiveType`
usReceiveType	UINT16	Specifies what kind of data is contained in receive process data image See *Table 125: Possible values of usTransmitType and* `usReceiveType`
usTxImageStartByteOffset	UINT16	Byte offset in transmit process data image Valid if usTransmitType is not equal 0
usRxImageStartByteOffset	UINT16	Byte offset in receive process data image Valid if usTransmitType is not equal 0
usImageByteLength	UINT16	Length of process data
usWkcCompareReceiveByteOffset	UINT16	Placement of received Working Counter (WKC) in receive process data image if not set to 0xFFFF

Table 132: Structure `ECM_IF_CYCLIC_CMD_MAPPING_ENTRY_T`

Packet structure reference

```
#define ECM_IF_MAX_CYCLIC_CMD_MAPPING_ENTRIES ((HIL_MAX_DATA_SIZE - sizeof(uint32_t) * 2)
/ sizeof(ECM_IF_CYCLIC_CMD_MAPPING_ENTRY_T))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_DATA_Ttag
{
  uint32_t ulEntriesStartOffset;
  uint32_t ulTotalEntries; /* this number can be larger than the atEntries field can hold
*/

  /* actual number of entries given by (ptPck->tHead.ulLen -
offsetof(ECM_IF_GET_WCSTATE_INFO_CNF_DATA_T, atEntries)) /
sizeof(ECM_IF_WCSTATE_INFO_ENTRY_T) */
  ECM_IF_CYCLIC_CMD_MAPPING_ENTRY_T atEntries[ECM_IF_MAX_CYCLIC_CMD_MAPPING_ENTRIES];
} ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_DATA_T tData;
} ECM_IF_GET_CYCLIC_CMD_MAPPING_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	8 + 12 * n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E25	`ECM_IF_CMD_GET_CYCLIC_CMD_MAPPING_CNF`
Data			
ulEntriesStartOffset	UINT32		Mirrored value from request
ulTotalEntries	UINT32		Total number of entries available
atEntries	ECM_IF_CYCLIC_CMD_MAPPING_ENTRY_T		Entries describing the location of a process data area The actual number of entries is `(ulLen – 8) / 12`. If `ulTotalEntries` is larger than the result, more entries are available.

Table 133: `ECM_IF_CMD_GET_CYCLIC_CMD_MAPPING_CNF` – Get cyclic command mapping confirmation

4.11.4 Get cyclic slave mapping

The EtherCAT master provides information about where the slave data is mapped and placement information about related status data.

4.11.4.1 Slave mapping direction

The direction specifies in which direction the process data of a slave is flowing

Value	Definition / description
1	VAL_ECM_IF_CYCLIC_SLAVE_MAPPING_TYPE_TRANSMIT Process data is transmitted to slave from master
2	VAL_ECM_IF_CYCLIC_SLAVE_MAPPING_TYPE_RECEIVE Process data is received by master from slave

Table 134: Definition of `usDirection`

4.11.4.2 Get cyclic slave mapping packet

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_CYCLIC_SLAVE_MAPPING_REQ_DATA_Ttag
{
  uint32_t ulEntriesStartOffset;
} ECM_IF_GET_CYCLIC_SLAVE_MAPPING_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_CYCLIC_SLAVE_MAPPING_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_CYCLIC_SLAVE_MAPPING_REQ_DATA_T tData;
} ECM_IF_GET_CYCLIC_SLAVE_MAPPING_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	4	Packet Data Length in bytes
ulCmd	UINT32	0x9E26	`ECM_IF_CMD_GET_CYCLIC_SLAVE_MAPPING_REQ`
Data			
ulEntriesStartOffset	UINT32		Start offset at which the response data has to start Needed when `ulTotalEntries` in confirmation is larger than the actual packet length First message is always containing 0 here.

Table 135: `ECM_IF_CMD_GET_CYCLIC_SLAVE_MAPPING_REQ` – Get cyclic slave mapping request

Cyclic slave mapping entry structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_CYCLIC_SLAVE_MAPPING_ENTRY_Ttag
{
  uint16_t usDirection;
  uint16_t usStationAddress;
  uint16_t usWkcCompareReceiveByteOffset; /* only valid if not set to 0xFFFF */
  uint32_t ulWcStateBitOffset; /* only valid if not set to 0xFFFFFFFF */
  uint32_t ulImageStartBitOffset;
  uint32_t ulImageBitLength;

  uint32_t ulBitOffsetWithin; /* when bSmNo == 0xFF and bFmmmuNo = 0xFF, we provide
register address multiplied by 8 */
  uint8_t bSmNo; /* 0xFF not set */
  uint8_t bFmmuNo; /* 0xFF not set */

  uint16_t usReserved;
} ECM_IF_CYCLIC_SLAVE_MAPPING_ENTRY_T;

enum ECM_IF_CYCLIC_SLAVE_MAPPING_TYPE_Etag
{
  VAL_ECM_IF_CYCLIC_SLAVE_MAPPING_TYPE_TRANSMIT = 1,
  VAL_ECM_IF_CYCLIC_SLAVE_MAPPING_TYPE_RECEIVE = 2
};
```

Cyclic slave mapping entry structure description

Variable name	Type	Meaning
usDirection	UINT16	Specifies what kind of data is contained in transmit process data image. See Table 134: Definition of `usDirection`
usStationAddress	UINT16	Fixed station address of slave that is referenced by the block
usWkcCompareReceiveByteOffset	UINT16	Byte offset in receive process data image References the actual received working counter if not set to 0xFFFF.
ulWcStateBitOffset	UINT32	Bit offset in receive process data image References the actual provided WcState bit which is referring to the data if not set to 0xFFFFFFFF.
ulImageStartBitOffset	UINT32	Current start bit offset of referenced process data area
ulImageBitLength	UINT32	Current bit length of referenced process data area
ulBitOffsetWithin	UINT32	Specifies current bit offset within referenced SyncMan, FMMU or memory of ESC controller If `bFmmuNo` is set unequal 0xFF, it references to the start of the FMMU specified by bFmmuNo. If `bSmNo` is set unequal 0xFF, it references to the start of the SM specified by bSmNo. If `bFmmuNo` and `bSmNo` are both set to 0xFF, it references to the physical start address within the slave in bits. (register byte address * 8)
bSmNo	UINT8	References SyncManager of ESC controller if not set to 0xFF
bFmmuNo	UINT8	References FMMU of ESC controller if not set to 0xFF
usReserved	UINT16	Reserved for future use

Table 136: Structure `ECM_IF_CYCLIC_SLAVE_MAPPING_ENTRY_T`

Packet structure reference

```
#define ECM_IF_MAX_CYCLIC_SLAVE_MAPPING_ENTRIES ((HIL_MAX_DATA_SIZE - sizeof(uint32_t) *
2) / sizeof(ECM_IF_CYCLIC_SLAVE_MAPPING_ENTRY_T))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_CYCLIC_SLAVE_MAPPING_CNF_DATA_Ttag
{
  uint32_t ulEntriesStartOffset;
  uint32_t ulTotalEntries; /* this number can be larger than the atEntries field can hold
*/

  /* actual number of entries given by (ptPck->tHead.ulLen -
offsetof(ECM_IF_GET_WCSTATE_INFO_CNF_DATA_T, atEntries)) /
sizeof(ECM_IF_WCSTATE_INFO_ENTRY_T) */
  ECM_IF_CYCLIC_SLAVE_MAPPING_ENTRY_T atEntries[ECM_IF_MAX_CYCLIC_SLAVE_MAPPING_ENTRIES];
} ECM_IF_GET_CYCLIC_SLAVE_MAPPING_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_CYCLIC_SLAVE_MAPPING_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_CYCLIC_SLAVE_MAPPING_CNF_DATA_T tData;
} ECM_IF_GET_CYCLIC_SLAVE_MAPPING_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	8 + 22 * n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E27	`ECM_IF_CMD_GET_CYCLIC_SLAVE_MAPPING_CNF`
Data			
ulEntriesStartOffset	UINT32		Mirrored value from request
ulTotalEntries	UINT32		Total number of entries available
atEntries	ECM_IF_CYCLIC_SLAVE_MAPPING_ENTRY_T		Entries describing the location of a process data area The actual number of entries is `(ulLen – 8) / 22`. If `ulTotalEntries` is larger than the result, more entries are available.

Table 137: ECM_IF_CMD_GET_CYCLIC_SLAVE_MAPPING_CNF – Get cyclic slave mapping confirmation

4.12 Retrieval of slave diagnostic information

The retrieval of slave diagnostic information uses a generic Hilscher DPM mechanism to provide that information.

4.12.1 Provided lists

The slave diagnostic provides the following lists for the application:

- the configured slaves
- the activated slaves
- the slaves with a known fault

4.12.2 Limitations of configured slaves list

The configured slaves list as given by *Get slave handle service* (page 160) can only communicate up to 388 slaves handles.

In that case, the number of configured slaves can be retrieved by one of the two following cases:

- the number of configured slaves within Common Status block
- *Get slave handle bit list service* (page 162)

With one of those two services being used, all slaves can be accessed via *Get slave connection information service* (page 164).

4.12.3 Limitations of active/faulted slaves list

The active slaves list and faulted slaves list as given by *Get slave handle service* (page 160) can only communicate up to 388 slaves.

If this is not sufficient, the service *Get slave handle bit list service* (page 162) has to be used for retrieving data about all slaves.

4.12.4 Addressing scheme

The `ulDeviceIndex` is based on the configuration order. It is neither based on topology position nor on fixed station address.

The device index addressing scheme is described in more detail in section *Device index* (page 19).

Most other services deal with the Fixed Station address as addressing scheme. For that, it is necessary to request the data via *Get slave connection information service* (page 164).

4.12.5 Usage of slave diagnostic information packets

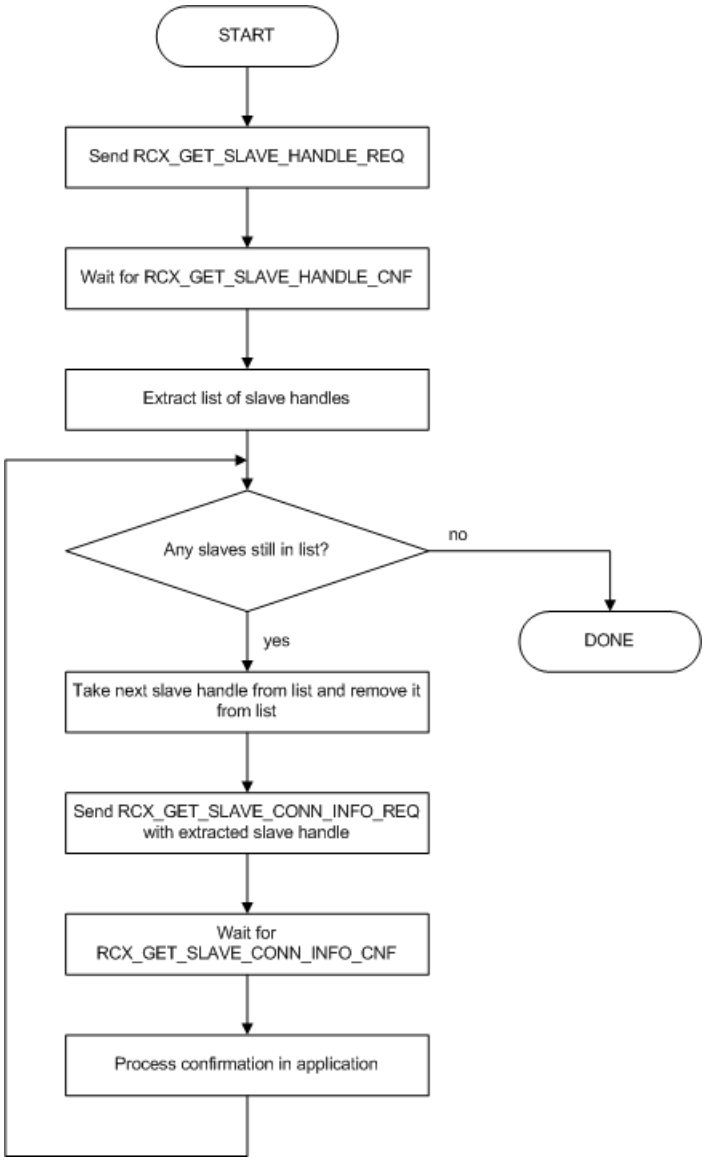


Figure 40: Flow diagram of slave diagnostic information packets

The following packets are referenced:

- HIL_GET_SLAVE_HANDLE_REQ

  Get slave handle service (page 160)

- HIL_GET_SLAVE_CONN_INFO_REQ

  Get slave connection information service (page 164)

4.12.6 Structure of per slave diagnostic data

The following structure is used for providing the slave diagnostic data in the slave diagnostic information functionality.

Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_DIAG_GET_SLAVE_DIAG_Ttag
{
  uint32_t ulStationAddress;
  uint32_t ulAutoIncAddress;
  uint32_t ulCurrentState;
  uint32_t ulLastError;
  uint8_t  szSlaveName[80];
  uint32_t fEmergencyReported;
} ETHERCAT_MASTER_DIAG_GET_SLAVE_DIAG_T;
```

Structure Description

Element	Meaning
ulStationAddress	Fixed station address of slave
ulAutoIncAddress	Auto-Increment address of slave 0x0000 first slave in topology 0xFFFF second slave in topology 0xFFFE third slave in topology
ulCurrentState	Current status of slave
ulLastError	Last error associated with slave
szSlaveName	Name of slave (NUL-terminated if not completely used)
fEmergencyReported	*TRUE* if a CoE emergency has been received by master

Table 138: Slave connection information

Description of ulCurrentState

ulCurrentState	Description
0x00	ECM_IF_STATE_NOT_CONNECTED Shown when slave is not connected (also shown when master is in INIT)
0x01	ECM_IF_STATE_INIT Slave is in INIT state
0x02	ECM_IF_STATE_PREOP Slave is in PREOP state
0x04	ECM_IF_STATE_SAFEOP Slave is in SAFEOP state
0x08	ECM_IF_STATE_OP Slave is in OP state
0x11	ECM_IF_STATE_INIT_ERR Slave is in INIT+ERR state
0x12	ECM_IF_STATE_PREOP_ERR Slave is in PREOP+ERR state
0x14	ECM_IF_STATE_SAFEOP_ERR Slave is in SAFEOP+ERR state

Table 139: ulCurrentState in Slave connection information

- When the master is in INIT state, it has not scanned the topology structure nor its slaves. So, it shows ECM_IF_STATE_NOT_CONNECTED in that case.
- ECM_IF_STATE_INIT is only shown when the master has scanned the topology and the slave is specifically set to INIT by the slave-specific target state.

4.12.7 Packets

4.12.7.1 Get slave handle service

This service retrieves the list of slaves matching a particular condition (configured, activated or faulted).

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
HIL_PACKET_GET_SLAVE_HANDLE_REQ_DATA_Tag
{
  uint32_t ulParam;
}  HIL_PACKET_GET_SLAVE_HANDLE_REQ_DATA_T;

#define HIL_LIST_CONF_SLAVES                0x00000001
#define HIL_LIST_ACTV_SLAVES                0x00000002
#define HIL_LIST_FAULTED_SLAVES             0x00000003

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_PACKET_GET_SLAVE_HANDLE_REQ_Tag
{
  HIL_PACKET_HEADER_T                     tHead;
  HIL_PACKET_GET_SLAVE_HANDLE_REQ_DATA_T  tData;
} HIL_PACKET_GET_SLAVE_HANDLE_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	4	Packet Data Length in bytes
ulCmd	UINT32	0x2F08	HIL_GET_SLAVE_HANDLE_REQ
Data			
ulParam	UINT32	0x00000001 ... 0x00000003	Parameter: 0x00000001 List of Configured Slaves 0x00000002 List of Activated Slaves 0x00000003 List of Faulted Slaves

Table 140: HIL_GET_SLAVE_HANDLE_REQ – Get slave handle request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
HIL_PACKET_GET_SLAVE_HANDLE_CNF_DATA_Tag
{
  uint32_t ulParam;
  uint32_t aulHandle[1];
}  HIL_PACKET_GET_SLAVE_HANDLE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_PACKET_GET_SLAVE_HANDLE_CNF_Tag
{
  HIL_PACKET_HEADER_T                      tHead;
  HIL_PACKET_GET_SLAVE_HANDLE_CNF_DATA_T  tData;
} HIL_PACKET_GET_SLAVE_HANDLE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	8 + amount of read data	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x2F09	HIL_GET_SLAVE_HANDLE_CNF
Data			
ulParam	UINT32	0x00000001 ... 0x00000003	Parameter: 0x00000001 List of Configured Slaves 0x00000002 List of Activated Slaves 0x00000003 List of Faulted Slaves
aulHandle[n]	UINT32		List of slave handles referred by the specified list n = (ulLen – 8) / 4

Table 141: HIL_GET_SLAVE_HANDLE_CNF – Confirmation of get slave handle request

4.12.7.2 Get slave handle bit list service

This service retrieves the bit list of slaves matching a particular condition (configured, activated or faulted). Each set bit translates into a slave handle. The first bit in a confirmation is referring to ulStartHandle. In contrast to *Get slave handle service* (page 160), this request does not have the 388 slaves limit and can address larger configurations.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_REQ_DATA_Ttag
{
  uint32_t ulListType; /* same enum as in HIL_GET_SLAVE_HANDLES_REQ */
  uint32_t ulStartHandle; /* first bit position in confirmation refers to the handle
ulStartHandle */
} ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_REQ_DATA_T tData;
} ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	8	Packet Data Length in bytes
ulCmd	UINT32	0x9E68	ECM_IF_CMD_GET_SLAVE_HANDLE_BIT_LIST_REQ
Data			
ulListType	UINT32	0x00000001 ... 0x00000003	Parameter: 0x00000001 List of Configured Slaves 0x00000002 List of Activated Slaves 0x00000003 List of Faulted Slaves
ulStartHandle	UINT32		Actual handle the first bit will refer to

Table 142: ECM_IF_CMD_GET_SLAVE_HANDLE_BIT_LIST_REQ – Get slave handle bit list request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_CNF_DATA_Ttag
{
  uint32_t ulListType; /* same enum as in HIL_GET_SLAVE_HANDLES_REQ */
  uint32_t ulStartHandle;
  uint32_t ulNumHandleBits;
  uint8_t abBitMap[1024];
} ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_CNF_DATA_T tData;
} ECM_IF_GET_SLAVE_HANDLE_BIT_LIST_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	12 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E69	ECM_IF_CMD_GET_SLAVE_HANDLE_BIT_LIST_CNF
Data			
ulListType	UINT32	0x00000001 ... 0x00000003	Parameter: 0x00000001 List of Configured Slaves 0x00000002 List of Activated Slaves 0x00000003 List of Faulted Slaves
ulStartHandle	UINT32		Actual handle the first bit will refer to
ulNumHandleBits	UINT32		Actual number of bits used in abBitmap
abBitmap[n]	UINT8		Bit List of slave handles referred by the specified list n = (ulLen – 12) referring to n *8 bits

Table 143: ECM_IF_CMD_GET_SLAVE_HANDLE_BIT_LIST_CNF – Confirmation of get save handle request

4.12.7.3 Get slave connection information service

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_DATA_Tag
{
  uint32_t ulHandle;
}  HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_Tag
{
  HIL_PACKET_HEADER_T                        tHead;
  HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_DATA_T  tData;
} HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	4	Packet Data Length in bytes
ulCmd	UINT32	0x2F0A	HIL_GET_SLAVE_CONN_INFO_REQ
Data			
ulHandle	UINT32		For addressing scheme see section *Device index* (page 19).

Table 144: HIL_GET_SLAVE_CONN_INFO_REQ – Get slave connection information request

Packet structure reference

Note: Field `tSlaveDiagData` is only shown for reference the actual packet definition does not have this field. It must be addressed by `((&ptPck->tData) + 1)`.

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_DATA_Tag
{
  uint32_t ulHandle;
  uint32_t ulStructID;
  /*
    Feldbus specific structure
  */
} HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_Tag
{
  HIL_PACKET_HEADER_T                          tHead;
  HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_DATA_T  tData;
} HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	108	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x2F0B	`HIL_GET_SLAVE_CONN_INFO_CNF`
Data			
ulHandle	UINT32		Value from request
ulStructId	UINT32	5938	structure id used by master for slave diagnostic information

Table 145: HIL_GET_SLAVE_CONN_INFO_CNF – Confirmation of get slave connection information request

Structure ETHERCAT_MASTER_DIAG_GET_SLAVE_DIAG_T

Structure described in section *Structure of per slave diagnostic data* (page 158).

4.12.7.4 Read CoE emergency messages

This service allows reading the buffered CoE emergency messages from the slaves.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_REQ_DATA_Ttag
{
  uint32_t ulSlaveHandle;
  uint32_t fDeleteEmergency; /* Flag to decide (keep emergeny(s) / clear emergeny(s)) */
} ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	8	Packet Data Length in bytes
ulCmd	UINT32	0x65001C	ETHERCAT_MASTER_CMD_READ_EMERGENCY_REQ
Data			
ulSlaveHandle	UINT32		For addressing scheme see section *Device index* (page 19).
fDeleteEmergency	UINT32		Flag to decide whether to keep emergencies (0) after read or remove from buffer (1)

Table 146: ETHERCAT_MASTER_CMD_READ_EMERGENCY_REQ – Get slave CoE emergencies request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ETHERCAT_MASTER_SLAVE_EMERGENCY_Ttag
{
  uint16_t usErrorCode;
  uint8_t bErrorRegister;
  uint8_t abErrorData[ETHERCAT_MASTER_COE_EMERGENCY_DATA_BYTES];
} ETHERCAT_MASTER_SLAVE_EMERGENCY_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_CNF_DATA_Ttag
{
  uint32_t ulSlaveHandle;
  uint32_t fDeleteEmergency;  /* value from request */
  uint32_t fOverflowOccured;  /* Emergency dropped cause of full buffer */
  ETHERCAT_MASTER_SLAVE_EMERGENCY_T
atEmergenyBuffer[ETHERCAT_MASTER_COE_NUMBER_OF_EMERGENCY]; /* up to five emergencies */
} ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER_T                              tHead;  /** packet header. */
  ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_CNF_DATA_T  tData;  /** packet request
data. */
} ETHERCAT_MASTER_PACKET_SLAVE_EMERGENCY_INFO_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	108	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x65001D	ETHERCAT_MASTER_CMD_READ_EMERGENCY_CNF
Data			
ulSlaveHandle	UINT32		Value from request
fDeleteEmergency	UINT32		Value from request
fOverflowOccured	UINT32		An overflow has occurred (i.e. emergencies have been dropped)
atEmergenyBuffer[5]	ETHERCAT_MASTER_SLAVE_EMERGENCY_T		Up to 5 emergencies (see Table 148: Structure of ETHERCAT_MASTER_SLAVE_EMERGENCY_T)

Table 147: ETHERCAT_MASTER_CMD_READ_EMERGENCY_CNF – Confirmation of get slave CoE emergencies request

Variable	Type	Value / Range	Description
usErrorCode	UINT16		Error code according to EtherCAT specification
bErrorRegister	UINT8		Error register
abErrorData[5]	UINT8		Error data

Table 148: Structure of ETHERCAT_MASTER_SLAVE_EMERGENCY_T

4.13 Retrieval of topology information

The EtherCAT master provides access to the currently known topology structure of the bus.

The service supports fragmentation for retrieving data. The current topology information is latched at start of fragmented transfer.

4.13.1 Get topology information entries

The topology information entries appear in topology order in confirmation. The port order of an EtherCAT slave is 0, 3, 1 and 2.

Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_TOPOLOGY_INFO_CNF_DATA_ENTRY_Ttag
{
  uint16_t usThisStationAddress;
  uint16_t ausPortConnectedTo[4]; /* 0xFFFF = NOT CONNECTED, 0 = CONNECTED TO MASTER */
  /* Entries in order of auto-increment position */
} ECM_IF_GET_TOPOLOGY_INFO_CNF_DATA_ENTRY_T;
```

Structure Description

Element	Meaning
usThisSlaveAddress	Fixed station address of slave
ausPortConnectedTo[0]	To which slave is the port 0 connected. 0 = MASTER 1 … n = slave fixed station address 0xFFFF = NOT CONNECTED
ausPortConnectedTo[1]	To which slave is the port 0 connected. 0 = MASTER 1 … n = slave fixed station address 0xFFFF = NOT CONNECTED
ausPortConnectedTo[2]	To which slave is the port 0 connected. 0 = MASTER 1 … n = slave fixed station address 0xFFFF = NOT CONNECTED
ausPortConnectedTo[3]	To which slave is the port 0 connected. 0 = MASTER 1 … n = slave fixed station address 0xFFFF = NOT CONNECTED

Table 149: Topology information entry

4.13.2 Get topology information packet

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_TOPOLOGY_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_GET_TOPOLOGY_INFO_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E50	ECM_IF_CMD_GET_TOPOLOGY_INFO_REQ

Table 150: ECM_IF_CMD_GET_TOPOLOGY_INFO_REQ – Get topology information request

Packet structure reference

```
#define ECM_IF_GET_TOPOLOGY_INFO_MAX_ENTRIES (1024 /
sizeof(ECM_IF_GET_TOPOLOGY_INFO_CNF_DATA_ENTRY_T))

#define ECM_IF_TOPOLOGY_INFO_PORT_NOT_CONNECTED 65535

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_TOPOLOGY_INFO_CNF_DATA_Ttag
{
  uint32_t ulTotalNumOfListEntries;
  uint32_t ulStartOfUnconnectedListEntries;
  ECM_IF_GET_TOPOLOGY_INFO_CNF_DATA_ENTRY_T
atEntries[ECM_IF_GET_TOPOLOGY_INFO_MAX_ENTRIES];
} ECM_IF_GET_TOPOLOGY_INFO_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_TOPOLOGY_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_TOPOLOGY_INFO_CNF_DATA_T tData;
} ECM_IF_GET_TOPOLOGY_INFO_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	8 + 10 * n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E51	ECM_IF_CMD_GET_TOPOLOGY_INFO_CNF
Data			
ulTotalNumOfListEntries	UINT32		Total number of list entries over all fragments
ulStartOfUnconnectedListEntries	UINT32		Start index of entries that are currently not connected
atEntries[]	ECM_IF_GET_TOPOLOGY_INFO_CNF_DATA_ENTRY_T		Topology information entries Actual number of entries in fragment is (ulLen – 8) / 10

Table 151: ECM_IF_CMD_GET_TOPOLOGY_INFO_CNF – Get topology information confirmation

4.14 ESC/SII access

4.14.1 ESC register access

4.14.1.1 Read ESC registers

This packet provides read access to a specific slave's ESC registers.

The following addressing schemes are used:

- During bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_REGS_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usPhysAddr;
  uint16_t usPhysLength;
} ECM_IF_READ_REGS_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_REGS_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_READ_REGS_REQ_DATA_T tData;
} ECM_IF_READ_REGS_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	6	Packet Data Length in bytes
ulCmd	UINT32	0x9E70	`ECM_IF_CMD_READ_REGS_REQ`
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usPhysAddr	UINT16		Physical start address in ESC physical memory address space
usPhysLength	UINT16		Byte Length of physical memory address space to be read

Table 152: `ECM_IF_CMD_READ_REGS_REQ` – Read ESC registers request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_REGS_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usPhysAddr;
  uint16_t usPhysLength;
  uint8_t abData[1024]; /* actual byte length based on usPhysLength */
} ECM_IF_READ_REGS_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_REGS_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_READ_REGS_CNF_DATA_T tData;
} ECM_IF_READ_REGS_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	6 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E71	ECM_IF_CMD_READ_REGS_CNF
Data			
usStationAddress	UINT16		Value from request
usPhysAddr	UINT16		Physical start address in ESC physical memory address space
usPhysLength	UINT16		Byte Length of physical memory address space to be read
abData[n]	UINT8[]		Register data having been read (actual array length equals to usPhysLength)

Table 153: ECM_IF_CMD_READ_REGS_CNF – Read ESC registers confirmation

4.14.1.2 Write ESC registers

This packet provides write access to a specific slave's ESC registers.

The following addressing schemes are used:

- During bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WRITE_REGS_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usPhysAddr;
  uint16_t usPhysLength;
  uint8_t abData[1024]; /* actual byte length based on usPhysLength */
} ECM_IF_WRITE_REGS_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WRITE_REGS_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_WRITE_REGS_REQ_DATA_T tData;
} ECM_IF_WRITE_REGS_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	6 + n	Packet Data Length in bytes
ulCmd	UINT32	0x9E72	`ECM_IF_CMD_WRITE_REGS_REQ`
Data			
usStationAddress	UINT16	Valid address	During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
usPhysAddr	UINT16		Physical start address in ESC physical memory address space
usPhysLength	UINT16		Byte length of physical memory address space to be written
abData[n]	UINT8[]		Register data to be written (actual array length equals to usPhysLength)

Table 154: `ECM_IF_CMD_WRITE_REGS_REQ` – Write ESC registers request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WRITE_REGS_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint16_t usPhysAddr;
  uint16_t usPhysLength;
} ECM_IF_WRITE_REGS_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WRITE_REGS_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_WRITE_REGS_CNF_DATA_T tData;
} ECM_IF_WRITE_REGS_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E73	ECM_IF_CMD_WRITE_REGS_CNF
Data			
usStationAddress	UINT16		Value from request
usPhysAddr	UINT16		Physical start address in ESC physical memory address space
usPhysLength	UINT16		Byte Length of physical memory address space being written

Table 155: ECM_IF_CMD_WRITE_REGS_CNF – Write ESC registers confirmation

4.14.2 ESC SII access

4.14.2.1 Read SII/EEPROM

This packet provides read access to a specific slave's SII/ EEPROM data.

The following addressing schemes are used:

- During bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_SII_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint32_t ulSiiWordOffset;
  uint32_t ulSiiByteLength; /* must be a multiple of 2 */
} ECM_IF_READ_SII_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_SII_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_READ_SII_REQ_DATA_T tData;
} ECM_IF_READ_SII_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	10	Packet Data Length in bytes
ulCmd	UINT32	0x9E80	ECM_IF_CMD_READ_SII_REQ
Data			
usStationAddress	UINT16		During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
ulSiiWordOffset	UINT32		SII word offset 0 => first word (at byte offset 0) 1 => second word (at byte offset 2)
ulSiiByteLength	UINT32		Length of SII data to be read in bytes Must be a multiple of 2.

Table 156: ECM_IF_CMD_READ_SII_REQ – Read SII/EEPROM request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_SII_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint32_t ulSiiWordOffset;
  uint32_t ulSiiByteLength; /* must be a multiple of 2 */
  uint8_t abData[1024];
} ECM_IF_READ_SII_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_READ_SII_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_READ_SII_CNF_DATA_T tData;
} ECM_IF_READ_SII_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	10 + n	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E81	ECM_IF_CMD_READ_SII_CNF
Data			
usStationAddress	UINT16		Value from request
ulSiiWordOffset	UINT32		SII word offset 0 => first word (at byte offset 0) 1 => second word (at byte offset 2)
ulSiiByteLength	UINT32		Length of SII data to be read in bytes
abData[n]	UINT8[]		Read data from SII/EEPROM Actual length depends on ulLen

Table 157: ECM_IF_CMD_READ_SII_CNF – Read SII/EEPROM confirmation

4.14.2.2 Write SII/EEPROM

This packet provides write access to a specific slave's SII/ EEPROM data.

The following addressing schemes are used:

- During bus scan, use *Topology position* (page 19)
- During normal operation, use *Fixed station address* (page 18)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WRITE_SII_REQ_DATA_Ttag
{
  uint16_t usStationAddress;
  uint32_t ulSiiWordOffset;
  uint32_t ulReserved; /* kept free for specific use in confirmation */
  uint8_t abData[1024]; /* actual length is defined by ulLen - 8 */
} ECM_IF_WRITE_SII_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WRITE_SII_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_WRITE_SII_REQ_DATA_T tData;
} ECM_IF_WRITE_SII_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	10 + n	Packet Data Length in bytes
ulCmd	UINT32	0x9E82	`ECM_IF_CMD_WRITE_SII_REQ`
Data			
usStationAddress	UINT16		During bus scan, use *Topology position* (page 19) During normal operation, use *Fixed station address* (page 18)
ulSiiWordOffset	UINT32		SII word offset 0 => first word (at byte offset 0) 1 => second word (at byte offset 2)
ulReserved	UINT32		Set to zero
abData[n]	UINT8[]		Data to be written Byte length must be a multiple of 2 and is defined as ulLen - 10

Table 158: `ECM_IF_CMD_WRITE_SII_REQ` – Write SII/EEPROM request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WRITE_SII_CNF_DATA_Ttag
{
  uint16_t usStationAddress;
  uint32_t ulSiiWordOffset;
  uint32_t ulWrittenByteLength;
} ECM_IF_WRITE_SII_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_WRITE_SII_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_WRITE_SII_CNF_DATA_T tData;
} ECM_IF_WRITE_SII_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E83	ECM_IF_CMD_WRITE_SII_CNF
Data			
usStationAddress	UINT16		Value from request
ulSiiWordOffset	UINT32		SII word offset 0 => first word (at byte offset 0) 1 => second word (at byte offset 2)
ulWrittenByteLength	UINT32		Actual byte length of SII data to being written in bytes

Table 159: ECM_IF_CMD_WRITE_SII_CNF – Write SII/EEPROM confirmation

4.14.3 Legacy ESC SII access (ECM V3.X API)

The legacy SII/EEPROM access API is provided for applications that have been originally developed for ECM V3.X.

However, some differences have to be considered when migrating from ECM V3.X.

- `fFixedAddressing = FALSE` is not available when master is configured
  - AutoIncAddress is not necessarily always mapped to same slave. Only supported during Bus scan.
  - For description of Auto-Inc address, see *Auto-increment address* (page 18)
- `fAssignAccessBack` is not evaluated, the ESC access is always given back to PDI

4.14.3.1 Read SII/EEPROM (Legacy)

This packet provides read access to a specific slave's SII/ EEPROM data.

The following addressing schemes are used:

- During bus scan and fFixedAddressing = FALSE, use *Auto-increment address* (page 18)
- During bus scan and fFixedAddressing = TRUE, use *Topology position* (page 19)
- During normal operation and fFixedAddressing = TRUE, use *Fixed station address* (page 18)

Note: This packet is provided for applications migrating from ECM V3.X.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_EEPROM_READ_REQ_DATA_Ttag
{
  uint32_t fFixedAddressing; /* TRUE: use fixed addressing (requires configuration),
FALSE: use auto increment addressing */
  uint16_t usSlaveAddress; /* Slave Address, fixed or auto increment address depending on
fFixedAddressing */
  uint16_t usEEPromStartOffset; /* Address to start EEPRom read from, (16bit WORD count)
*/
  uint16_t usReadLen; /* number of 16bit-WORDs */
  uint16_t usTimeout; /* time in ms */
} ETHERCAT_MASTER_PACKET_EEPROM_READ_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_EEPROM_READ_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_EEPROM_READ_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_EEPROM_READ_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	12	Packet Data Length in bytes
ulCmd	UINT32	0x650040	ETHERCAT_MASTER_CMD_EEPROM_READ_REQ
Data			
fFixedAddressing	BOOL	TRUE, FALSE	Addressing mode: TRUE: use fixed addressing (configured or bus scan) FALSE: use auto increment addressing (only allowed on Bus scan)
usSlaveAddress	UINT16	Valid address	Slave Address (fixed or auto increment address depending on fFixedAddressing)
usEEPromStartOffset	UINT16		Word Start offset in EEPROM 0x0000 => first word (at byte offset 0) 0x0001 => second word (at byte offset 2)
usReadLen	UINT16		Number of 16 bit words
usTimeout	UINT16		Timeout in ms

Table 160: ETHERCAT_MASTER_CMD_EEPROM_READ_REQ – Read SII/ EEPROM request (Legacy)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_EEPROM_READ_CNF_DATA_Ttag
{
  uint32_t fFixedAddressing; /* value from request */
  uint16_t usSlaveAddress; /* value from request */
  uint16_t usEEPromStartOffset; /* value from request */
  uint16_t ausReadData[750]; /* read data, up to 750 WORDs */
} ETHERCAT_MASTER_PACKET_EEPROM_READ_CNF_DATA_T;

#define ETHERCAT_MASTER_EEPROM_READ_EMPTY_SIZE (
sizeof(ETHERCAT_MASTER_PACKET_EEPROM_READ_CNF_DATA_T) - 750 * sizeof (uint16_t))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_EEPROM_READ_CNF_Ttag
{
  HIL_PACKET_HEADER_T                          tHead;  /** packet header.       */
  ETHERCAT_MASTER_PACKET_EEPROM_READ_CNF_DATA_T  tData;  /** packet request data. */
} ETHERCAT_MASTER_PACKET_EEPROM_READ_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	8 + n * 2	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x650041	ETHERCAT_MASTER_CMD_EEPROM_READ_CNF
Data			
fFixedAddressing	BOOL	TRUE, FALSE	Value from request
usSlaveAddress	UINT16		Value from request
usEEPromStartOffset	UINT16		Value from request
ausReadData[n]	UINT16[]		Words read from SII/EEPROM Actual length depends on (ulLen – 8) / 2

Table 161: ETHERCAT_MASTER_CMD_EEPROM_READ_CNF – Read SII/EEPROM confirmation (Legacy)

4.14.3.2 Write SII/EEPROM (Legacy)

This packet provides write access to a specific slave's SII/ EEPROM data.

The following addressing schemes are used:

- During bus scan and fFixedAddressing = FALSE, use *Auto-increment address* (page 18)
- During bus scan and fFixedAddressing = TRUE, use *Topology position* (page 19)
- During normal operation and fFixedAddressing = TRUE, use *Fixed station address* (page 18)

Note: This packet is provided for applications migrating from ECM V3.X.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_EEPROM_WRITE_REQ_DATA_Ttag
{
  uint32_t fFixedAddressing; /* TRUE: use fixed addressing (requires configuration),
FALSE: use auto increment addressing */
  uint16_t usSlaveAddress; /* Slave Address, fixed or auto increment address depending on
fFixedAddressing */
  uint16_t usEEPromStartOffset; /* Address to start EEPRom write from (16bit WORD count)
*/
  uint32_t fAssignAccessBack; /* give slave the EEPROM control back? Set to TRUE to apply
new data. Set to FALSE if further fragments follows. */
  uint16_t usTimeout; /* in ms */
  uint16_t ausWriteData[750]; /* data to write, up to 750 WORDs */
} ETHERCAT_MASTER_PACKET_EEPROM_WRITE_REQ_DATA_T;

/* packet without payload */
#define ETHERCAT_MASTER_EEPROM_WRITE_EMPTY_SIZE (
sizeof(ETHERCAT_MASTER_PACKET_EEPROM_WRITE_REQ_DATA_T) - 750 * sizeof (uint16_t) )

/* at least one WORD must be written */
#define ETHERCAT_MASTER_EEPROM_WRITE_MIN_SIZE ( ETHERCAT_MASTER_EEPROM_WRITE_EMPTY_SIZE +
sizeof (uint16_t) )

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_EEPROM_WRITE_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_EEPROM_WRITE_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_EEPROM_WRITE_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	14 + n * 2	Packet Data Length in bytes
ulCmd	UINT32	0x650042	ETHERCAT_MASTER_CMD_EEPROM_WRITE_REQ
Data			
fFixedAddressing	BOOL	TRUE, FALSE	Addressing mode: TRUE: use fixed addressing (configured or bus scan) FALSE: use auto increment addressing (only allowed on Bus scan)
usSlaveAddress	UINT16	Valid address	Slave Address (fixed or auto increment address depending on fFixedAddressing)
usEEPromStartOffset	UINT16		Word Start offset in EEPROM 0x0000 => first word (at byte offset 0) 0x0001 => second word (at byte offset 2)
fAssignAccessBack	BOOL		
usTimeout	UINT16		Timeout in ms
ausWriteData[n]	UINT16[]		Data to be written to SII Actual length depends on (ulLen – 14) / 2

Table 162: ETHERCAT_MASTER_CMD_EEPROM_WRITE_REQ – Write SII/EEPROM request (Legacy)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_EEPROM_WRITE_CNF_DATA_Ttag
{
  uint32_t fFixedAddressing; /* value from request */
  uint16_t usSlaveAddress; /* value from request */
  uint16_t usEEPromStartOffset; /* value from request */
} ETHERCAT_MASTER_PACKET_EEPROM_WRITE_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_EEPROM_WRITE_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_EEPROM_WRITE_CNF_DATA_T tData;
} ETHERCAT_MASTER_PACKET_EEPROM_WRITE_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x650043	ETHERCAT_MASTER_CMD_EEPROM_WRITE_CNF
Data			
fFixedAddressing	BOOL	TRUE, FALSE	Value from request
usSlaveAddress	UINT16		Value from request
usEEPromStartOffset	UINT16		Word Start offset in EEPROM 0x0000 => first word (at byte offset 0) 0x0001 => second word (at byte offset 2)

Table 163: ETHERCAT_MASTER_CMD_EEPROM_WRITE_CNF – Write SII/EEPROM confirmation (Legacy)

4.15 ExtSync

4.15.1 Description of ExtSync functionality

ExtSync provides a means to synchronize an EtherCAT network to another time source e.g. another EtherCAT network.

ExtSync specifics

- ExtSync synchronize two networks so that the time difference between both is defined and stable.
- ExtSync does not match the start of the bus cycles of two EtherCAT networks. It only ensures that a timestamp can be transformed from primary to secondary network and vice versa.

Detailed status information specific to ExtSync is provided through:

- Via process data: *Process data area type: ExtSync status* (page 147)
- Via packet: *Get ExtSync information packet* (page 184)

4.15.2 Get ExtSync information packet

This packet allows get information about the current ExtSync status.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_EXT_SYNC_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_GET_EXT_SYNC_INFO_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E66	ECM_IF_CMD_GET_EXT_SYNC_INFO_REQ

Table 164: ECM_IF_CMD_GET_EXT_SYNC_INFO_REQ – Get ExtSync information request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_EXT_SYNC_INFO_CNF_DATA_Ttag
{
  uint32_t ulExtSyncInfoFlags;
  uint64_t ullInternalTimestampNs;
  uint64_t ullExternalTimestampNs;
  int32_t lTimeControlValueBySlave;
  uint16_t usExtSyncStationAddress;
  uint64_t ullDcToExtTimeOffsetNs; /* internal DC timestamp (ns) + ullDcToExtTimeOffsetNs
=> external clock time (ns) */
  uint32_t ulLastUpdateDiffNs;
  int32_t lLastControlDeltaDiffNs;
  int32_t lLastControlDeltaDeltaDiffNs;
  uint16_t usControlledStationAddress;
  uint32_t ulExtSyncUpdateCount;
  uint32_t ulDeviationPosMaxMag;
  uint32_t ulDeviationNegMaxMag;
} ECM_IF_GET_EXT_SYNC_INFO_CNF_DATA_T;

/* ulExtSyncInfoFlags */
enum ECM_IF_EXT_SYNC_INFO_FLAGS_Etag
{
  MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_SYNC_MODE_SLAVE = 0x00000001,
  MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_IS_64BIT = 0x00000004,
  MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_NOT_CONNECTED = 0x00000010,
  MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_SYNC_MODE_MASTER = 0x00000020,
  MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_CONNECTED_AS_SLAVE = 0x00008000, /* result of
!(External Device Not Connected) && (Sync Mode.Bit 1) */
  MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_SYNC_CONTROL_STATE = 0x00FF0000,
  SRT_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_SYNC_CONTROL_STATE = 16,
  MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_DC_TO_EXT_OFFSET_VALID = 0x20000000,
  MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_ACTIVE = 0x40000000, /* master is actively
using External Synchronization on device */
  MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_CONFIGURED = 0x80000000
};

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_EXT_SYNC_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_EXT_SYNC_INFO_CNF_DATA_T tData;
} ECM_IF_GET_EXT_SYNC_INFO_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E67	`ECM_IF_CMD_GET_EXT_SYNC_INFO_CNF`
Data			
ulExtSyncInfoFlags	UINT32		See *Table 166: Parameter `ulExtSyncInfoFlags`*
ullInternalTimestampNs	UINT64		Last known 32 bit / 64 bit own network DC timestamp as seen by ExtSync device
ullExternalTimestampNs	UINT64		Last known 32 bit / 64 bit external timestamp as seen by ExtSync device
lTimeControlValueBySlave	INT32		Time value provided from ExtSync device data Not used by master for actual ExtSync implementation
usExtSyncStationAddress	UINT16		Fixed station address of Ext Sync device See *Fixed station address* (page 18)
ullDcToExtTimeOffsetNs	UINT64		Time difference between this network and the other network providing the ExtSync base time. Relationship of timestamps: `DcToExtTimeOffsetNs = ExternalTimestampNs - InternalTimestampNs`
ulLastUpdateDiffNs	UINT32		Last delta time between ExtSync processing in ns
lLastControlDeltaDiffNs	INT32		ExtSync diagnostic data. Internal use
lLastControlDeltaDeltaDiffNs	INT32		ExtSync diagnostic data. Internal use
usControlledStationAddress	UINT16		Fixed station address of DC reference clock See *Fixed station address* (page 18)
ulExtSyncUpdateCount	UINT32		Counter of processed ExtSync updates

Table 165: `ECM_IF_CMD_GET_EXT_SYNC_INFO_CNF` – Get ExtSync information confirmation

Definition of ulExtSyncInfoFlags

Bits	Name (Bit mask)	Description
31	MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_CONFIGURED (0x80000000)	Ext Sync device has been configured
30	MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_ACTIVE (0x40000000)	Master is actively using External synchronization on device
29	MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_DC_TO_EXT_OFFSET_VALID (0x20000000)	ullDcToExtTimeOffsetNs is valid
28 … 24	Reserved	reserved
23 … 16	Used internally	Used internally
15	MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_CONNECTED_AS_SLAVE (0x00008000)	Ext Sync Device is connected as slave
14 … 6	Reserved	Reserved for future use
5	MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_SYNC_MODE_MASTER (0x00000020)	Master is providing ExtSync to other network
4	MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_EXT_DEVICE_NOT_CONNECTED (0x00000010)	ExtSync device is not connected when bit is set
3	Reserved	Reserved
2	MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_IS_64BIT (0x00000004)	ExtSync timestamp size If not set, the timestamp has 32 bit. If set, the timestamp has 64 bit.
1	Reserved	Reserved
0	MSK_ECM_IF_EXT_SYNC_INFO_FLAGS_SYNC_MODE_SLAVE (0x00000001)	Master is using ExtSync reference provided by other network

Table 166: Parameter ulExtSyncInfoFlags

4.15.3 Reset ExtSync Max Deviations

The application can use this service to reset the maximum ExtSync deviations diagnostic values.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_RESET_EXT_SYNC_MAX_DEVIATIONS_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_RESET_EXT_SYNC_MAX_DEVIATIONS_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x9E52	ECM_IF_CMD_RESET_EXT_SYNC_MAX_DEVIATIONS_REQ

Table 167: ECM_IF_RESET_EXT_SYNC_MAX_DEVIATIONS_REQ – Reset Max Deviations request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_RESET_EXT_SYNC_MAX_DEVIATIONS_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_RESET_EXT_SYNC_MAX_DEVIATIONS_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E53	ECM_IF_CMD_RESET_EXT_SYNC_MAX_DEVIATIONS_CNF

Table 168: ECM_IF_CMD_RESET_EXT_SYNC_MAX_DEVIATIONS_CNF – Reset Max Deviations confirmation

4.16 Sync with external Pin

To synchronize two EtherCAT Masters, external signals can be used.

Possibility 1: An external synchronization output signal is connected to the sync input pin of each EtherCAT Master.

Possibility 2: One EtherCAT Master has a synchronization output signal that is connected to the sync input pin of the other EtherCAT Master(s).

On the EtherCAT Master device, the synchronization pins Sync0 and Sync1 can be used for synchronization purpose with external signals. Pin Sync0 or Sync1 can be used as sync input or as sync output.

Note: The external pin synchronization configuration can only be changed while the master is in BUS_OFF state.

To set the configuration for pins Sync0 and Sync1, the application has to use the service *Select Sync for External Pin* during BUS_OFF state.

Please obey the following note, when you connect pins.

Note: Make sure that never two outputs drive against each other. Two outputs that drive against each other cause a too high current and result in device damage.

4.16.1 Select Sync for External Pin

This service allows the application to reconfigure the external synchronization modules.

Note: The external pin synchronization configuration can only be changed while the master is in BUS_OFF state.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SELECT_SYNC_CONFIG_REQ_DATA_Ttag
{
  uint32_t ulSyncModuleId;
  uint32_t ulTimeSyncModuleId;
  uint16_t usSlaveAddress;
  uint32_t aulSyncModuleParameters[10]; /* depends on ulSyncModuleTypeId */
  uint32_t aulTimeSyncModuleParameters[10]; /* depends on ulTimeSyncModuleTypeId */
} ECM_IF_SELECT_SYNC_CONFIG_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SELECT_SYNC_CONFIG_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_SELECT_SYNC_CONFIG_REQ_DATA_T tData;
} ECM_IF_SELECT_SYNC_CONFIG_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	90	Packet Data Length in bytes
ulCmd	UINT32	0x9E58	ECM_IF_CMD_SELECT_SYNC_CONFIG_REQ
Data			
ulSyncModuleId	UINT32	0 … 6	Sync module slection See Table 170 on page 189.
ulTimeSyncModuleId	UINT32	0 … 4	Time sync module selection See table Table 171 on page Table 171. Table 172 lists possible combinations of ulSyncModuleId and ulTimeSyncModuleId.
usSlaveAddress	UINT16	0, 0xFFFF	0 when not using ETG.1200 External Synchronization via PDO 0xFFFF when allowing autoselecting first available ETG.1200 External Synchronization slave
aulSyncModuleParameters	UINT32[10]	0	Sync module parameters Set to zero. Reserved for future use.
aulTimeSyncModuleParameters	UINT32[10]	0	Time sync module parameters Set to zero. Reserved for future use.

Table 169: ECM_IF_SELECT_SYNC_CONFIG_REQ – Select sync configuration request

ulSyncModuleId to select sync module

ulSyncModuleId	Sync0/Sync1	Definition / description
0	-	No SyncOut
1	Sync0 output	Sync0 RxEnd PosEdge Compatibility mode derived from ECM V3
2	Sync1 output	Sync1 RxEnd PosEdge Compatibility mode for ECM V3 drop-in replacement. Default setting
3	Sync0 output	Sync0 CycleStart PosEdge Usable as input for latch mode with PosEdge
4	Sync1 output	Sync1 CycleStart PosEdge Usable as input for latch mode with PosEdge
5	Sync0 output	Sync0 CycleStart NegEdge Usable as input for latch mode with NegEdge
6	Sync1 output	Sync1 CycleStart NegEdge Usable as input for latch mode with NegEdge

Table 170: ulSyncModuleId values

ulTimeSyncModuleId to select sync module

ulSyncModuleId	Sync0/Sync1	Definition / description
0	-	No SyncIn
1	Sync0 input	Sync0 Latch PosEdge for CycleStart
2	Sync1 input	Sync1 Latch PosEdge for CycleStart
3	Sync0 input	Sync0 Latch NegEdge for CycleStart
4	Sync1 input	Sync1 Latch NegEdge for CycleStart

Table 171: ulTimeSyncModuleId values

Synchronization modes and combinations of ulSyncModuleId and ulTimeSyncModuleId

The following tables lists value possible combinations of ulSyncModuleId and ulTimeSyncModuleId for the Loadable Firmware (LFW) for netX 100/500.

Configuration	ulSyncModuleId	ulTimeSyncModuleId	Definition / description
No SyncOut / No SyncIn	0	0	Synchronization via pins Sync0 and Sync1 deactivated.
SyncOut	1	0	Sync0 RxEnd PosEdge
	2	0	Sync1 RxEnd PosEdge
	3	0	Sync0 CycleStart PosEdge
	4	0	Sync1 CycleStart PosEdge
	5	0	Sync0 CycleStart NegEdge
	6	0	Sync1 CycleStart NegEdge
SyncIn	0	1	Sync0 Latch PosEdge for CycleStart
	0	2	Sync1 Latch PosEdge for CycleStart
	0	3	Sync0 Latch NegEdge for CycleStart
	0	4	Sync1 Latch NegEdge for CycleStart

Table 172: Possible combinations of ulSyncModuleId and ulTimeSyncModuleId

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_SELECT_SYNC_CONFIG_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} ECM_IF_SELECT_SYNC_CONFIG_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E59	ECM_IF_CMD_SELECT_SYNC_CONFIG_CNF

Table 173: ECM_IF_SELECT_SYNC_CONFIG_CNF – Set sync configuration confirmation

4.16.2 Enumeration of available sync and timesync modules

This service allows the application to request data about available sync and timesync modules. Additionally, it provides the exclusion list.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_ENUM_SYNC_CONFIG_REQ_DATA_Ttag
{
  uint32_t ulIdx;
} ECM_IF_ENUM_SYNC_CONFIG_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_ENUM_SYNC_CONFIG_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_ENUM_SYNC_CONFIG_REQ_DATA_T tData;
} ECM_IF_ENUM_SYNC_CONFIG_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	4	Packet Data Length in bytes
ulCmd	UINT32	0x9E5A	ECM_IF_CMD_ENUM_SYNC_CONFIG_REQ
Data			
ulIdx	UINT32		Index of module table to retrieve specific module

Table 174: ECM_IF_ENUM_SYNC_CONFIG_REQ – Enumerate sync configurations request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_ENUM_SYNC_CONFIG_CNF_DATA_Ttag
{
  uint32_t ulIdx;
  uint32_t ulEntryType;
  uint32_t ulModuleId;
  uint32_t aulExcludes[(HIL_MAX_DATA_SIZE - sizeof(uint16_t) * 5) / sizeof(uint32_t)];
} ECM_IF_ENUM_SYNC_CONFIG_CNF_DATA_T;

/* ulEntryType */
#define ECM_IF_ENUM_SYNC_CONFIG_TYPE_SYNC 0
#define ECM_IF_ENUM_SYNC_CONFIG_TYPE_TIMESYNC 1
#define ECM_IF_ENUM_SYNC_CONFIG_TYPE_EXTSYNC 2 /* ulModuleId is slave address */

/* ulModuleId for ECM_IF_ENUM_SYNC_CONFIG_TYPE_SYNC */
#define ECM_IF_ENUM_SYNC_CONFIG_SYNC_TYPE_DEACTIVATED 0
#define ECM_IF_ENUM_SYNC_CONFIG_SYNC_TYPE_SYNC0_RXEND_POSEDGE 1
#define ECM_IF_ENUM_SYNC_CONFIG_SYNC_TYPE_SYNC1_RXEND_POSEDGE 2
#define ECM_IF_ENUM_SYNC_CONFIG_SYNC_TYPE_SYNC0_CYCLESTART_POSEDGE 3
#define ECM_IF_ENUM_SYNC_CONFIG_SYNC_TYPE_SYNC1_CYCLESTART_POSEDGE 4
#define ECM_IF_ENUM_SYNC_CONFIG_SYNC_TYPE_SYNC0_CYCLESTART_NEGEDGE 5
#define ECM_IF_ENUM_SYNC_CONFIG_SYNC_TYPE_SYNC1_CYCLESTART_NEGEDGE 6

/* ulModuleId for ECM_IF_ENUM_SYNC_CONFIG_TYPE_TIMESYNC */
#define ECM_IF_ENUM_SYNC_CONFIG_TIMESYNC_TYPE_DEACTIVATED 0
#define ECM_IF_ENUM_SYNC_CONFIG_TIMESYNC_TYPE_SYNC0_LATCH_POSEDGE 1
#define ECM_IF_ENUM_SYNC_CONFIG_TIMESYNC_TYPE_SYNC1_LATCH_POSEDGE 2
#define ECM_IF_ENUM_SYNC_CONFIG_TIMESYNC_TYPE_SYNC0_LATCH_NEGEDGE 3
#define ECM_IF_ENUM_SYNC_CONFIG_TIMESYNC_TYPE_SYNC1_LATCH_NEGEDGE 4

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_ENUM_SYNC_CONFIG_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_ENUM_SYNC_CONFIG_CNF_DATA_T tData;
} ECM_IF_ENUM_SYNC_CONFIG_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	12+n*4	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x9E5B	ECM_IF_CMD_ENUM_SYNC_CONFIG_CNF
Data			
ulIdx	UINT32		Same as in request
ulEntryType	UINT32	0 … 2	See Table 176
ulModuleId	UINT32		Module id
aulExcludes	UINT32[n]		Excluded modules If Sync Module, time sync modules are referenced If Time Sync Module, sync modules are referenced

Table 175: ECM_IF_ENUM_SYNC_CONFIG_CNF – Enumerate sync configurations confirmation

Values of `ulEntryType`

Value	Definition / description
0	`ECM_IF_ENUM_SYNC_CONFIG_TYPE_SYNC` Sync module
1	`ECM_IF_ENUM_SYNC_CONFIG_TYPE_TIMESYNC` Time sync module (e.g. external pin latch)
2	`ECM_IF_ENUM_SYNC_CONFIG_TYPE_EXTSYNC` ETG.1200 ExtSync slave clock. ulModuleId holds slave address in this case

Table 176: ulEntryType values

4.17 Reading frame error counters

The service allows reading error counters for send and receive frames on the main port and redundancy port.

Note: This service is not meant for determining I/O data consistency. For determining I/O data consistency, please use the Wc state bits.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_ERROR_COUNTERS_REQ_DATA_Ttag
{
  uint32_t fResetAfterRead;
} ECM_IF_GET_ERROR_COUNTERS_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_ERROR_COUNTERS_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_ERROR_COUNTERS_REQ_DATA_T tData;
} ECM_IF_GET_ERROR_COUNTERS_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	1	Packet Data Length in bytes
ulCmd	UINT32	0x9E6E	ECM_IF_CMD_GET_ERROR_COUNTERS_REQ
Data			
fResetAfterRead	UINT32		0 = counters are not reset after read. 1 = counters are reset after read.

Table 177: ECM_IF_GET_ERROR_COUNTERS_REQ – Get error counters request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ECM_IF_GET_ERROR_COUNTERS_CNF_DATA_PORT_Ttag
{
  uint32_t fValid;
  uint32_t ulTransmittedOk;
  uint32_t ulLinkDownDuringTransmission;
  uint32_t ulUtxUnderflowDuringTransmission;
  uint32_t ulFramesReceivedOk;
  uint32_t ulFcsErrors;
  uint32_t ulAlignmentErrors;
  uint32_t ulFrameTooLongErrors;
  uint32_t ulRuntFramesReceived;
  uint32_t ulCollisionFragmentsReceived;
  uint32_t ulDroppedDueLowResource;
  uint32_t ulDroppedDueUrxOverflow;
  uint32_t ulRxFatalError;
} ECM_IF_GET_ERROR_COUNTERS_CNF_DATA_PORT_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_ERROR_COUNTERS_CNF_DATA_Ttag
{
  ECM_IF_GET_ERROR_COUNTERS_CNF_DATA_PORT_T tMainPort;
  ECM_IF_GET_ERROR_COUNTERS_CNF_DATA_PORT_T tRedPort;
} ECM_IF_GET_ERROR_COUNTERS_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECM_IF_GET_ERROR_COUNTERS_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ECM_IF_GET_ERROR_COUNTERS_CNF_DATA_T tData;
} ECM_IF_GET_ERROR_COUNTERS_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue handle, unchanged
ulLen	UINT32	104	Packet Data Length in bytes
ulCmd	UINT32	0x9E6F	ECM_IF_CMD_GET_ERROR_COUNTERS_CNF
Data - Structure for tMainPort and tRedPort			
fValid	UINT32		0 = Error counters not valid 1 = Error counters valid
ulTransmittedOk	UINT32		Counter for successful transmissions
ulLinkDownDuringTransmission	UINT32		Counter for link down during transmission
ulUtxUnderflowDuringTransmission	UINT32		Counter for underlow during transmission
ulFramesReceivedOk	UINT32		Counter for successful received frames
ulFcsErrors	UINT32		Counter for FCS errors
ulAlignmentErrors	UINT32		Counter for alignment errors
ulFrameTooLongErrors	UINT32		Counter for frame too long errors
ulRuntFramesReceived	UINT32		Counter for frames with less than the minimum length of 64 bytes with bad FCS (runt frames) received
ulCollisionFragmentsReceived	UINT32		Counter for collision fragments received
ulDroppedDueLowResource	UINT32		Counter for frames dropped due to low resources
ulDroppedDueUrxOverflow	UINT32		Counter for frames dropped due to urx overflow
ulRxFatalError	UINT32		Counter for fatal receive errors

Table 178: ECM_IF_GET_ERROR_COUNTERS_CNF – Get error counters confirmation

4.18 Bus scan

4.18.1 Packet parameter `ulPortState`

The port state specifies what ports are active on a given slave.

The following table outlines what port state flags are defined:

Bits	Name of bitmask / Description
15	DL Status: Communication on Port 3
14	DL Status: Communication on Port 2
13	DL Status: Communication on Port 1
12	DL Status: Communication on Port 0
11	DL Status: Loop closed on Port 3
10	DL Status: Loop closed on Port 2
9	DL Status: Loop closed on Port 1
8	DL Status: Loop closed on Port 0
7	DL Status: Link on Port 3
6	DL Status: Link on Port 2
5	DL Status: Link on Port 1
4	DL Status: Link on Port 0
3	Slave is connected to Port 3
2	Slave is connected to Port 2
1	Slave is connected to Port 1
0	Slave is connected to Port 0

Table 179: Meaning of `ulPortState`

Bits 0 to 3 are generated by the following operation on Bits 4-15:

```
ulLowNibble = ((~(ulPortState >> 8)) & (ulPortState >> 12));
```

If a port does not exist, all bits of a given port are set to 0.

The port order of an EtherCAT slave is 0, 3, 1 and 2.

4.18.2 Generic bus scan

The generic bus scan provides a common definition of how to handle the service in detail. Therefore, this chapter will only present the specifics related to the EtherCAT master. Its basic implementation is done according to document Automatic Bus scan which is listed in 1.8 References.

4.18.2.1 Relation: Topology Address and Auto-Increment Address in generic bus scan

The device index in generic bus scan is mapped to the auto-increment addresses according the following rules in UInt16 calculation:

```
TopologyAddress = 1 – AutoInc-Address
AutoInc-Address = 1 – Topology-Address
```

First slave in topology has topology address 1 and Auto-Inc address 0.

Therefore, the application can directly use the device index for accessing a particular slave's service channel which is based on topology address as well.

One reason for this mapping is to have access to acyclic access features during bus scan. This will allow extracting more details from the slave on an as needed basis.

This section is a short summary of the following two sections provided earlier:

- *Auto-increment address* (page 18)
- *Topology position* (page 19)

4.18.2.2 Acyclic services access

The generic bus scan allows accessing all acyclic services that can be reached on a slave when being in PREOP.

The used addressing scheme when using generic bus scan is topology position (see *Topology position*, page 19) for all services including retrieving Identity information via *Get device info service* (page 199).

The following acyclic services can be used with restrictions in generic bus scan:

- *CoE services* (page 70)
- *SoE services* (page 120)
- *ESC register access* (page 170)
- *ESC SII access* (page 174)

The following legacy acyclic services can only be used with fixed station address mode (effectively using topology position during generic bus scan):

- *Legacy ESC SII access (ECM V3.X API)* (page 178)

4.18.2.3 Using generic bus scan flow

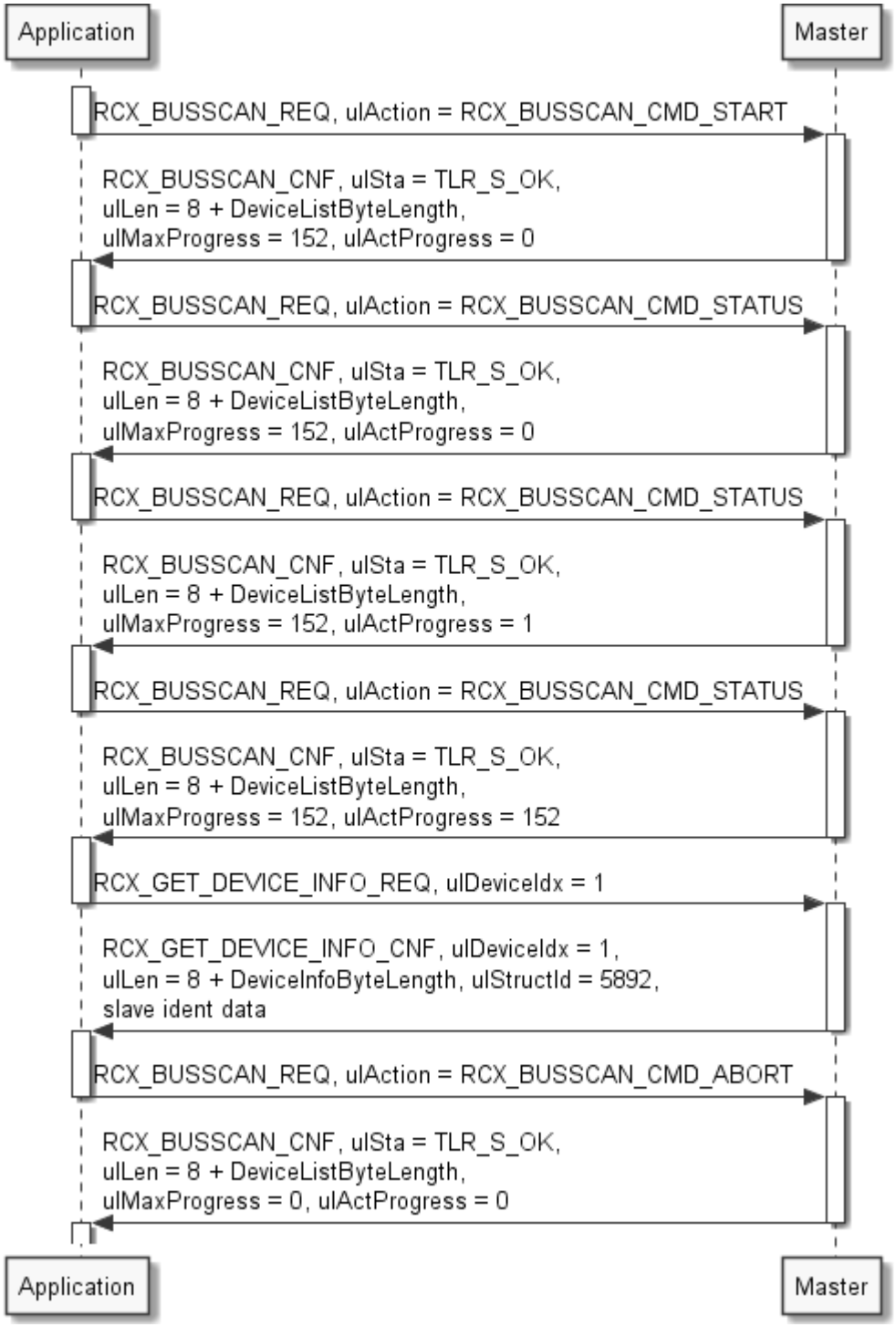


Figure 41: Example packet flow for using generic bus scan

`ulMaxProgress` and `ulActProgress` are compared for equality to determine completion of Bus scan.

4.18.2.4 Get device info service

This service is used for requesting the current device information of a connected slave and has EtherCAT specific parts which will be shown here.

The following addressing schemes are used:

- *Topology position* (page 19)

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_GET_DEVICE_INFO_REQ_DATA_Ttag
{
  uint32_t ulDeviceIdx;
} HIL_GET_DEVICE_INFO_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_GET_DEVICE_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER_T            tHead;
  HIL_GET_DEVICE_INFO_REQ_DATA_T tData;
} HIL_GET_DEVICE_INFO_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	4	Packet Data Length in bytes
ulCmd	UINT32	0x2F24	`HIL_GET_DEVICE_INFO_REQ`
Data			
ulDeviceIdx	UINT32		Device index references to topology position based on 0 to number of slaves - 1

Table 180: `HIL_GET_DEVICE_INFO_REQ` – Get device info request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_GET_DEVICE_INFO_CNF_DATA_Ttag
{
  uint32_t ulDeviceIdx;
  uint32_t ulStructId;
  /* uint8_t  tStruct; Fieldbus specific structure */
} HIL_GET_DEVICE_INFO_CNF_DATA_T;

typedef struct HIL_GET_DEVICE_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER_T            tHead;
  HIL_GET_DEVICE_INFO_CNF_DATA_T tData;
} HIL_GET_DEVICE_INFO_CNF_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ETHERCAT_MASTER_BUS_SCAN_INFO_Ttag
{
  uint32_t ulVendorId;
  uint32_t ulProductCode;
  uint32_t ulRevisionNumber;
  uint32_t ulSerialNumber;
  uint32_t ulPortState;
} ETHERCAT_MASTER_BUS_SCAN_INFO_T;
```

Note: `tDevInfoData` is not part of the `HIL_GET_DEVICE_INFO_CNF_T`. It is merely shown where it is placed. It must be addressed by `((&ptPck->tData) + 1)`.

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	4 on success or 0 in case of error	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x2F25	`HIL_GET_DEVICE_INFO_CNF`
Data			
ulDeviceIdx	UINT32		Device Index
ulStructId	UINT32	5892	Structure Id
Data			
ulVendorId	UINT32		Vendor Id
ulProductCode	UINT32		Product Code
ulRevisionNumber	UINT32		Revision Number
ulSerialNumber	UINT32		Serial Number
ulPortState	UINT32		For details see Packet parameter `ulPortState` (page 196)

Table 181: `HIL_GET_DEVICE_INFO_CNF` – Get device info confirmation

4.18.3 Legacy bus scan

4.18.3.1 Limits of legacy bus scan

The legacy bus scan is limited to retrieving slave identity data only. It cannot access acyclic services. Therefore, it cannot be used for scanning Modular Device Profile devices.

- For new applications, please use the *Generic bus scan* (page 197)

The legacy bus scan is only provided for applications that have been originally developed for ECM V3.X.

4.18.3.2 Using legacy bus scan

The following sequence of packets applies to the legacy bus scan mechanism:

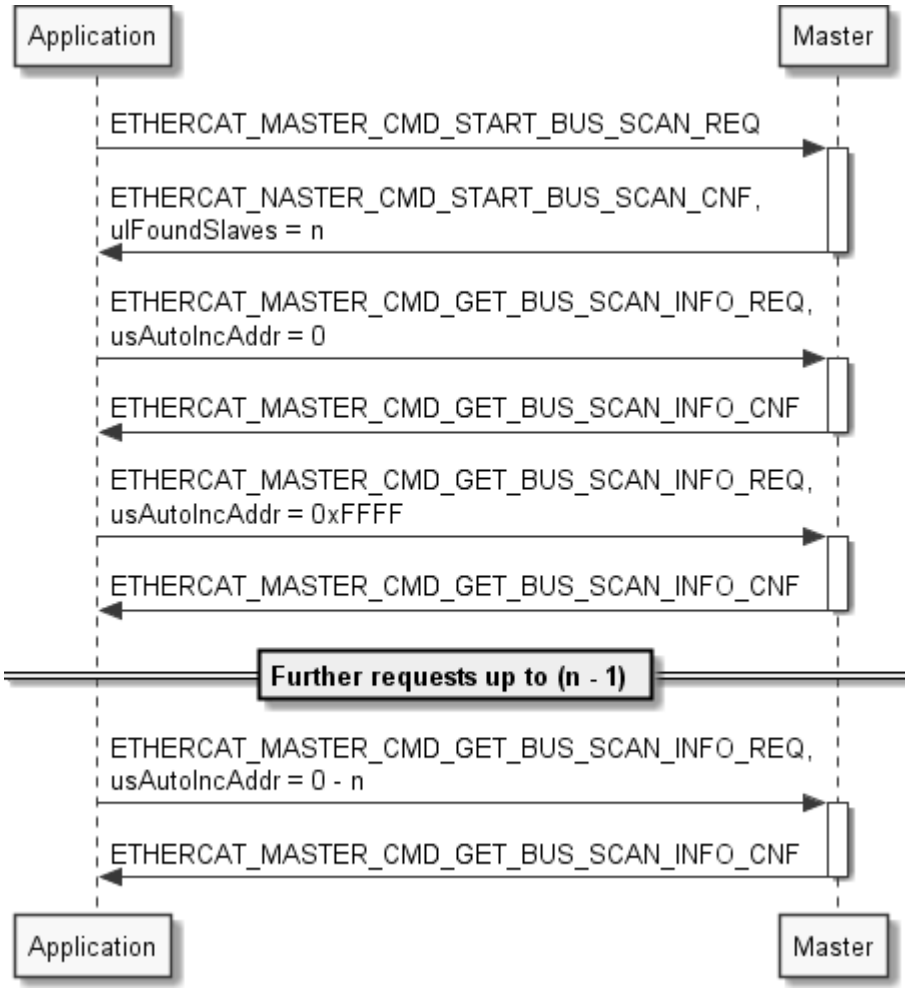


Figure 42: Using legacy bus scan

4.18.3.3 Start bus scan service (Legacy)

The bus scan is started using the packet ETHERCAT_MASTER_CMD_START_BUS_SCAN_REQ. After the confirmation was returned, the slave information can be read out. This is done by using the packet ETHERCAT_MASTER_CMD_GET_BUS_SCAN_INFO_REQ.

If the master does not find any slaves within given timeout (e. g. no Ethernet link or no slaves attached) the packet is returned with an error code.

Note: This packet is provided for applications migrating from ECM V3.X.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_START_BUS_SCAN_REQ_DATA_Ttag
{
  uint32_t ulTimeout; /* in ms */
} ETHERCAT_MASTER_PACKET_START_BUS_SCAN_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_START_BUS_SCAN_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_START_BUS_SCAN_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_START_BUS_SCAN_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	0	Packet Data Length in bytes
ulCmd	UINT32	0x650020	ETHERCAT_MASTER_CMD_START_BUS_SCAN_REQ
Data			
ulTimeout	UINT32		Timeout in ms

Table 182: ETHERCAT_MASTER_CMD_START_BUS_SCAN_REQ – (Re)start the bus scan request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_START_BUS_SCAN_CNF_DATA_Ttag
{
  uint32_t ulFoundSlaves;
} ETHERCAT_MASTER_PACKET_START_BUS_SCAN_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_START_BUS_SCAN_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_START_BUS_SCAN_CNF_DATA_T tData;
} ETHERCAT_MASTER_PACKET_START_BUS_SCAN_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	4 on success or 0 in case of error	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x650021	ETHERCAT_MASTER_START_BUS_SCAN_CNF
Data			
ulFoundSlaves	UINT32		Number of slaves found during bus scan

Table 183: ETHERCAT_MASTER_CMD_START_BUS_SCAN_CNF – (Re)start the bus scan confirmation

4.18.3.4 Get bus scan results service (Legacy)

This packet provides access to the data being collected by a preceding bus scan.

The actual confirmation packet will carry the following information which is related to the **Identity** and topology:

- Vendor Id
- Product Code
- Revision Number
- Serial Number
- Port State

The following addressing schemes are used:

- *Auto-increment address* (page 18)

Note: This packet is provided for applications migrating from ECM V3.X.

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_REQ_DATA_Ttag
{
  uint16_t usAutoIncAddr;
} ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_REQ_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_REQ_T;
```

Packet description

Variable	Type	Value / range	Description
ulDest	UINT32		Destination queue-handle
ulLen	UINT32	2	Packet Data Length in bytes
ulCmd	UINT32	0x650022	`ETHERCAT_MASTER_CMD_GET_BUS_SCAN_INFO_REQ`
Data			
usAutoIncAddr	UINT16	0x0000 0xFFFF 0xFFFE …	Use the *Auto-increment address* (page 18) for addressing slaves during Legacy Bus scan.

Table 184: `ETHERCAT_MASTER_CMD_GET_BUS_SCAN_INFO_REQ` – Get results from bus scan request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_CNF_DATA_Ttag
{
  uint32_t ulVendorId;
  uint32_t ulProductCode;
  uint32_t ulRevisionNumber;
  uint32_t ulSerialNumber;
  uint32_t ulPortState;
} ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_CNF_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
  ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_CNF_DATA_T tData;
} ETHERCAT_MASTER_PACKET_GET_BUS_SCAN_INFO_CNF_T;
```

Packet description

Variable	Type	Value / range	Description
ulLen	UINT32	0 in case of error 20 otherwise	Packet Data Length in bytes
ulSta	UINT32		See section Status/Error codes overview
ulCmd	UINT32	0x650023	ETHERCAT_MASTER_CMD_GET_BUS_SCAN_INFO_CNF
Data			
ulVendorId	UINT32		Vendor Id of slave
ulProductCode	UINT32		Product Code of slave
ulRevisionNumber	UINT32		Revision Number of slave
ulSerialNumber	UINT32		Serial Number of slave
ulPortState	UINT32		For details see Packet parameter ulPortState (page 196)

Table 185: ETHERCAT_MASTER_CMD_GET_BUS_SCAN_INFO_CNF – Get results from bus scan confirmation

5 Feature Configuration via Tag List

Loadable firmware supports the feature to configure firmware parameters. During startup of the firmware, it reads the configuration parameters from the tag list of the firmware.

The firmware reads the tag list parameters

- to customize the resource allocation, and
- to configure features.

5.1 EtherCAT Master Tag List Parameter

This tag only applies, if the EtherCAT Master is configured with an ENI file (ETHERCAT.XML).

Tag list	Parameter / Tag	Value	Description
EtherCAT Master Bus State for ENI	Target Bus State	On (default)	The EtherCAT Master firmware is configured to reach the target state bus on.
		Off	The EtherCAT Master firmware is configured to stay in the target state bus off. In this state, the firmware waits for the application. The application has the possibility to set/change the sync signal for example. The application has set the bus state to on, in order to start the EtherCAT Master communication.

Table 186: EtherCAT Master Tag List Parameters

6 Status/Error codes overview

6.1 EtherCAT Master packet status codes

Hexadecimal Value	Definition Description
0x00000000	SUCCESS_HIL_OK Status ok
0xC0650001	ERR_ETHERCAT_MASTER_COMMAND_INVALID Invalid command received.
0xC0650002	ERR_ETHERCAT_MASTER_NO_LINK No link exists.
0xC0650003	ERR_ETHERCAT_MASTER_ERROR_READING_BUSCONFIG Error during reading the bus configuration.
0xC0650004	ERR_ETHERCAT_MASTER_ERROR_PARSING_BUSCONFIG Error during processing the bus configuration.
0xC0650005	ERR_ETHERCAT_MASTER_ERROR_BUSSCAN_FAILED Existing bus does not match configured bus.
0xC0650006	ERR_ETHERCAT_MASTER_NOT_ALL_SLAVES_AVAIL Not all slaves are available.
0xC0650007	ERR_ETHERCAT_MASTER_STOPMASTER_ERROR Error during Reset (stopping the master).
0xC0650008	ERR_ETHERCAT_MASTER_DEINITMASTER_ERROR Error during Reset (deinitialize the master).
0xC0650009	ERR_ETHERCAT_MASTER_CLEANUP_ERROR Error during Reset (cleanup the dynamic resources).
0xC065000A	ERR_ETHERCAT_MASTER_CRITIAL_ERROR_STATE Master is in critical error state, reset required.
0xC065000B	ERR_ETHERCAT_MASTER_INVALID_BUSCYCLETIME The requested bus cycle time is invalid.
0xC065000C	ERR_ETHERCAT_MASTER_INVALID_BROKEN_SLAVE_BEHAVIOUR_PARA Invalid parameter for broken slave behavior.
0xC065000D	ERR_ETHERCAT_MASTER_WRONG_INTERNAL_STATE Master is in wrong internal state.
0xC065000E	ERR_ETHERCAT_MASTER_WATCHDOG_TIMEOUT_EXPIRED The watchdog expired.
0xC065000F	ERR_ETHERCAT_MASTER_COE_INVALID_SLAVEID Invalid Slave ID was used for CoE.
0xC0650010	ERR_ETHERCAT_MASTER_COE_NO_RESOURCE No available resources for CoE transfer.
0xC0650011	ERR_ETHERCAT_MASTER_COE_INTERNAL_ERROR Internal error during CoE usage.
0xC0650012	ERR_ETHERCAT_MASTER_COE_INVALID_INDEX Invalid slave index requested.
0xC0650013	ERR_ETHERCAT_MASTER_COE_INVALID_COMMUNICATION_STATE Invalid bus communication state for CoE usage.
0xC0650014	ERR_ETHERCAT_MASTER_COE_FRAME_LOST Frame with CoE data is lost.

Hexadecimal Value	Definition Description
0xC0650015	ERR_ETHERCAT_MASTER_COE_TIMEOUT Timeout during CoE service.
0xC0650016	ERR_ETHERCAT_MASTER_COE_SLAVE_NOT_ADDRESSABLE Slave is not addressable (not on bus or power down?).
0xC0650017	ERR_ETHERCAT_MASTER_COE_INVALID_LIST_TYPE Invalid list type requested (during GetOdList).
0xC0650018	ERR_ETHERCAT_MASTER_COE_SLAVE_RESPONSE_TOO_BIG Data in Slave Response is too big for confirmation packet.
0xC0650019	ERR_ETHERCAT_MASTER_COE_INVALID_ACCESSBITMASK Invalid access mask selected (during GetEntryDesc).
0xC065001A	ERR_ETHERCAT_MASTER_COE_WKC_ERROR Slave Working Counter error during CoE service.
0xC065001B	ERR_ETHERCAT_MASTER_SERVICE_IN_USE The service is already in use.
0xC065001C	ERR_ETHERCAT_MASTER_INVALID_COMMUNICATION_STATE Command is not useable in this communication state.
0xC065001D	ERR_ETHERCAT_MASTER_DC_NOT_ACTIVATED Distributed Clocks must be activated for this command.
0xC065001E	ERR_ETHERCAT_MASTER_BUS_SCAN_CURRENTLY_RUNNING Bus Scan is currently running.
0xC065001F	ERR_ETHERCAT_MASTER_BUS_SCAN_TIMEOUT Bus Scan Timeout. No slave found.
0xC0650020	ERR_ETHERCAT_MASTER_BUS_SCAN_NOT_READY_YET Bus Scan is not ready yet.
0xC0650021	ERR_ETHERCAT_MASTER_BUS_SCAN_INVALID_SLAVE Invalid slave. No information available.
0xC0650022	ERR_ETHERCAT_MASTER_COE_INVALIDACCESS Slave does not allow reading or writing (CoE access).
0xC0650023	ERR_ETHERCAT_MASTER_COE_NO_MBX_SUPPORT Slave does not support a mailbox.
0xC0650024	ERR_ETHERCAT_MASTER_COE_NO_COE_SUPPORT Slave does not support CoE.
0xC0650025	ERR_ETHERCAT_MASTER_TASK_CREATION_FAILED Task could not be created during runtime.
0xC0650026	ERR_ETHERCAT_MASTER_INVALID_SLAVE_SM_CONFIGURATION The Sync Manager configuration of a slave is invalid.
0xC0650027	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_TOGGLE SDO abort code: Toggle bit not alternated.
0xC0650028	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_TIMEOUT SDO abort code: SDO protocol timed out.
0xC0650029	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_CCS_SCS SDO abort code: Client/server command specifier not valid or unknown.
0xC065002A	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_BLK_SIZE SDO abort code: Invalid block size (block mode only).
0xC065002B	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_SEQNO SDO abort code: Invalid sequence number (block mode only).

Hexadecimal Value	Definition Description
0xC065002C	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_CRC SDO abort code: CRC error (block mode only).
0xC065002D	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_MEMORY SDO abort code: Out of memory.
0xC065002E	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_ACCESS SDO abort code: Unsupported access to an object.
0xC065002F	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_WRITEONLY SDO abort code: Attempt to read a write only object.
0xC0650030	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_READONLY SDO abort code: Attempt to write a read only object.
0xC0650031	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_INDEX SDO abort code: Object does not exist in the object dictionary.
0xC0650032	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_PDO_MAP SDO abort code: Object cannot be mapped to the PDO.
0xC0650033	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_PDO_LEN SDO abort code: The number and length of the objects to be mapped would exceed PDO length.
0xC0650034	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_P_INCOMP SDO abort code: General parameter incompatibility reason.
0xC0650035	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_I_INCOMP SDO abort code: General internal incompatibility in the device.
0xC0650036	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_HARDWARE SDO abort code: Access failed due to a hardware error.
0xC0650037	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_SIZE SDO abort code: Data type does not match, length of service parameter does not match.
0xC0650038	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_SIZE1 SDO abort code: Data type does not match, service parameter too long.
0xC0650039	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_SIZE2 SDO abort code: Data type does not match, service parameter too short.
0xC065003A	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_OFFSET SDO abort code: Subindex does not exist.
0xC065003B	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_RANGE SDO abort code: Value range of parameter exceeded (only for write access).
0xC065003C	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_RANGE1 SDO abort code: Value of parameter written too high.
0xC065003D	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_RANGE2 SDO abort code: Value of parameter written too low.
0xC065003E	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_MINMAX SDO abort code: Maximum value is less than minimum value.
0xC065003F	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_GENERAL SDO abort code: General error.
0xC0650040	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_TRANSFER SDO abort code: Data cannot be transferred to or stored in the application.
0xC0650041	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_TRANSFER1 SDO abort code: Data cannot be transferred to or stored in the application because of local control.

Hexadecimal Value	Definition Description
0xC0650042	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_TRANSFER2 SDO abort code: Data cannot be transferred to or stored in the application because of the present device state.
0xC0650043	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_DICTIONARY SDO abort code: Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of a file error).
0xC0650044	ERR_ETHERCAT_MASTER_SDO_ABORTCODE_UNKNOWN SDO abort code: unknown code.
0xC0650045	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_ERROR Slave status code: Unspecified error.
0xC0650046	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVREQSTATECNG Slave status code: Invalid requested state change.
0xC0650047	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_UNKREQSTATE Slave status code: Unknown requested state.
0xC0650048	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_BOOTSTRAPNSUPP Slave status code: Bootstrap not supported.
0xC0650049	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_NOVALIDFW Slave status code: No valid firmware.
0xC065004A	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVALIDMBXCNF1 Slave status code: Invalid mailbox configuration1.
0xC065004B	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVALIDMBXCNF2 Slave status code: Invalid mailbox configuration2.
0xC065004C	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVALIDSMCNF Slave status code: Invalid sync manager configuration.
0xC065004D	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_NOVALIDIN Slave status code: No valid inputs available.
0xC065004E	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_NOVALIDOUT Slave status code: No valid outputs.
0xC065004F	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_SYNCERROR Slave status code: Synchronization error.
0xC0650050	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_SMWATCHDOG Slave status code: Sync manager watchdog.
0xC0650051	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVSMTYPES Slave status code: Invalid Sync Manager Types.
0xC0650052	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVOUTCONFIG Slave status code: Invalid Output Configuration.
0xC0650053	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVINCONFIG Slave status code: Invalid Input Configuration.
0xC0650054	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVWDCONFIG Slave status code: Invalid Watchdog Configuration.
0xC0650055	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_SLVNEEDCOLDRS Slave status code: Slave needs cold start.
0xC0650056	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_SLVNEEDINIT Slave status code: Slave needs INIT.
0xC0650057	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_SLVNEEDPREOP Slave status code: Slave needs PREOP.

Hexadecimal Value	Definition Description
0xC0650058	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_SLVNEEDSAFEOP Slave status code: Slave needs SAFEOP.
0xC0650059	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVOUTFMMUCNFG Slave status code: Invalid Output FMMU Configuration.
0xC065005A	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVINFMMUCNFG Slave status code: Invalid Input FMMU Configuration.
0xC065005B	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVDCSYNCCNFG Slave status code: Invalid DC SYNCH Configuration.
0xC065005C	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVDCLATCHCNFG Slave status code: Invalid DC Latch Configuration.
0xC065005D	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_PLLERROR Slave status code: PLL Error.
0xC065005E	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVDCIOERROR Slave status code: Invalid DC IO Error.
0xC065005F	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_INVDCTOERROR Slave status code: Invalid DC Timeout Error.
0xC0650060	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_MBX_EOE Slave status code: MBX_EOE.
0xC0650061	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_MBX_COE Slave status code: MBX_COE.
0xC0650062	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_MBX_FOE Slave status code: MBX_FOE.
0xC0650063	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_MBX_SOE Slave status code: MBX_SOE.
0xC0650064	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_MBX_VOE Slave status code: MBX_VOE.
0xC0650065	ERR_ETHERCAT_MASTER_DEVICE_STATUSCODE_OTHER Slave status code: vendor specific error code.
0xC0650066	ERR_ETHERCAT_MASTER_PREVIOUS_PORT_MISSING Slave status code: PreviousPort configuration missing in bus configuration file (outdated configurator).
0xC0650067	ERR_ETHERCAT_MASTER_CONFIG_ALREADY_STARTED Configuration already started, cannot be started again.
0xC0650068	ERR_ETHERCAT_MASTER_CONFIG_NOT_STARTED Configuration was not started before.
0xC0650069	ERR_ETHERCAT_MASTER_CONFIG_SLAVE_INDEX_ALREADY_EXISTS Slave index already exists, cannot be created again.
0xC065006A	ERR_ETHERCAT_MASTER_CONFIG_SLAVE_PHYS_ADDR_ALREADY_EXISTS Slave physical address already exists, cannot be created again.
0xC065006B	ERR_ETHERCAT_MASTER_CONFIG_SLAVE_AUTOINC_ADDR_ALREADY_EXISTS Slave auto increment address already exists, cannot be created again.
0xC065006C	ERR_ETHERCAT_MASTER_CONFIG_SLAVE_INDEX_NOT_EXISTS Slave index does not exist, must be created before.
0xC065006D	ERR_ETHERCAT_MASTER_WRONG_VALIDATE_DATA_LEN Wrong length value for validate data.
0xC065006E	ERR_ETHERCAT_MASTER_INVALID_ECAT_CMD Invalid value for EtherCAT command.

Hexadecimal Value	Definition Description
0xC065006F	ERR_ETHERCAT_MASTER_PRECONFIGURED_DATA_CURRENTLY_NOT_SUPPORTED Sending preconfigured cyclic data is currently not supported.
0xC0650070	ERR_ETHERCAT_MASTER_INVALID_STATE Invalid value for EtherCAT state.
0xC0650071	ERR_ETHERCAT_MASTER_INVALID_TRANSITION Invalid value for EtherCAT transition.
0xC0650072	ERR_ETHERCAT_MASTER_COPY_INFOS_EXCEEDED Maximum amount of copy infos exceeded.
0xC0650073	ERR_ETHERCAT_MASTER_REDUNDANCY_AND_DC_ENABLED Redundancy and Distributed clocks enabled at the same time (not possible).
0xC0650074	ERR_ETHERCAT_MASTER_NO_SLAVES_CONFIGURED At least one slave must be configured.
0xC0650075	ERR_ETHERCAT_MASTER_STATE_CHANGE_BUSY State change is currently busy.
0xC0650076	ERR_ETHERCAT_MASTER_INVALID_TARGET_PHASE Parameter target phase is invalid.

Table 187: EtherCAT Master packet status codes

6.2 EtherCAT Master V4 LLD error codes

Hexadecimal Value	Definition Description
0xC0CC0001	ERR_ECMV4_LLD_TIMEOUT ESC Register access timeout.
0xC0CC0003	ERR_ECMV4_LLD_UNSUPPORTED_COMMAND LLD: Unsupported EtherCAT telegram command.
0xC0CC0004	ERR_ECMV4_LLD_DUPLICATE_FIXED_STATION_ADDRESS LLD: Duplicate fixed station address.
0xC0CC0005	ERR_ECMV4_LLD_SII_CHECKSUM_ERROR LLD: SII Checksum Error.
0xC0CC0006	ERR_ECMV4_LLD_SII_EEPROM_LOADING_ERROR LLD: SII EEPROM Loading Error.
0xC0CC0007	ERR_ECMV4_LLD_SII_MISSING_ERROR_ACK LLD: SII Missing Error Ack.
0xC0CC0008	ERR_ECMV4_LLD_STATE_CHANGE_FAILED LLD: ESM State Change failed.
0xC0CC0009	ERR_ECMV4_LLD_UNEXPECTED_AL_STATUS LLD: Unexpected ALSTATUS.
0xC0CC000A	ERR_ECMV4_LLD_UNEXPECTED_WKC LLD: Unexpected Working Count.
0xC0CC000B	ERR_ECMV4_LLD_MAILBOX_NOT_AVAILABLE LLD: Mailbox not available.
0xC0CC000C	ERR_ECMV4_LLD_MAILBOX_MESSAGE_TOO_LARGE LLD: Mailbox message too large.
0xC0CC000D	ERR_ECMV4_LLD_CONFIGURATION_IN_PROGRESS LLD: Configuration in progress.
0xC0CC000E	ERR_ECMV4_LLD_TOO_MANY_CYCLIC_FRAMES LLD: Too many cyclic frames.
0xC0CC000F	ERR_ECMV4_LLD_CYCLIC_FRAME_EXCEEDS_MTU LLD: SII Checksum Error.
0xC0CC0010	ERR_ECMV4_LLD_INVALID_CYCLIC_TELEGRAM_CONFIG LLD: Invalid cyclic telegram config.
0xC0CC0011	ERR_ECMV4_LLD_BUILDING_COPY_ROUTINES_FAILED LLD: Building copy routines failed.
0xC0CC0012	ERR_ECMV4_LLD_UNSUPPORTED_SLAVE_STATION_ADDRESS LLD: Unsupported slave station address.
0xC0CC0013	ERR_ECMV4_LLD_STATION_ADDRESS_NOT_ALLOWED LLD: Station address not allowed.
0xC0CC0014	ERR_ECMV4_LLD_INVALID_STD_TX_MBX_PHYS_OFFSET LLD: Invalid Std TxMbx PhysOffset.
0xC0CC0015	ERR_ECMV4_LLD_INVALID_STD_RX_MBX_PHYS_OFFSET LLD: Invalid Std RxMbx PhysOffset.
0xC0CC0016	ERR_ECMV4_LLD_INVALID_BOOT_TX_MBX_PHYS_OFFSET LLD: Invalid Boot TxMbx PhysOffset.
0xC0CC0017	ERR_ECMV4_LLD_INVALID_BOOT_RX_MBX_PHYS_OFFSET LLD: Invalid Boot RxMbx PhysOffset.

Hexadecimal Value	Definition Description
0xC0CC0018	ERR_ECMV4_LLD_INVALID_STD_TX_MBX_SM_NO LLD: Invalid Std TxMbx SyncManager number.
0xC0CC0019	ERR_ECMV4_LLD_INVALID_STD_RX_MBX_SM_NO LLD: Invalid Std RxMbx SyncManager number.
0xC0CC001A	ERR_ECMV4_LLD_INVALID_BOOT_TX_MBX_SM_NO LLD: Invalid Boot TxMbx SyncManager number.
0xC0CC001B	ERR_ECMV4_LLD_INVALID_BOOT_RX_MBX_SM_NO LLD: Invalid Boot RxMbx SyncManager number.
0xC0CC001C	ERR_ECMV4_LLD_UNCONFIGURED_SLAVE_STATION_ADDRESS LLD: Unconfigured slave station address.
0xC0CC001D	ERR_ECMV4_LLD_WRONG_SLAVE_STATE LLD: Wrong slave state.
0xC0CC001E	ERR_ECMV4_LLD_CYCLE_TIME_TOO_SMALL LLD: Cycle time too short.
0xC0CC001F	ERR_ECMV4_LLD_REPETITION_COUNT_NOT_SUPPORTED LLD: Repetition count not supported.
0xC0CC0020	ERR_ECMV4_LLD_INVALID_CALLBACK_TYPE LLD: Invalid callback type.
0xC0CC0021	ERR_ECMV4_LLD_INVALID_CYCLE_MULTIPLIER LLD: Invalid cycle multiplier.
0xC0CC0022	ERR_ECMV4_LLD_UNKNOWN_ERROR LLD: Unknown error.
0xC0CC0023	ERR_ECMV4_LLD_INVALID_REG_LENGTH LLD: Invalid register length.
0xC0CC0024	ERR_ECMV4_LLD_INVALID_PARAMETER LLD: Invalid parameter.
0xC0CC0025	ERR_ECMV4_LLD_IRQ_NOT_AVAILABLE LLD: IRQ not available.
0xC0CC0026	ERR_ECMV4_LLD_IOMEM_IRQ_NOT_AVAILABLE LLD: I/O Mem or IRQ not available.
0xC0CC0027	ERR_ECMV4_LLD_HW_INIT_FAILED LLD: Hardware init failed.
0xC0CC0028	ERR_ECMV4_LLD_MUTEX_CREATION_FAILED LLD: Mutex creation failed.
0xC0CC0029	ERR_ECMV4_LLD_DC_RX_LATCH_COMMAND_REQUIRED_FOR_DC LLD: DC Rx-Latch command required for DC.
0xC0CC002A	ERR_ECMV4_LLD_TX_PROCESS_IMAGE_EXCEEDED LLD: Tx process image exceeded.
0xC0CC002B	ERR_ECMV4_LLD_RX_PROCESS_IMAGE_EXCEEDED LLD: Rx process image exceeded.
0xC0CC002C	ERR_ECMV4_LLD_MBX_STATE_IMAGE_EXCEEDED LLD: MbxState image exceeded.
0xC0CC002D	ERR_ECMV4_LLD_RESULT_DUPLICATE_BWR_RX_LATCH_CMD LLD: Duplicate BWR RX Latch Cmd.
0xC0CC002E	ERR_ECMV4_LLD_RESULT_DUPLICATE_EXT_SYSTIME_CONTROL_CMD LLD: Duplicate ExtSync Control Cmd.

Hexadecimal Value	Definition Description
0xC0CC002F	ERR_ECMV4_LLD_CC_PROCESS_IMAGE_EXCEEDED LLD: Cross communication process image exceeded.
0xC0CC0030	ERR_ECMV4_LLD_SII_TIMEOUT LLD: SII Timeout.
0xC0CC0031	ERR_ECMV4_LLD_BUS_NOT_ENABLED LLD: Bus not enabled.

Table 188: EtherCAT Master V4 LLD error codes

6.3 EtherCAT Master V4 EMC error codes

Hexadecimal Value	Definition Description
0x40CD0017	ERR_ECMV4_EMC_BUS_IS_OFF Bus is off.
0xC0CD0001	ERR_ECMV4_EMC_REQUEST_DESTINATION_PROBLEM Request destination problem.
0xC0CD0002	ERR_ECMV4_EMC_INVALID_SLAVE_STATION_ADDRESS Invalid slave station address.
0xC0CD0003	ERR_ECMV4_EMC_CONFIGURATION_BUFFER_IS_OPEN Configuration Buffer is open.
0xC0CD0004	ERR_ECMV4_EMC_WRONG_STATE_FOR_RECONFIGURATION Wrong state for reconfiguration.
0xC0CD0005	ERR_ECMV4_EMC_CONFIGURATION_BUFFER_IS_NOT_OPEN Configuration Buffer is not open.
0xC0CD0006	ERR_ECMV4_EMC_SLAVE_STATION_ADDRESS_ALREADY_IN_CONFIG Slave station address is already in configuration.
0xC0CD0007	ERR_ECMV4_EMC_INVALID_STD_MBX_PARAMETERS Invalid Std Mbx parameters.
0xC0CD0008	ERR_ECMV4_EMC_INVALID_BOOT_MBX_PARAMETERS Invalid Boot Mbx parameters.
0xC0CD0009	ERR_ECMV4_EMC_STD_MBX_SM_ARE_OVERLAPPING Std Mbx SyncManagers are overlapping.
0xC0CD000A	ERR_ECMV4_EMC_BOOT_MBX_SM_ARE_OVERLAPPING Boot Mbx SyncManagers are overlapping.
0xC0CD000B	ERR_ECMV4_EMC_SM_PARAMS_ALREADY_ADDED SyncManager params already added.
0xC0CD000C	ERR_ECMV4_EMC_INVALID_SM_NUMBER Invalid SyncManager number.
0xC0CD000D	ERR_ECMV4_EMC_FMMU_PARAMS_ALREADY_ADDED FMMU params already added.
0xC0CD000E	ERR_ECMV4_EMC_INVALID_FMMU_NUMBER Invalid FMMU number.
0xC0CD000F	ERR_ECMV4_EMC_INVALID_MIN_STATE Invalid Min State.

Hexadecimal Value	Definition Description
0xC0CD0010	ERR_ECMV4_EMC_CYCLE_FRAME_AMOUNT_EXCEEDED Cycle frame amount exceeded.
0xC0CD0011	ERR_ECMV4_EMC_INVALID_CYCLIC_FRAME_IN_CONFIGURATION Invalid cyclic frame in configuration.
0xC0CD0012	ERR_ECMV4_EMC_CYCLE_FRAME_INDEX_NOT_VALID Cycle frame index not valid.
0xC0CD0013	ERR_ECMV4_EMC_INVALID_TELEGRAM_LENGTH Invalid telegram length.
0xC0CD0014	ERR_ECMV4_EMC_CYCLE_FRAME_LENGTH_EXCEEDED Cycle frame length exceeded.
0xC0CD0015	ERR_ECMV4_EMC_AMOUNT_OF_TELEGRAMS_IN_CYCLIC_FRAMES_EXCEEDED Amount of telegrams in cyclic frame exceeded.
0xC0CD0016	ERR_ECMV4_EMC_STATE_CHANGE_IN_PROGRESS State change in progress.
0xC0CD0018	ERR_ECMV4_EMC_TOO_MANY_SLAVES_GIVEN Too many slaves given.
0xC0CD0019	ERR_ECMV4_EMC_DUPLICATE_STATION_ADDRESS_IN_LIST Duplicate station address in list.
0xC0CD001B	ERR_ECMV4_EMC_CONFIGURATION_DATA_INCORRECT Configuration data incorrect.
0xC0CD001C	ERR_ECMV4_EMC_VENDORID_MISMATCH Vendor Id mismatch.
0xC0CD001D	ERR_ECMV4_EMC_PRODUCTCODE_MISMATCH Product Code mismatch.
0xC0CD001E	ERR_ECMV4_EMC_REVISIONNO_MISMATCH Revision Number mismatch.
0xC0CD001F	ERR_ECMV4_EMC_SERIALNO_MISMATCH Serial Number mismatch.
0xC0CD0020	ERR_ECMV4_EMC_LOST_CONNECTION Lost Connection.
0xC0CD0021	ERR_ECMV4_EMC_UNKNOWN_STATE_CHANGE_HAPPENED Unknown state change happened.
0xC0CD0022	ERR_ECMV4_EMC_UNEXPECTED_STATE_CHANGE_HAPPENED Unexpected state change happened.
0xC0CD0023	ERR_ECMV4_EMC_SLAVE_CHANGED_STATE Slave changed state.
0xC0CD0024	ERR_ECMV4_EMC_FILE_PROVIDER_INIT_FAILED File Provider init failed.
0xC0CD0026	ERR_ECMV4_EMC_DC_RX_TIMESTAMP_ERROR DC Rx Timestamp Error.
0xC0CD0027	ERR_ECMV4_EMC_DC_MASTER_PORT_TIMESTAMP_ERROR DC Master Port Timestamp Error.
0xC0CD0028	ERR_ECMV4_EMC_INVALID_SLAVE_INDEX Invalid slave index.
0xC0CD0029	ERR_ECMV4_EMC_WRONG_MASTER_STATE Wrong master state.

Hexadecimal Value	Definition Description
0xC0CD002A	ERR_ECMV4_EMC_INVALID_TRANSFER_ID Invalid transfer id.
0xC0CD002B	ERR_ECMV4_EMC_INVALID_SEGMENTATION Invalid segmentation.
0xC0CD002C	ERR_ECMV4_EMC_EOE_IP_PARAMS_ALREADY_ADDED EoE IP Parameter already added.
0xC0CD002D	ERR_ECMV4_EMC_EOE_SUPPORT_NOT_AVAILABLE EoE Support not available.
0xC0CD002E	ERR_ECMV4_EMC_END_CONFIGURATION_IN_PROGRESS End configuration in progress.
0xC0CD002F	ERR_ECMV4_EMC_WRONG_STATE_FOR_RECONFIGURATION_BUS_IS_ON Wrong state for reconfiguration: Bus Is On.
0xC0CD0030	ERR_ECMV4_EMC_WRONG_STATE_FOR_RECONFIGURATION_BUS_SCAN_ACTIVE Wrong state for reconfiguration: Bus Scan is active.
0xC0CD0031	ERR_ECMV4_EMC_WRONG_STATE_FOR_RECONFIGURATION_IN_PROGRESS_TO_BUSOFF Wrong state for reconfiguration: Bus Off in progress.
0xC0CD0032	ERR_ECMV4_EMC_NO_DIAG_ENTRY_AVAILABLE No diag entry available.
0xC0CD0033	ERR_ECMV4_EMC_SLAVE_SYNC_PARAMS_NOT_POSSIBLE_WITHOUT_WORKING_DC Slave Sync parameters are not possible without working DC.
0xC0CD0034	ERR_ECMV4_EMC_MANDATORY_SLAVE_MISSING Mandatory slave missing.
0xC0CD0035	ERR_ECMV4_EMC_WRONG_SLAVE_AT_POSITION Wrong slave at position.
0xC0CD0036	ERR_ECMV4_EMC_NO_DC_REF_CLOCK No DC Reference clock available.
0xC0CD0038	ERR_ECMV4_EMC_INVALID_DC_REF_CLOCK Invalid DC reference clock.
0xC0CD0039	ERR_ECMV4_EMC_COE_SUPPORT_NOT_AVAILABLE CoE Support not available.
0xC0CD003A	ERR_ECMV4_EMC_SOE_SUPPORT_NOT_AVAILABLE SoE Support not available.
0xC0CD003B	ERR_ECMV4_EMC_FOE_SUPPORT_NOT_AVAILABLE FoE Support not available.
0xC0CD003C	ERR_ECMV4_EMC_AOE_SUPPORT_NOT_AVAILABLE AoE Support not available.
0x40CD003E	ERR_ECMV4_EMC_RECONNECTED Slave reconnected.
0x80CD003F	ERR_ECMV4_EMC_DC_STOPPED DC ARMW/FRMW has been stopped.
0xC0CD0040	ERR_ECMV4_EMC_STOPPED_DUE_SYNC_ERROR Stopped due sync error.
0xC0CD0041	ERR_ECMV4_EMC_MANDATORY_SLAVE_NOT_IN_OP Mandatory slave is not in OP.
0xC0CD0042	ERR_ECMV4_EMC_BUS_CYCLE_TIME_NOT_POSSIBLE Bus cycle time not possible.

Hexadecimal Value	Definition Description
0xC0CD0043	ERR_ECMV4_EMC_TOPOLOGY_ERROR_DETECTED Topology error detected.
0xC0CD0044	ERR_ECMV4_EMC_TOPOLOGY_MISMATCH_DETECTED Topology mismatch detected.
0xC0CD0045	ERR_ECMV4_EMC_NO_VALID_TOPOLOGY_CONFIGURATION_DATA No valid topology configuration data.
0xC0CD0046	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT0 Unexpected slave at port 0.
0xC0CD0047	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT1 Unexpected slave at port 1.
0xC0CD0048	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT2 Unexpected slave at port 2.
0xC0CD0049	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT3 Unexpected slave at port 3.
0xC0CD004A	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_CONNECTED Unexpected slave connected.
0xC0CD004B	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT0 Missing slave at port 0.
0xC0CD004C	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT1 Missing slave at port 1.
0xC0CD004D	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT2 Missing slave at port 2.
0xC0CD004E	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT3 Missing slave at port 3.
0xC0CD004F	ERR_ECMV4_EMC_SLAVE_NOT_CHECKED Slave not checked.
0xC0CD0050	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT0_1 Unexpected slaves at ports 0 and 1.
0xC0CD0051	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT0_2 Unexpected slaves at ports 0 and 2.
0xC0CD0052	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT0_3 Unexpected slaves at ports 0 and 3.
0xC0CD0053	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT1_2 Unexpected slaves at ports 1 and 2.
0xC0CD0054	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT1_3 Unexpected slaves at ports 1 and 3.
0xC0CD0055	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT2_3 Unexpected slaves at ports 2 and 3.
0xC0CD0056	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT0_1_2 Unexpected slaves at ports 0, 1 and 2.
0xC0CD0057	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT0_1_3 Unexpected slaves at ports 0, 1 and 3.
0xC0CD0058	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT0_2_3 Unexpected slaves at ports 0, 2 and 3.
0xC0CD0059	ERR_ECMV4_EMC_UNEXPECTED_SLAVE_AT_PORT1_2_3 Unexpected slaves at ports 1, 2 and 3.

Hexadecimal Value	Definition Description
0xC0CD005A	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT0_1 Missing slaves at ports 0 and 1.
0xC0CD005B	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT0_2 Missing slaves at ports 0 and 2.
0xC0CD005C	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT0_3 Missing slaves at ports 0 and 3.
0xC0CD005D	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT1_2 Missing slaves at ports 1 and 2.
0xC0CD005E	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT1_3 Missing slaves at ports 1 and 3.
0xC0CD005F	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT2_3 Missing slaves at ports 2 and 3.
0xC0CD0060	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT0_1_2 Missing slaves at ports 0, 1 and 2.
0xC0CD0061	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT0_1_3 Missing slaves at ports 0, 1 and 3.
0xC0CD0062	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT0_2_3 Missing slaves at ports 0, 2 and 3.
0xC0CD0063	ERR_ECMV4_EMC_MISSING_SLAVE_AT_PORT1_2_3 Missing slaves at ports 1, 2 and 3.
0xC0CD0065	ERR_ECMV4_EMC_HC_PARTICIPANT_NOT_ALLOWED_IN_MANDATORY_SLAVE_LIST Hot Connect participant is not allowed in mandatory slave list.
0xC0CD0066	ERR_ECMV4_EMC_HC_PARTICIPANT_NOT_ALLOWED_IN_MULTIPLE_HC_GROUPS Hot Connect participant is not allowed in multiple Hot Connect groups.
0xC0CD0067	ERR_ECMV4_EMC_HC_GROUP_HEAD_IS_NOT_LISTED_FOR_HC_DETECTION Hot Connect Group Head is not listed for Hot Connect detection.
0xC0CD0068	ERR_ECMV4_EMC_DC_SETUP_CALCULATION_ERROR DC Setup calculation error.
0xC0CD0069	ERR_ECMV4_EMC_NON_DC_SLAVE_MORE_THAN_2_PORTS_IN_DC_SETUP A Slave without DC support has more than 2 ports in DC setup.
0xC0CD006A	ERR_ECMV4_EMC_HC_GROUP_CONTAINS_NOT_CONFIGURED_SLAVE Hot Connect Group contains not configured slave.
0xC0CD006B	ERR_ECMV4_EMC_ALCONTROL_TIMEOUT ALControl Timeout.
0xC0CD006C	ERR_ECMV4_EMC_DC_MEASUREMENT_ERROR DC Measurement Error.
0xC0CD006D	ERR_ECMV4_EMC_RX_DESTINATION_EXCEEDS_RX_IMAGE_SIZE Rx Process data destination exceeds Rx Image Size.
0xC0CD006E	ERR_ECMV4_EMC_TX_SOURCE_EXCEEDS_TX_IMAGE_SIZE Tx Process data source exceeds Tx image Size.
0xC0CD006F	ERR_ECMV4_EMC_WCSTATEBIT_EXCEEDS_RX_IMAGE_SIZE WcState bit exceeds Rx Image Size.
0xC0CD0070	ERR_ECMV4_EMC_WKC_MAPPING_EXCEEDS_RX_IMAGE_SIZE WKC mapping exceeds Rx Image Size.
0xC0CD0071	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT0 DC Rx-Latch Error at Port 0.

Hexadecimal Value	Definition Description
0xC0CD0072	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT1 DC Rx-Latch Error at Port 1.
0xC0CD0073	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT2 DC Rx-Latch Error at Port 2.
0xC0CD0074	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT3 DC Rx-Latch Error at Port 3.
0xC0CD0075	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1 DC Rx-Latch Error at Ports 0 and 1.
0xC0CD0076	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT0_2 DC Rx-Latch Error at Ports 0 and 2.
0xC0CD0077	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT0_3 DC Rx-Latch Error at Ports 0 and 3.
0xC0CD0078	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT1_2 DC Rx-Latch Error at Ports 1 and 2.
0xC0CD0079	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT1_3 DC Rx-Latch Error at Ports 1 and 3.
0xC0CD007A	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT2_3 DC Rx-Latch Error at Ports 2 and 3.
0xC0CD007B	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1_2 DC Rx-Latch Error at Ports 0, 1 and 2.
0xC0CD007C	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1_3 DC Rx-Latch Error at Ports 0, 1 and 3.
0xC0CD007D	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT0_2_3 DC Rx-Latch Error at Ports 0, 2 and 3.
0xC0CD007E	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT1_2_3 DC Rx-Latch Error at Ports 1, 2 and 3.
0xC0CD007F	ERR_ECMV4_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1_2_3 DC Rx-Latch Error at Ports 0, 1, 2 and 3.
0xC0CD0080	ERR_ECMV4_EMC_ASSIGN_PDO_IS_MISSING_PDO_MAPPING AssignPDO is missing PDO-Mapping.
0xC0CD0081	ERR_ECMV4_EMC_EXT_SYNC_OBJ_IS_NOT_MAPPED_TO_SAME_SM ExtSync object is not mapped to same SyncManager.
0xC0CD0082	ERR_ECMV4_EMC_DUPLICATE_EXT_SYNC_OBJ Duplicate ExtSync object.
0xC0CD0083	ERR_ECMV4_EMC_UNSUPPORTED_EXT_SYNC_OBJ_RECORD Unsupported ExtSync object record.
0xC0CD0084	ERR_ECMV4_EMC_UNSUPPORTED_MAPPING_OF_EXT_SYNC_OBJ_RECORD Unsupported mapping of ExtSync object.
0xC0CD0085	ERR_ECMV4_EMC_MISSING_MAPPING_OF_EXT_SYNC_OBJ_RECORD Missing mapping of ExtSync object.
0xC0CD0086	ERR_ECMV4_EMC_EXT_SYNC_OBJ_IS_NOT_MAPPED_TO_SAME_FMMU ExtSync object is not mapped to same FMMU.
0xC0CD0087	ERR_ECMV4_EMC_EXT_SYNC_OBJ_INTERNAL_ERROR Internal error encountered with ExtSync object.
0xC0CD0088	ERR_ECMV4_EMC_EXT_SYNC_OBJ_IS_NOT_MAPPED_IN_ONE_CYCLIC_CMD ExtSync object is not mapped in one cyclic command.

Hexadecimal Value	Definition Description
0xC0CD0089	ERR_ECMV4_EMC_UNSUPPORTED_FMMU_MAPPING_OF_EXT_SYNC_OBJ_RECORD Unsupported FMMU mapping of Ext Sync object.
0xC0CD008A	ERR_ECMV4_EMC_EXT_SYNC_REQUIRES_ADJUST_EXT_SYNC_CMD ExtSync requires Adjust Ext Sync Cmd.
0xC0CD008B	ERR_ECMV4_EMC_EXT_SYNC_CMD_DOES_NOT_MATCH_XRMW_CMD ExtSync command does not match xRMW command.
0xC0CD008C	ERR_ECMV4_EMC_EXT_SYNC_REQUIRES_XRMW_CMD ExtSync command requires xRMW command.
0xC0CD008D	ERR_ECMV4_EMC_EXPLICIT_DEV_IDENT_FAILED_ALSTATUS Explicit Device Identification failed (ALSTATUS).
0xC0CD008E	ERR_ECMV4_EMC_EXPLICIT_DEV_IDENT_FAILED_REG Explicit Device Identification failed (register).
0xC0CD008F	ERR_ECMV4_EMC_COPY_INFOS_FOUND_AT_UNMAPPED_RECEIVE_DATA CopyInfos found at unmapped receive data.
0xC0CD0090	ERR_ECMV4_EMC_COPY_INFO_RECEIVE_DATA_AREA_NOT_MATCHING CopyInfo receive data area is not matching.
0xC0CD0091	ERR_ECMV4_EMC_SDO_UPLOAD_TOO_LONG SDO Upload data is too long.
0xC0CD0092	ERR_ECMV4_EMC_SDO_UPLOAD_TOO_SHORT SDO Upload data is too short.
0xC0CD0093	ERR_ECMV4_EMC_SDO_UPLOAD_COMPARE_DOES_NOT_MATCH_EXPECTATION SDO Upload compare does not match expectation.
0xC0CD0094	ERR_ECMV4_EMC_SOE_READ_TOO_LONG SoE Read data too long.
0xC0CD0095	ERR_ECMV4_EMC_SOE_UPLOAD_TOO_SHORT SoE Read data too short.
0xC0CD0096	ERR_ECMV4_EMC_SOE_READ_COMPARE_DOES_NOT_MATCH_EXPECTATION SoE Read compare does not match expectation.
0xC0CD0097	ERR_ECMV4_EMC_REG_INITCMD_COMPARE_DOES_NOT_MATCH_EXPECTATION Register InitCmd compare does not match expectation.
0xC0CD0098	ERR_ECMV4_EMC_REDUNDANCY_PORT_ONLY_POSSIBLE_ONCE Redundancy port only possible once.
0xC0CD0099	ERR_ECMV4_EMC_STARTUP_SCAN_SII_FAILED Startup scan of SII failed.
0xC0CD009A	ERR_ECMV4_EMC_STARTUP_VERIFY_SII_FAILED Startup verify of SII failed.
0xC0CD009B	ERR_ECMV4_EMC_MAIN_PORT_NOT_CONNECTED Main port not connected.
0xC0CD009C	ERR_ECMV4_EMC_BUS_SCAN_TOO_MANY_SLAVES Bus Scan: Too many slaves.
0xC0CD009D	ERR_ECMV4_EMC_BUS_SCAN_SPLIT_RING_NOT_SUPPORTED Bus Scan: Split ring not supported.
0xC0CD009E	ERR_ECMV4_EMC_BUS_SHUTDOWN Bus Shutdown.
0xC0CD009F	ERR_ECMV4_EMC_MASTER_ADDRESS_NOT_ALLOWED_AS_STATION_ADDRESS Master address not allowed as station address.

Hexadecimal Value	Definition Description
0xC0CD00A0	ERR_ECMV4_EMC_FIRST_STATION_HAS_INVALID_PORT_0 First Station has invalid Port 0.
0xC0CD00A1	ERR_ECMV4_EMC_STATION_HAS_INVALID_PORT Station has invalid port.
0xC0CD00A2	ERR_ECMV4_EMC_STATION_HAS_NOT_LISTED_STATION_ADDRESS_IN_PORT Station has not listed station address in port.
0xC0CD00A3	ERR_ECMV4_EMC_PORT_CONNECTION_BETWEEN_STATIONS_DOES_NOT_MATCH Port connection between stations does not match.
0xC0CD00A4	ERR_ECMV4_EMC_STATION_HAS_ALREADY_USED_STATION_ADDRESS_IN_PORT Station has already used station address in port.
0xC0CD00A5	ERR_ECMV4_EMC_INVALID_SM_PHYS_START_ADDRESS Invalid SyncMan physical start address.
0xC0CD00A6	ERR_ECMV4_EMC_DC_TOPOLOGY_ON_REDUNDANCY_PORT_NOT_SUPPORTED DC Topology on redundancy port not supported.
0xC0CD00A7	ERR_ECMV4_EMC_SM_ASSIGN_PDO_ALREADY_ADDED SM-AssignPDO already added.
0xC0CD00A8	ERR_ECMV4_EMC_DC_BASE_SYNC_OFFSET_PERCENTAGE_OUT_OF_RANGE DC Base Sync Offset out of range.

Table 189: EtherCAT Master V4 EMC error codes

6.4 EtherCAT Master V4 AoE error codes

Hexadecimal Value	Definition Description
0xC0CE0006	ERR_ECMV4_AOE_TARGET_PORT_NOT_FOUND AoE: Target Port not found.
0xC0CE0007	ERR_ECMV4_AOE_TARGET_MACHINE_NOT_FOUND AoE: Target Machine not found.
0xC0CE0701	ERR_ECMV4_AOE_SERVICE_NOT_SUPPORTED AoE: Service not supported.
0xC0CE0702	ERR_ECMV4_AOE_INVALID_INDEX_GROUP AoE: Invalid IndexGroup.
0xC0CE0703	ERR_ECMV4_AOE_INVALID_INDEX_OFFSET AoE: Invalid IndexOffset.
0xC0CE0704	ERR_ECMV4_AOE_INVALID_ACCESS AoE: Invalid access.
0xC0CE0705	ERR_ECMV4_AOE_INVALID_SIZE AoE: Invalid size.
0xC0CE0706	ERR_ECMV4_AOE_INVALID_DATA AoE: Invalid data.
0xC0CE0707	ERR_ECMV4_AOE_NOTREADY AoE: Not ready.
0xC0CE0708	ERR_ECMV4_AOE_BUSY AoE: Busy.
0xC0CE070C	ERR_ECMV4_AOE_NOT_FOUND AoE: Not Found.
0xC0CE070E	ERR_ECMV4_AOE_INCOMPATIBLE AoE: Incompatible.
0xC0CE0712	ERR_ECMV4_AOE_WRONG_SLAVE_SATTE AoE: Wrong slave state.
0xC0CE0718	ERR_ECMV4_AOE_NOTINIT AoE: Not initialized.
0xC0CE0719	ERR_ECMV4_AOE_DEVICE_TIMEOUT AoE: Device Timeout.
0xC0CEF001	ERR_ECMV4_AOE_SHUTTING_DOWN AoE: Shutting down.
0xC0CEF002	ERR_ECMV4_AOE_INITIALIZATION_ERROR AoE: Initialization Error.
0xC0CEF003	ERR_ECMV4_AOE_INVALID_TRANSFER_HANDLE AoE: Invalid transfer handle.
0xC0CEF004	ERR_ECMV4_AOE_INVALID_TRANSFER_STATE AoE: Invalid transfer state.
0xC0CEF005	ERR_ECMV4_AOE_PROTOCOL_TIMEOUT AoE: Protocol Timeout.
0xC0CEF006	ERR_ECMV4_AOE_TRANSFER_SEGMENT_TOO_LONG AoE: Transfer Segment too long.
0xC0CEF007	ERR_ECMV4_AOE_NO_MAILBOX_AVAILABLE AoE: No mailbox available.

Hexadecimal Value	Definition Description
0xC0CEF008	ERR_ECMV4_AOE_RECONFIGURATION_IN_PROGRESS AoE: Reconfiguration in progress.
0xC0CEF009	ERR_ECMV4_AOE_INVALID_SLAVE_STATION_ADDRESS AoE: Invalid slave station address.
0xC0CEF00A	ERR_ECMV4_AOE_TRANSFER_ABORTED AoE: Transfer aborted.
0xC0CEF00B	ERR_ECMV4_AOE_REQUEST_DESTINATION_PROBLEM AoE: Request destination problem.
0xC0CEF00C	ERR_ECMV4_AOE_DUPLICATE_NETID AoE: Duplicate Net ID.
0xC0CEF00D	ERR_ECMV4_AOE_INVALID_NETID_HANDLE AoE: Invalid Net ID handle.
0xC0CEF00E	ERR_ECMV4_AOE_CONFIGURATION_IS_NOT_OPEN AoE: Configuration is not open.
0xC0CEF00F	ERR_ECMV4_AOE_CONFIGURATION_IS_ALREADY_OPEN AoE: Configuration is already open.
0xC0CEF010	ERR_ECMV4_AOE_CLIENT_INVALID_TRANSFER_HANDLE AoE Client: Invalid transfer handle.
0xC0CEF011	ERR_ECMV4_AOE_CLIENT_INVALID_TRANSFER_STATE AoE Client: Invalid transfer state.
0xC0CEF012	ERR_ECMV4_AOE_CLIENT_TRANSFER_ABORTED AoE Client: Transfer aborted.
0xC0CEF013	ERR_ECMV4_AOE_CLIENT_PROTOCOL_TIMEOUT AoE Client: Protocol Timeout.
0xC0CEF014	ERR_ECMV4_AOE_UNKNOWN_RETURN_CODE AoE: Unknown return code.
0xC0CEF015	ERR_ECMV4_AOE_CLIENT_UNKNOWN_AOE_ERROR AoE Client: Unknown AoE Error.
0xC0CEF016	ERR_ECMV4_AOE_CLIENT_TRANSFER_SEGMENT_TOO_LONG AoE Client: Transfer segment too long.
0xC0CEF017	ERR_ECMV4_AOE_CLIENT_IS_INITIALIZING AoE Client: Is initializing.
0xC0CEF018	ERR_ECMV4_AOE_CLIENT_REQUEST_DESTINATION_PROBLEM AoE Client: Request Destination Problem.
0xC0CEF019	ERR_ECMV4_AOE_CLIENT_MAX_SEGMENT_BYTES_TOO_LOW_FOR_FIRST_SEGMENT AoE Client: Max segment bytes too low for first segment.

Table 190: EtherCAT Master V4 AoE error codes

6.5 EtherCAT Master V4 CoE error codes

Hexadecimal Value	Definition Description
0xC0CF0001	ERR_ECMV4_COE_INITIALIZATION_ERROR CoE Initialization Error.
0xC0CF0002	ERR_ECMV4_COE_INVALID_TRANSFER_HANDLE CoE Invalid transfer handle.
0xC0CF0003	ERR_ECMV4_COE_NO_MAILBOX_AVAILABLE CoE Mailbox not available.
0xC0CF0004	ERR_ECMV4_COE_INVALID_TRANSFER_STATE CoE Invalid transfer state.
0xC0CF0005	ERR_ECMV4_COE_TRANSFER_SEGMENT_TOO_LONG CoE Transfer-Segment too long.
0xC0CF0006	ERR_ECMV4_COE_SHUTTING_DOWN CoE Shutting Down.
0xC0CF0007	ERR_ECMV4_COE_MAX_TOTAL_BYTES_SMALLER_THAN_ACTUAL_TOTAL_BYTES CoE Max Total Bytes is smaller than Actual Total Bytes.
0xC0CF0008	ERR_ECMV4_COE_MAILBOX_TRANSMIT_FAILED CoE Mailbox transmit failed.
0xC0CF0009	ERR_ECMV4_COE_TRANSFER_ABORTED CoE Transfer aborted.
0xC0CF000A	ERR_ECMV4_COE_SDOINFO_INITIALIZATION_ERROR CoE SDOINFO Initialization error.
0xC0CF000B	ERR_ECMV4_COE_WRONG_SLAVE_STATE CoE: Wrong slave state.
0xC0CF000C	ERR_ECMV4_COE_PROTOCOL_ERROR CoE Protocol Error.
0xC0CF000D	ERR_ECMV4_NO_AOE_AVAILABLE CoE: No AoE available.
0xC0CF000E	ERR_ECMV4_COE_REQUEST_DESTINATION_PROBLEM CoE: Request destination problem.
0xC0CF000F	ERR_ECMV4_COE_INVALID_SLAVE_STATION_ADDRESS CoE: Invalid slave station address.
0xC0CF8000	ERR_ECMV4_COE_ABORTCODE_TOGGLE_BIT_NOT_ALTERNATED Toggle bit was not changed.
0xC0CF8001	ERR_ECMV4_COE_ABORTCODE_COMMAND_SPECIFIER_NOT_VALID Client/Server command specifier not valid or unknown.
0xC0CF8002	ERR_ECMV4_COE_ABORTCODE_PROTOCOL_TIMEOUT SDO protocol timeout.
0xC0CF8003	ERR_ECMV4_COE_ABORTCODE_OUT_OF_MEMORY Out of memory.
0xC0CF8004	ERR_ECMV4_COE_ABORTCODE_UNSUPPORTED_ACCESS Unsupported access to an object.
0xC0CF8005	ERR_ECMV4_COE_ABORTCODE_OBJECT_IS_WRITE_ONLY Attempt to read a write only object.
0xC0CF8006	ERR_ECMV4_COE_ABORTCODE_OBJECT_IS_READ_ONLY Attempt to write to a read only object.

Hexadecimal Value	Definition Description
0xC0CF8007	ERR_ECMV4_COE_ABORTCODE_SUBINDEX_CANNOT_BE_WRITTEN_SI0_NZ Subindex cannot be written, Subindex 0 must be 0 for write access.
0xC0CF8008	ERR_ECMV4_COE_ABORTCODE_COMPLETE_ACCESS_NOT_SUPPORTED Complete Access not supported.
0xC0CF8009	ERR_ECMV4_COE_ABORTCODE_OBJECT_LENGTH_EXCEEDS_MAILBOX_SIZE Object length exceeds mailbox size.
0xC0CF800A	ERR_ECMV4_COE_ABORTCODE_OBJECT_MAPPED_TO_RXPDO_NO_WRITE Object mapped to RxPDO. SDO Download blocked.
0xC0CF800B	ERR_ECMV4_COE_ABORTCODE_OBJECT_DOES_NOT_EXIST The object does not exist in the object dictionary.
0xC0CF800C	ERR_ECMV4_COE_ABORTCODE_OBJECT_CANNOT_BE_PDO_MAPPED The object cannot be mapped into the PDO.
0xC0CF800D	ERR_ECMV4_COE_ABORTCODE_PDO_LENGTH_WOULD_EXCEED The number and length of the objects to be mapped would exceed the PDO length.
0xC0CF800E	ERR_ECMV4_COE_ABORTCODE_GEN_PARAM_INCOMPATIBILITY General parameter incompatibility reason.
0xC0CF800F	ERR_ECMV4_COE_ABORTCODE_ACCESS_FAILED_DUE_TO_HW_ERROR Access failed due to a hardware error.
0xC0CF8010	ERR_ECMV4_COE_ABORTCODE_DATATYPE_DOES_NOT_MATCH Data type does not match, length of service parameter does not match.
0xC0CF8011	ERR_ECMV4_COE_ABORTCODE_DATATYPE_LENGTH_TOO_LONG Data type does not match, service parameter too long.
0xC0CF8012	ERR_ECMV4_COE_ABORTCODE_DATATYPE_LENGTH_TOO_SHORT Data type does not match, service parameter too short.
0xC0CF8013	ERR_ECMV4_COE_ABORTCODE_SUBINDEX_DOES_NOT_EXIST Subindex does not exist.
0xC0CF8014	ERR_ECMV4_COE_ABORTCODE_RANGE_OF_PARAMETER_EXCEEDED Value range of parameter exceeded (only for write access).
0xC0CF8015	ERR_ECMV4_COE_ABORTCODE_VALUE_OF_PARAM_WRITTEN_TOO_HIGH Value of parameter written too high.
0xC0CF8016	ERR_ECMV4_COE_ABORTCODE_VALUE_OF_PARAM_WRITTEN_TOO_LOW Value of parameter written too low.
0xC0CF8017	ERR_ECMV4_COE_ABORTCODE_MIN_VALUE_IS_LESS_THAN_MAX_VALUE Maximum value is less than minimum value.
0xC0CF8018	ERR_ECMV4_COE_ABORTCODE_GENERAL_ERROR General error.
0xC0CF8019	ERR_ECMV4_COE_ABORTCODE_NO_TRANSFER_TO_APP Data cannot be transferred to or stored in the application.
0xC0CF801A	ERR_ECMV4_COE_ABORTCODE_LOCAL_CONTROL Data cannot be transferred to or stored in the application because of local control.
0xC0CF801B	ERR_ECMV4_COE_ABORTCODE_NO_TRANSFER_DUE_TO_CURRENT_STATE Data cannot be transferred to or stored in the application because of the present device state.
0xC0CF801C	ERR_ECMV4_COE_ABORTCODE_NO_OBJECT_DICTIONARY_PRESENT Object dictionary dynamic generation fails or no object dictionary is present.
0xC0CF801D	ERR_ECMV4_COE_ABORTCODE_UNKNOWN_ABORT_CODE Unknown SDO abort code.

Hexadecimal Value	Definition Description
0xC0CF801E	ERR_ECMV4_COE_ABORTCODE_GEN_INTERNAL_INCOMPAT General internal incompatibility in the device.

Table 191: EtherCAT Master V4 CoE error codes

6.6 EtherCAT Master V4 EoE error codes

Hexadecimal Value	Definition Description
0xC0D00001	ERR_ECMV4_EOE_INVALID_MAC_ADDRESS EoE: Invalid MAC address.
0xC0D00002	ERR_ECMV4_EOE_INVALID_CALLBACK_TYPE EoE: Invalid callback type.
0xC0D00003	ERR_ECMV4_EOE_DESTINATION_UNREACHABLE EoE: Destination unreachable.
0xC0D00004	ERR_ECMV4_EOE_INVALID_EOE_RESPONSE EoE: Invalid EoE Response.
0xC0D00005	ERR_ECMV4_EOE_UNKNOWN_ERROR EoE: Unknown Error.
0xC0D00006	ERR_ECMV4_EOE_UNSPECIFIED_ERROR EoE: Unspecified error.
0xC0D00007	ERR_ECMV4_EOE_UNSUPPORTED_FRAME_TYPE EoE: Unsupported frame type.
0xC0D00008	ERR_ECMV4_EOE_NO_IP_SUPPORT EoE: No IP support.
0xC0D00009	ERR_ECMV4_EOE_DHCP_NOT_SUPPORTED EoE: DHCP not supported.
0xC0D0000A	ERR_ECMV4_EOE_NO_FILTER_SUPPORT EoE: No filter support.
0xC0D0000B	ERR_ECMV4_EOE_TIMEOUT EoE: Timeout.
0xC0D0000C	ERR_ECMV4_EOE_SHUTTING_DOWN EoE: Shutting Down.
0xC0D0000D	ERR_ECMV4_EOE_MASTER_ADDRESS_NOT_ALLOWED EoE: Master address not allowed.
0xC0D0000E	ERR_ECMV4_EOE_CONFIGURATION_IS_NOT_OPEN EoE: Configuration is not open.
0xC0D0000F	ERR_ECMV4_EOE_CONFIGURATION_IS_ALREADY_OPEN EoE: Configuration is already open.
0xC0D00010	ERR_ECMV4_EOE_DUPLICATE_IP_ADDRESS EoE: Duplicate IP address.
0xC0D00011	ERR_ECMV4_EOE_DUPLICATE_MAC_ADDRESS_ON_MULTIPLE_PORTS EoE: Duplicate MAC address on multiple ports.
0xC0D00012	ERR_ECMV4_EOE_FRAME_TOO_LARGE EoE: Frame too large.

Hexadecimal Value	Definition Description
0xC0D00013	ERR_ECMV4_EOE_IF_INITIALIZATION_ERROR EoE IF: Initialization Error.
0xC0D00014	ERR_ECMV4_EOE_IF_NO_FRAME_AVAILABLE EoE IF: No frame available.
0xC0D00015	ERR_ECMV4_EOE_LINK_DOWN EoE: Link Down.
0xC0D00016	ERR_ECMV4_EOE_REQUEST_DESTINATION_PROBLEM EoE: Request Destination Problem.

Table 192: EtherCAT Master V4 EoE error codes

6.7 EtherCAT Master V4 FoE error codes

Hexadecimal Value	Definition Description
0xC0D10001	ERR_ECMV4_FOE_INITIALIZATION_ERROR FoE: Initialization Error.
0xC0D10002	ERR_ECMV4_FOE_ERROR_UNKNOWN_ERROR FoE: Unknown Error.
0xC0D10003	ERR_ECMV4_FOE_INVALID_TRANSFER_HANDLE FoE: Invalid transfer handle.
0xC0D10004	ERR_ECMV4_FOE_INVALID_TRANSFER_STATE FoE: Invalid transfer state.
0xC0D10005	ERR_ECMV4_FOE_INVALID_SLAVE_STATION_ADDRESS FoE: Invalid slave station address.
0xC0D10006	ERR_ECMV4_FOE_WRONG_SLAVE_STATE FoE: Wrong slave state.
0xC0D10007	ERR_ECMV4_FOE_NO_MAILBOX_AVAILABLE FoE: No mailbox available.
0xC0D10008	ERR_ECMV4_FOE_TRANSFER_ABORTED FoE: Transfer aborted.
0xC0D10009	ERR_ECMV4_FOE_PROTOCOL_TIMEOUT FoE: Protocol Timeout.
0xC0D1000A	ERR_ECMV4_FOE_TRANSFER_SEGMENT_TOO_LONG FoE: Transfer segment too long.
0xC0D1000B	ERR_ECMV4_FOE_MAILBOX_TRANSMIT_FAILED FoE: Mailbox transmit failed.
0xC0D1000C	ERR_ECMV4_FOE_FILENAME_TOO_LONG FoE: Filename too long.
0xC0D1000D	ERR_ECMV4_FOE_BUFFER_EXCEEDED FoE: Buffer exceeded.
0xC0D1000E	ERR_ECMV4_FOE_FIRST_SEGMENT_SHOULD_NOT_BE_EMPTY FoE: First segment should not be empty.
0xC0D1000F	ERR_ECMV4_FOE_SEGMENT_SHOULD_BE_EMPTY FoE: Segment should be empty.

Hexadecimal Value	Definition Description
0xC0D10010	ERR_ECMV4_FOE_REQUEST_DESTINATION_PROBLEM FoE: Request Destination Problem.
0xC0D18000	ERR_ECMV4_FOE_ERROR_NOT_DEFINED FoE: Not Defined.
0xC0D18001	ERR_ECMV4_FOE_ERROR_NOT_FOUND FoE: Not Found.
0xC0D18002	ERR_ECMV4_FOE_ERROR_ACCESS_DENIED FoE: Access Denied.
0xC0D18003	ERR_ECMV4_FOE_ERROR_DISK_FULL FoE: Disk full.
0xC0D18004	ERR_ECMV4_FOE_ERROR_ILLEGAL FoE: Illegal.
0xC0D18005	ERR_ECMV4_FOE_ERROR_PACKET_NUMBER_WRONG FoE: Packet number wrong.
0xC0D18006	ERR_ECMV4_FOE_ERROR_ALREADY_EXISTS FoE: Already exists.
0xC0D18007	ERR_ECMV4_FOE_ERROR_NO_USER FoE: No user.
0xC0D18008	ERR_ECMV4_FOE_ERROR_BOOTSTRAP_ONLY FoE: Bootstrap only.
0xC0D18009	ERR_ECMV4_FOE_ERROR_NOT_BOOTSTRAP FoE: Not Bootstrap.
0xC0D1800A	ERR_ECMV4_FOE_ERROR_NO_RIGHTS FoE: No rights.
0xC0D1800B	ERR_ECMV4_FOE_ERROR_PROGRAM_ERROR FoE: Program Error.

Table 193: EtherCAT Master V4 FoE error codes

6.8 EtherCAT Master V4 SoE error codes

Hexadecimal Value	Definition Description
0xC0D20001	ERR_ECMV4_SOE_UNKNOWN_SOE_ERROR Unknown SoE error.
0xC0D20002	ERR_ECMV4_SOE_INITIALIZATION_ERROR SoE: Initialization Error.
0xC0D20003	ERR_ECMV4_SOE_INVALID_TRANSFER_HANDLE SoE: Invalid transfer handle.
0xC0D20004	ERR_ECMV4_SOE_NO_MAILBOX_AVAILABLE SoE: No mailbox available.
0xC0D20005	ERR_ECMV4_SOE_INVALID_TRANSFER_STATE SoE: Invalid transfer state.
0xC0D20006	ERR_ECMV4_SOE_TRANSFER_SEGMENT_TOO_LONG SoE: Transfer segment too long.

Hexadecimal Value	Definition Description
0xC0D20007	ERR_ECMV4_SOE_SHUTTING_DOWN SoE: Shutting Down.
0xC0D20008	ERR_ECMV4_SOE_MAX_TOTAL_BYTES_SMALLER_THAN_ACTUAL_TOTAL_BYTES SoE: Max Total bytes is smaller than actual total bytes.
0xC0D20009	ERR_ECMV4_SOE_MAILBOX_TRANSMIT_FAILED SoE: Mailbox transmit failed.
0xC0D2000A	ERR_ECMV4_SOE_INVALID_SOE_HEADER SoE: Invalid SoE header.
0xC0D2000B	ERR_ECMV4_SOE_PROTOCOL_TIMEOUT SoE: Protocol Timeout.
0xC0D2000C	ERR_ECMV4_SOE_PROTOCOL_ERROR SoE: Protocol Error.
0xC0D2000D	ERR_ECMV4_SOE_TRANSFER_ABORTED SoE: Transfer aborted.
0xC0D2000E	ERR_ECMV4_SOE_WRONG_SLAVE_STATE SoE: Wrong slave state.
0xC0D2000F	ERR_ECMV4_SOE_REQUEST_DESTINATION_PROBLEM SoE: Request Destination Problem.
0xC0D20010	ERR_ECMV4_SOE_NO_AOE_AVAILABLE SoE: No AoE available.
0xC0D20011	ERR_ECMV4_SOE_INVALID_SLAVE_STATION_ADDRESS SoE: Invalid slave station address.
0xC0D21001	ERR_ECMV4_SOE_SSC_NO_IDN SoE: No IDN.
0xC0D21009	ERR_ECMV4_SOE_SSC_INVALID_ACCESS_TO_ELEMENT_1 SoE: Invalid access to element 1.
0xC0D22001	ERR_ECMV4_SOE_SSC_NO_NAME SoE: No Name.
0xC0D22002	ERR_ECMV4_SOE_SSC_NAME_TRANSMISSION_IS_TOO_SHORT SoE: Name transmission is too short.
0xC0D22003	ERR_ECMV4_SOE_SSC_NAME_TRANSMISSION_IS_TOO_LONG SoE: Name transmission is too long.
0xC0D22004	ERR_ECMV4_SOE_SSC_NAME_CANNOT_BE_CHANGED SoE: Name cannot be changed.
0xC0D22005	ERR_ECMV4_SOE_SSC_NAME_IS_WRITE_PROTECTED_AT_THIS_TIME SoE: Name is write protected at this time.
0xC0D23002	ERR_ECMV4_SOE_SSC_ATTRIBUTE_TRANSMISSION_IS_TOO_SHORT SoE: Attribute transmission is too short.
0xC0D23003	ERR_ECMV4_SOE_SSC_ATTRIBUTE_TRANSMISSION_IS_TOO_LONG SoE: Attribute transmission is too long.
0xC0D23004	ERR_ECMV4_SOE_SSC_ATTRIBUTE_CANNOT_BE_CHANGED SoE: Attribute cannot be changed.
0xC0D23005	ERR_ECMV4_SOE_SSC_ATTRIBUTE_IS_WRITE_PROTECTED_AT_THIS_TIME SoE: Attribute is write protected at this time.
0xC0D24001	ERR_ECMV4_SOE_SSC_NO_UNIT SoE: No Unit.

Hexadecimal Value	Definition Description
0xC0D24002	ERR_ECMV4_SOE_SSC_UNIT_TRANSMISSION_IS_TOO_SHORT SoE: Unit transmission is too short.
0xC0D24003	ERR_ECMV4_SOE_SSC_UNIT_TRANSMISSION_IS_TOO_LONG SoE: Name transmission is too long.
0xC0D24004	ERR_ECMV4_SOE_SSC_UNIT_CANNOT_BE_CHANGED SoE: Unit cannot be changed.
0xC0D24005	ERR_ECMV4_SOE_SSC_UNIT_IS_WRITE_PROTECTED_AT_THIS_TIME SoE: Unit is write protected at this time.
0xC0D25001	ERR_ECMV4_SOE_SSC_NO_MINIMUM_VALUE SoE: No minimum value.
0xC0D25002	ERR_ECMV4_SOE_SSC_MINIMUM_VALUE_TRANSMISSION_IS_TOO_SHORT SoE: Minimum value transmission is too short.
0xC0D25003	ERR_ECMV4_SOE_SSC_MINIMUM_VALUE_TRANSMISSION_IS_TOO_LONG SoE: Minimum value transmission is too long.
0xC0D25004	ERR_ECMV4_SOE_SSC_MINIMUM_VALUE_CANNOT_BE_CHANGED SoE: Minimum value cannot be changed.
0xC0D25005	ERR_ECMV4_SOE_SSC_MINIMUM_VALUE_IS_WRITE_PROTECTED_AT_THIS_TIME SoE: Minimum value is write protected at this time.
0xC0D26001	ERR_ECMV4_SOE_SSC_NO_MAXIMUM_VALUE SoE: No maximum value.
0xC0D26002	ERR_ECMV4_SOE_SSC_MAXIMUM_VALUE_TRANSMISSION_IS_TOO_SHORT SoE: Maximum value transmission is too short.
0xC0D26003	ERR_ECMV4_SOE_SSC_MAXIMUM_VALUE_TRANSMISSION_IS_TOO_LONG SoE: Maximum value transmission is too long.
0xC0D26004	ERR_ECMV4_SOE_SSC_MAXIMUM_VALUE_CANNOT_BE_CHANGED SoE: Maximum value cannot be changed.
0xC0D26005	ERR_ECMV4_SOE_SSC_MAXIMUM_VALUE_IS_WRITE_PROTECTED_AT_THIS_TIME SoE: Maximum value is write protected at this time.
0xC0D27002	ERR_ECMV4_SOE_SSC_OPDATA_TRANSMISSION_IS_TOO_SHORT SoE: Operation data transmission is too short.
0xC0D27003	ERR_ECMV4_SOE_SSC_OPDATA_TRANSMISSION_IS_TOO_LONG SoE: Operation data transmission is too long.
0xC0D27004	ERR_ECMV4_SOE_SSC_OPDATA_CANNOT_BE_CHANGED SoE: Operation data cannot be changed.
0xC0D27005	ERR_ECMV4_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_AT_THIS_TIME SoE: Operation data is write protected at this time.
0xC0D27006	ERR_ECMV4_SOE_SSC_OPDATA_IS_LOWER_THAN_MINIMUM_VALUE SoE: Operation data is lower than minimum value.
0xC0D27007	ERR_ECMV4_SOE_SSC_OPDATA_IS_HIGHER_THAN_MAXIMUM_VALUE SoE: Operation data is higher than maximum value.
0xC0D27008	ERR_ECMV4_SOE_SSC_OPDATA_IS_INVALID SoE: Operation data is invalid.
0xC0D27009	ERR_ECMV4_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_BY_PASSWORD SoE: Operation data is write protected by password.
0xC0D2700A	ERR_ECMV4_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_DUE_CYCLICALLY_CONFIGURED SoE: Operation data is write protected due to being cyclically configured.

Hexadecimal Value	Definition Description
0xC0D2700B	ERR_ECMV4_SOE_SSC_OPDATA_INVALID_INDIRECT_ADDRESSING SoE: Invalid indirect addressing.
0xC0D2700C	ERR_ECMV4_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_DUE_OTHER_SETTINGS SoE: Operation data is write protected due other settings.
0xC0D2700D	ERR_ECMV4_SOE_SSC_OPDATA_INVALID_FLOATING_POINT_NUMBER SoE: Invalid floating point number.
0xC0D2700E	ERR_ECMV4_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_AT_PARAMETERIZATION_LEVEL SoE: Operation data is write protected at parameterization level.
0xC0D2700F	ERR_ECMV4_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_AT_OPERATION_LEVEL SoE: Operation data is write protected at operation level.
0xC0D27010	ERR_ECMV4_SOE_SSC_PROCEDURE_COMMAND_ALREADY_ACTIVE SoE: Procedure command already active.
0xC0D27011	ERR_ECMV4_SOE_SSC_PROCEDURE_COMMAND_NOT_INTERRUPTIBLE SoE: Procedure command is not interruptible.
0xC0D27012	ERR_ECMV4_SOE_SSC_PROCEDURE_COMMAND_NOT_EXECUTABLE_AT_THIS_TIME SoE: Procedure command is not executable at this time.
0xC0D27013	ERR_ECMV4_SOE_SSC_PROCEDURE_COMMAND_NOT_EXECUTABLE_INVALID_PARAM SoE: Procedure command is not executable due to invalid parameter.

Table 194: EtherCAT Master V4 SoE error codes

6.9 EtherCAT Master V4 ENI error codes

Hexadecimal Value	Definition Description
0xC0D40001	ERR_ECMV4_ENI_CLOSING_TAG_DOES_NOT_MATCH_OPENING_TAG ENI: Closing tag does not match opening tag.
0xC0D40002	ERR_ECMV4_ENI_UNEXPECTED_OPENING_TAG_IN_NUMBER_FIELD ENI: Unexpected opening tag in number field.
0xC0D40003	ERR_ECMV4_ENI_UNEXPECTED_SINGLE_TAG_IN_NUMBER_FIELD ENI: Unexpected single tag in number field.
0xC0D40004	ERR_ECMV4_ENI_CLOSING_TAG_DOES_NOT_MATCH_NUMBER_FIELD_TAG ENI: Closing tag does not match number field tag.
0xC0D40005	ERR_ECMV4_ENI_NUMBER_FIELD_IS_INVALID ENI: Number field is invalid.
0xC0D40006	ERR_ECMV4_ENI_UNEXPECTED_OPENING_TAG_IN_STRING_FIELD ENI: Unexpected opening tag in string field.
0xC0D40007	ERR_ECMV4_ENI_UNEXPECTED_SINGLE_TAG_IN_STRING_FIELD ENI: Unexpected single tag in string field.
0xC0D40008	ERR_ECMV4_ENI_CLOSING_TAG_DOES_NOT_MATCH_STRING_FIELD_TAG ENI: Closing tag does not match string field tag.
0xC0D40009	ERR_ECMV4_ENI_DATA_FIELD_IS_NOT_A_HEX_STRING ENI: Data field is not a hex string.

Hexadecimal Value	Definition Description
0xC0D4000A	ERR_ECMV4_ENI_UNEXPECTED_OPENING_TAG_IN_DATA_FIELD ENI: Unexpected opening tag in data field.
0xC0D4000B	ERR_ECMV4_ENI_UNEXPECTED_SINGLE_TAG_IN_DATA_FIELD ENI: Unexpected single tag in data field.
0xC0D4000C	ERR_ECMV4_ENI_CLOSING_TAG_DOES_NOT_MATCH_DATA_FIELD_TAG ENI: Closing tag does not match data field tag.
0xC0D4000D	ERR_ECMV4_ENI_INTERNAL_ERROR ENI: Internal Error.
0xC0D4000E	ERR_ECMV4_ENI_PREMATURE_END_OF_FILE ENI: Premature End of File.
0xC0D4000F	ERR_ECMV4_ENI_END_OF_FILE ENI: End of file.
0xC0D40010	ERR_ECMV4_ENI_INVALID_XML ENI: Invalid XML.
0xC0D40011	ERR_ECMV4_ENI_UNEXPECTED_SINGLE_TAG_IN_ECAT_INITCMD_BLOCK ENI: Unexpected single tag in InitCmd block.
0xC0D40012	ERR_ECMV4_ENI_DUPLICATE_TAG_IN_ECAT_INITCMD_BLOCK ENI: Duplicate tag in InitCmd block.
0xC0D40013	ERR_ECMV4_ENI_UNEXPECTED_OPENING_TAG_IN_ECAT_INITCMD_BLOCK ENI: Unexpected opening tag in InitCmd block.
0xC0D40014	ERR_ECMV4_ENI_CLOSING_TAG_DOES_NOT_MATCH_ECAT_INITCMD_TAG ENI: Closing tag does not match InitCmd opening tag.
0xC0D40015	ERR_ECMV4_ENI_INVALID_TRANSITION_IN_ECAT_INITCMD_BLOCK ENI: Invalid transition in InitCmd block.
0xC0D40016	ERR_ECMV4_ENI_ECAT_INITCMD_IS_INCOMPLETE ENI: InitCmd block is incomplete.
0xC0D40017	ERR_ECMV4_ENI_ECAT_INITCMD_VALIDATE_BLOCK_IS_INVALID ENI: InitCmd validate block is invalid.
0xC0D40018	ERR_ECMV4_ENI_UNEXPECTED_SINGLE_TAG_IN_ECAT_INITCMD_VALIDATE_BLOCK ENI: Unexpected single tag in InitCmd validate block.
0xC0D40019	ERR_ECMV4_ENI_DUPLICATE_TAG_IN_ECAT_INITCMD_VALIDATE_BLOCK ENI: Duplicate tag in InitCmd validate block.
0xC0D4001A	ERR_ECMV4_ENI_UNEXPECTED_OPENING_TAG_IN_ECAT_INITCMD_VALIDATE_BLOCK ENI: Unexpected opening tag in InitCmd validate block.
0xC0D4001B	ERR_ECMV4_ENI_CLOSING_TAG_DOES_NOT_MATCH_ECAT_INITCMD_VALIDATE_TAG ENI: Closing tag does not match InitCmd validate opening tag.
0xC0D4001C	ERR_ECMV4_ENI_XML_FILE_IS_NOT_AN_ENI_XML ENI: XML file is not an ENI file.
0xC0D4001D	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_ETHERCATCONFIG_BLOCK ENI: Unexpected closing tag in EtherCATConfig block.
0xC0D4001E	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_CONFIG_BLOCK ENI: Unexpected closing tag in Config block.
0xC0D4001F	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_MASTER_BLOCK ENI: Unexpected closing tag in Master block.
0xC0D40020	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_MASTER_INITCMDS_BLOCK ENI: Unexpected closing tag in Master/InitCmds block.

Hexadecimal Value	Definition Description
0xC0D40021	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_BLOCK ENI: Unexpected closing tag in Slave block.
0xC0D40022	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_INITCMDS_BLOCK ENI: Unexpected closing tag in Slave/InitCmds block.
0xC0D40023	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_INFO_BLOCK ENI: Unexpected closing tag in Slave/Info block.
0xC0D40024	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_BLOCK ENI: Unexpected closing tag in Slave/Mailbox block.
0xC0D40025	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_DC_BLOCK ENI: Unexpected closing tag in Slave/DC block.
0xC0D40026	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_HOTCONNECT_BLOCK ENI: Unexpected closing tag in Slave/HotConnect block.
0xC0D40027	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_PREVIOUSPORT_BLOCK ENI: Unexpected closing tag in Slave/PreviousPort block.
0xC0D40028	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_CYCLIC_BLOCK ENI: Unexpected closing tag in Cyclic block.
0xC0D40029	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_CYCLIC_FRAME_BLOCK ENI: Unexpected closing tag in Cyclic/Frame block.
0xC0D4002A	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_CYCLIC_FRAME_CMD_BLOCK ENI: Unexpected closing tag in Cyclic/Frame/Cmd block.
0xC0D4002B	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_COE_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/CoE block.
0xC0D4002C	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_COE_INITCMDS_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/CoE/InitCmds block.
0xC0D4002D	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_COE_INITCMD_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/CoE/InitCmds/InitCmd block.
0xC0D4002E	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_SOE_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/SoE block.
0xC0D4002F	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_SOE_INITCMDS_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/SoE/InitCmds block.
0xC0D40030	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_SOE_INITCMD_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/SoE/InitCmds/InitCmd block.
0xC0D40031	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_EOE_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/EoE block.
0xC0D40032	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_EOE_INITCMDS_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/EoE/InitCmds block.
0xC0D40033	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_EOE_INITCMD_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/EoE/InitCmds/InitCmd block.
0xC0D40034	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_AOE_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/AoE block.
0xC0D40035	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_AOE_INITCMDS_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/AoE/InitCmds block.

Hexadecimal Value	Definition Description
0xC0D40036	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_AOE_INITCMD_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/AoE/InitCmds/InitCmd block.
0xC0D40037	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_FOE_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/FoE block.
0xC0D40038	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_FOE_INITCMDS_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/FoE/InitCmds block.
0xC0D40039	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_FOE_INITCMD_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/FoE/InitCmds/InitCmd block.
0xC0D4003A	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_VOE_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/VoE block.
0xC0D4003B	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_VOE_INITCMDS_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/VoE/InitCmds block.
0xC0D4003C	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_VOE_INITCMD_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/VoE/InitCmds/InitCmd block.
0xC0D4003D	ERR_ECMV4_ENI_LOADING_NOT_ENABLED ENI: Loading not enabled.
0xC0D4003E	ERR_ECMV4_ENI_COULD_NOT_OPEN_FILE ENI: Could not open file.
0xC0D4003F	ERR_ECMV4_ENI_BASE_CYCLE_TIME_TOO_SMALL ENI: Base cycle time too small.
0xC0D40040	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_INFO_PREVIOUSPORT_BLOCK ENI: Unexpected closing tag in Slave/Info/PreviousPort block.
0xC0D40041	ERR_ECMV4_ENI_INVALID_PORT_IN_SLAVE_INFO_PREVIOUSPORT_BLOCK ENI: Invalid port in Slave/Info/PreviousPort block.
0xC0D40042	ERR_ECMV4_ENI_INVALID_PHYSADDR_IN_SLAVE_INFO_PREVIOUSPORT_BLOCK ENI: Invalid PhysAddr in Slave/Info/PreviousPort block.
0xC0D40043	ERR_ECMV4_ENI_INVALID_TRANSITION_IN_COE_INITCMD_BLOCK ENI: Invalid transition in CoE/InitCmd block.
0xC0D40044	ERR_ECMV4_ENI_MISSING_TRANSITIONS_IN_COE_INITCMD_BLOCK ENI: Missing transitions in CoE/InitCmd block.
0xC0D40045	ERR_ECMV4_ENI_INVALID_CCS_IN_COE_INITCMD_BLOCK ENI: Invalid Ccs in CoE/InitCmd block.
0xC0D40046	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_SEND_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/Send block.
0xC0D40047	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_RECV_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/Recv block.
0xC0D40048	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_MAILBOX_BOOTSTRAP_BLOCK ENI: Unexpected closing tag in Slave/Mailbox/Bootstrap block.
0xC0D40049	ERR_ECMV4_ENI_INVALID_EOE_INITCMD ENI: Invalid EoE InitCmd.

Hexadecimal Value	Definition Description
0xC0D4004A	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_MASTER_MAILBOXSTATES_BLOCK ENI: Unexpected closing tag in Master/MailboxStates block.
0xC0D4004B	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_INFO_PROCESSDATA_BLOCK ENI: Unexpected closing tag in Slave/Info/ProcessData block.
0xC0D4004C	ERR_ECMV4_ENI_INVALID_MBOX_STATE_BIT_NO_IN_SLAVE_INFO_PROCESSDATA_RECV_BLOCK ENI: Invalid MBoxState Bit number in Slave/Info/ProcessData/Recv block.
0xC0D4004D	ERR_ECMV4_ENI_PROCESS_DATA_CONFIG_OFFSET_NOT_POSSIBLE ENI: Process Data config offset not possible.
0xC0D4004E	ERR_ECMV4_ENI_MISSING_TRANSITIONS_IN_SOE_INITCMD_BLOCK ENI: Missing transitions in SoE/InitCmd block.
0xC0D4004F	ERR_ECMV4_ENI_INVALID_OPCODE_IN_SOE_INITCMD_BLOCK ENI: Invalid OpCode in SoE/InitCmd block.
0xC0D40050	ERR_ECMV4_ENI_UNSUPPORTED_ECAT_CMD_IN_IDENTIFYCMD_BLOCK ENI: Unsupported EtherCAT command in IdentifyCmd block.
0xC0D40051	ERR_ECMV4_ENI_UNSUPPORTED_ECAT_REG_IN_IDENTIFYCMD_BLOCK ENI: Unsupported EtherCAT register in IdentifyCmd block.
0xC0D40052	ERR_ECMV4_ENI_MISSING_IDENTIFYCMD_DATA_FOR_HOT_CONNECT_SLAVE ENI: Missing IdentifyCmd data for HotConnect slave.
0xC0D40053	ERR_ECMV4_ENI_INVALID_IDENTIFYCMD_DATA_FOR_HOT_CONNECT_SLAVE ENI: Invalid IdentifyCmd data for HotConnect slave.
0xC0D40054	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_INFO_SM_BLOCK ENI: Unexpected closing tag in Slave/Info/Sm block.
0xC0D40055	ERR_ECMV4_ENI_TOO_MANY_PDOS_LISTED_IN_SLAVE_INFO_SM_BLOCK ENI: Too many PDOs listed in Slave/Info/Sm block.
0xC0D40056	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_INFO_PDO_BLOCK ENI: Unexpected closing tag in Slave/Info/Pdo block.
0xC0D40057	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_SLAVE_INFO_PDO_ENTRY_BLOCK ENI: Unexpected closing tag in Slave/Info/Pdo/Entry block.
0xC0D40058	ERR_ECMV4_ENI_TOO_MANY_PDO_ENTRIES_LISTED_IN_SLAVE_INFO_PDO_BLOCK ENI: Too many PDO entries listed in Slave/Info/PDO block.
0xC0D40059	ERR_ECMV4_ENI_INDEX_OF_PDO_ENTRY_INVALID ENI: Index of PDO entry invalid.
0xC0D4005A	ERR_ECMV4_ENI_SUBINDEX_OF_PDO_ENTRY_INVALID ENI: Subindex of PDO entry invalid.
0xC0D4005B	ERR_ECMV4_ENI_BIT_LENGTH_OF_PDO_ENTRY_EXCEEDS_MAXIMUM_BIT_LENGTH ENI: Bit length of PDO entry exceeds maximum bit length.
0xC0D4005C	ERR_ECMV4_ENI_NO_SLAVES_IN_ENI ENI: No slaves in ENI.
0xC0D4005D	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_CYCLIC_FRAME_CMD_COPYINFOS_BLOCK ENI: Unexpected closing tag in Cyclic/Frame/Cmd/CopyInfos block.
0xC0D4005E	ERR_ECMV4_ENI_UNEXPECTED_CLOSING_TAG_IN_CYCLIC_FRAME_CMD_COPYINFOS_COPYINFO_BLOCK ENI: Unexpected closing tag in Cyclic/Frame/Cmd/CopyInfos/CopyInfo block.

Hexadecimal Value	Definition Description
0xC0D4005F	ERR_ECMV4_ENI_DUPLICATE_TAG_IN_CYCLIC_FRAME_CMD_COPYINFOS_COPYINFO_BLOCK ENI: Duplicate tag in Cyclic/Frame/Cmd/CopyInfos/CopyInfo block.
0xC0D40060	ERR_ECMV4_ENI_INVALID_COPYINFO_BLOCK ENI: Invalid CopyInfo block.
0xC0D40061	ERR_ECMV4_ENI_INVALID_ATTRIBUTE_IN_ECAT_INITCMD_VALIDATE_BLOCK ENI: Invalid XML attribute in InitCmd/Validate block.
0xC0D40062	ERR_ECMV4_ENI_INVALID_TRANSITION_IN_VOE_INITCMD_BLOCK ENI: Invalid transition in VoE/InitCmd block.
0xC0D40063	ERR_ECMV4_ENI_MISSING_TRANSITIONS_IN_VOE_INITCMD_BLOCK ENI: Missing transitions in VoE/InitCmd block.
0xC0D40064	ERR_ECMV4_ENI_MISSING_DATA_IN_VOE_INITCMD_BLOCK ENI: Missing data in VoE/InitCmd block.
0xC0D40065	ERR_ECMV4_ENI_INVALID_NETID_IN_AOE_BLOCK ENI: Invalid Net ID in AoE block.
0xC0D40066	ERR_ECMV4_ENI_INVALID_TRANSITION_IN_AOE_INITCMD_BLOCK ENI: Invalid transition in AoE/InitCmd block.
0xC0D40067	ERR_ECMV4_ENI_MISSING_TRANSITIONS_IN_AOE_INITCMD_BLOCK ENI: Missing transition in AoE/InitCmd block.
0xC0D40068	ERR_ECMV4_ENI_MISSING_DATA_IN_AOE_INITCMD_BLOCK ENI: Missing data in AoE/InitCmd block.
0xC0D40069	ERR_ECMV4_ENI_INVALID_BEFORE_SLAVE_VALUE ENI: Invalid BeforeSlave value.
0xC0D4006A	ERR_ECMV4_ENI_INVALID_COMPLETE_ACCESS_ATTRIBUTE ENI: Invalid CompleteAccess XML attribute.
0xC0D4006B	ERR_ECMV4_ENI_ECAT_INITCMD_MISSING_TRANSITIONS ENI: Ecat InitCmd misses transitions.

Table 195: EtherCAT Master V4 ENI error codes

6.10 EtherCAT Master V4 AL Status codes

The range from 0xC0D50000 to 0xC0D5FFFF is used for direct mapping of ALStatusCodes. Therefore, the EtherCAT Master stack can report error codes from ESM here that are not yet listed in the following list. ALStatusCode mapping example: ALStatusCode 0x001D (Invalid Output Configuration) becomes 0xC0D5001D.

Hexadecimal Value	Definition Description
0xC0D50001	ERR_ECMV4_ALSTATUSCODE_UNSPECIFIED_ERROR AlStatusCode: Unspecified Error.
0xC0D50002	ERR_ECMV4_ALSTATUSCODE_NO_MEMORY AlStatusCode: No Memory.
0xC0D50003	ERR_ECMV4_ALSTATUSCODE_INVALID_DEVICE_SETUP AlStatusCode: Invalid device setup.
0xC0D50011	ERR_ECMV4_ALSTATUSCODE_INVALID_REQUESTED_STATE_CHANGE AlStatusCode: Invalid requested state change.
0xC0D50012	ERR_ECMV4_ALSTATUSCODE_UNKNOWN_REQUESTED_STATE AlStatusCode: Unknown requested state.
0xC0D50013	ERR_ECMV4_ALSTATUSCODE_BOOTSTRAP_NOT_SUPPORTED AlStatusCode: Bootstrap not supported.
0xC0D50014	ERR_ECMV4_ALSTATUSCODE_NO_VALID_FIRMWARE AlStatusCode: No valid firmware.
0xC0D50015	ERR_ECMV4_ALSTATUSCODE_INVALID_BOOT_MAILBOX_CONFIGURATION AlStatusCode: Invalid BOOT mailbox configuration.
0xC0D50016	ERR_ECMV4_ALSTATUSCODE_INVALID_PREOP_MAILBOX_CONFIGURATION AlStatusCode: Invalid PREOP mailbox configuration.
0xC0D50017	ERR_ECMV4_ALSTATUSCODE_INVALID_SYNC_MANAGER_CONFIGURATION AlStatusCode: Invalid Sync Manager configuration.
0xC0D50018	ERR_ECMV4_ALSTATUSCODE_NO_VALID_INPUTS_AVAILABLE AlStatusCode: No valid inputs available.
0xC0D50019	ERR_ECMV4_ALSTATUSCODE_NO_VALID_OUTPUTS AlStatusCode: No valid outputs.
0xC0D5001A	ERR_ECMV4_ALSTATUSCODE_SYNCHRONIZATION_ERROR AlStatusCode: Synchronization Error.
0xC0D5001B	ERR_ECMV4_ALSTATUSCODE_SYNC_MANAGER_WATCHDOG AlStatusCode: Sync Manager watchdog.
0xC0D5001C	ERR_ECMV4_ALSTATUSCODE_INVALID_SYNC_MANAGER_TYPES AlStatusCode: Invalid Sync Manager types.
0xC0D5001D	ERR_ECMV4_ALSTATUSCODE_INVALID_OUTPUT_CONFIGURATION AlStatusCode: Invalid output configuration.
0xC0D5001E	ERR_ECMV4_ALSTATUSCODE_INVALID_INPUT_CONFIGURATION AlStatusCode: Invalid input configuration.
0xC0D5001F	ERR_ECMV4_ALSTATUSCODE_INVALID_WATCHDOG_CONFIGURATION AlStatusCode: Invalid watchdog configuration.
0xC0D50020	ERR_ECMV4_ALSTATUSCODE_SLAVE_NEEDS_COLD_START AlStatusCode: Slave needs cold start.
0xC0D50021	ERR_ECMV4_ALSTATUSCODE_SLAVE_NEEDS_INIT AlStatusCode: Slave needs INIT.

Hexadecimal Value	Definition Description
0xC0D50022	ERR_ECMV4_ALSTATUSCODE_SLAVE_NEEDS_PREOP AlStatusCode: Slave needs PREOP.
0xC0D50023	ERR_ECMV4_ALSTATUSCODE_SLAVE_NEEDS_SAFEOP AlStatusCode: Slave needs SAFEOP.
0xC0D50024	ERR_ECMV4_ALSTATUSCODE_INVALID_INPUT_MAPPING AlStatusCode: Invalid input mapping.
0xC0D50025	ERR_ECMV4_ALSTATUSCODE_INVALID_OUTPUT_MAPPING AlStatusCode: Invalid output mapping.
0xC0D50026	ERR_ECMV4_ALSTATUSCODE_INCONSISTENT_SETTINGS AlStatusCode: Inconsistent settings.
0xC0D50027	ERR_ECMV4_ALSTATUSCODE_FREERUN_NOT_SUPPORTED AlStatusCode: Free Run not supported.
0xC0D50028	ERR_ECMV4_ALSTATUSCODE_SYNCMODE_NOT_SUPPORTED AlStatusCode: SyncMode not supported.
0xC0D50029	ERR_ECMV4_ALSTATUSCODE_FREERUN_NEEDS_3BUFFER_MODE AlStatusCode: Free Run needs 3 Buffer Mode.
0xC0D5002A	ERR_ECMV4_ALSTATUSCODE_BACKGROUND_WATCHDOG AlStatusCode: Background Watchdog.
0xC0D5002B	ERR_ECMV4_ALSTATUSCODE_NO_VALID_INPUTS_AND_OUTPUTS AlStatusCode: No valid inputs and outputs.
0xC0D5002C	ERR_ECMV4_ALSTATUSCODE_FATAL_SYNC_ERROR AlStatusCode: Fatal Sync Error.
0xC0D5002D	ERR_ECMV4_ALSTATUSCODE_NO_SYNC_ERROR AlStatusCode: No Sync Error.
0xC0D50030	ERR_ECMV4_ALSTATUSCODE_INVALID_DC_SYNC_CONFIGURATION AlStatusCode: Invalid DC Sync configuration.
0xC0D50031	ERR_ECMV4_ALSTATUSCODE_INVALID_DC_LATCH_CONFIGURATION AlStatusCode: Invalid DC Latch configuration.
0xC0D50032	ERR_ECMV4_ALSTATUSCODE_PLL_ERROR AlStatusCode: PLL Error.
0xC0D50033	ERR_ECMV4_ALSTATUSCODE_DC_SYNC_IO_ERROR AlStatusCode: DC Sync I/O Error.
0xC0D50034	ERR_ECMV4_ALSTATUSCODE_DC_SYNC_TIMEOUT_ERROR AlStatusCode: DC Sync Timeout Error.
0xC0D50035	ERR_ECMV4_ALSTATUSCODE_DC_INVALID_SYNC_CYCLE_TIME AlStatusCode: Invalid DC Sync cycle time.
0xC0D50036	ERR_ECMV4_ALSTATUSCODE_DC_SYNC0_CYCLE_TIME AlStatusCode: DC Sync0 Cycle Time.
0xC0D50037	ERR_ECMV4_ALSTATUSCODE_DC_SYNC1_CYCLE_TIME AlStatusCode: DC Sync0 Cycle Time.
0xC0D50041	ERR_ECMV4_ALSTATUSCODE_MBX_AOE AlStatusCode: Mbx AoE.
0xC0D50042	ERR_ECMV4_ALSTATUSCODE_MBX_EOE AlStatusCode: Mbx EoE.
0xC0D50043	ERR_ECMV4_ALSTATUSCODE_MBX_COE AlStatusCode: Mbx CoE.

Hexadecimal Value	Definition Description
0xC0D50044	ERR_ECMV4_ALSTATUSCODE_MBX_FOE AlStatusCode: Mbx FoE.
0xC0D50045	ERR_ECMV4_ALSTATUSCODE_MBX_SOE AlStatusCode: Mbx SoE.
0xC0D5004F	ERR_ECMV4_ALSTATUSCODE_MBX_VOE AlStatusCode: Mbx VoE.
0xC0D50050	ERR_ECMV4_ALSTATUSCODE_EEPROM_NO_ACCESS AlStatusCode: EEPROM: No PDI Access.
0xC0D50051	ERR_ECMV4_ALSTATUSCODE_EEPROM_ERROR AlStatusCode: EEPROM: Error.
0xC0D50060	ERR_ECMV4_ALSTATUSCODE_SLAVE_RESTARTED_LOCALLY AlStatusCode: Slave restarted locally.
0xC0D50061	ERR_ECMV4_ALSTATUSCODE_DEVICE_IDENTIFICATION_VALUE_UPDATED AlStatusCode: Device identification value updated.
0xC0D500F0	ERR_ECMV4_ALSTATUSCODE_APPLICATION_CONTROLLER_AVAILABLE AlStatusCode: Application controller available.

Table 196: EtherCAT Master V4 AL Status codes

6.11 EtherCAT Master V4 IF error codes

Hexadecimal Value	Definition Description
0xC0D60001	ERR_ECMV4_IF_COE_SUPPORT_NOT_AVAILABLE CoE Support not available.
0xC0D60002	ERR_ECMV4_IF_SOE_SUPPORT_NOT_AVAILABLE SoE Support not available.
0xC0D60003	ERR_ECMV4_IF_FOE_SUPPORT_NOT_AVAILABLE FoE Support not available.
0xC0D60004	ERR_ECMV4_IF_AOE_SUPPORT_NOT_AVAILABLE AoE Support not available.
0xC0D60005	ERR_ECMV4_IF_INVALID_TRANSPORT_TYPE Invalid transport type.
0xC0D60006	ERR_ECMV4_IF_SOE_INVALID_DRIVE_NO SoE: Invalid drive number.
0xC0D60007	ERR_ECMV4_IF_SOE_INVALID_ELEMENT_FLAGS Invalid element flags.
0xC0D60008	ERR_ECMV4_IF_INVALID_SOE_TRANSFER_ID Invalid SoE Transfer Id.
0xC0D60009	ERR_ECMV4_IF_TRANSFER_ABORTED Transfer aborted.
0xC0D6000A	ERR_ECMV4_IF_OUT_OF_PACKETS Out of packets.
0xC0D6000B	ERR_ECMV4_IF_OUT_OF_TRANSFER_CONTEXTS Out of transfer contexts.

Hexadecimal Value	Definition Description
0xC0D6000C	ERR_ECMV4_IF_INVALID_SUBINDEX_FOR_COMPLETE_ACCESS Invalid Subindex for Complete Access.
0xC0D6000D	ERR_ECMV4_IF_INVALID_COE_TRANSFER_ID Invalid CoE Transfer Id.
0xC0D6000E	ERR_ECMV4_IF_INVALID_COE_SDOINFO_LISTTYPE Invalid CoE SDOINFO ListType.
0xC0D6000F	ERR_ECMV4_IF_FILE_READ_ERROR File Read Error.
0xC0D60010	ERR_ECMV4_IF_COULD_NOT_OPEN_FILE Could not open file.
0xC0D60011	ERR_ECMV4_IF_INVALID_CONFIG_NXD Invalid config.nxd.
0xC0D60012	ERR_ECMV4_IF_CONFIG_NXD_WITHOUT_SLAVES Config.nxd without slaves.
0xC0D60013	ERR_ECMV4_IF_INVALID_FILE_NAME Invalid filename.
0xC0D60014	ERR_ECMV4_IF_INVALID_FOE_TRANSFER_ID Invalid FoE Transfer Id.
0xC0D60015	ERR_ECMV4_IF_INVALID_GET_TOPOLOGY_TRANSFER_ID Invalid Get Topology Transfer Id.
0xC0D60016	ERR_ECMV4_IF_INVALID_AOE_TRANSFER_ID Invalid AoE Transfer Id.
0xC0D60017	ERR_ECMV4_IF_CONFIG_ACFG_SUPPORT_NOT_AVAILABLE AutoCfg support not available.

Table 197: EtherCAT Master V4 IF error codes

6.12 EtherCAT Master V4 AP Task error codes

Hexadecimal Value	Definition Description
0xC0D70001	ERR_ECMV4_AP_FIRMWARE_HAS_CRASHED Firmware has crashed.
0xC0D70002	ERR_ECMV4_AP_CONFIGURATION_INTERFACE_NOT_AVAILABLE Configuration interface not currently available.
0xC0D70003	ERR_ECMV4_AP_SET_TARGET_STATE_NOT_ALLOWED_DURING_CFG_LOADING Set Target State is not allowed during configuration loading.
0xC0D70004	ERR_ECMV4_AP_INVALID_STARTUP_PARAMETER Invalid startup parameter.

Table 198: EtherCAT Master V4 AP Task error codes

7 Appendix

7.1 Accessing the protocol stack by programming the AP task's queue

In general, programming the AP task or the stack has to be performed according to the rules explained in the Hilscher Task Layer Reference Manual. There you can also find more information about the variables discussed in the following.

7.1.1 Getting the receiver task handle of the process queue

To get the handle of the process queue of the ECM_IF-Task the Macro `TLR_QUE_IDENTIFY()` needs to be used. This macro delivers a pointer to the handle of the intended queue to be accessed (which is returned within the third parameter, `phQue`), if you provide it with the name of the queue (and an instance of your own task). The correct ASCII-queue names for accessing the CP-Task, which you have to use as current value for the first parameter (`pszIdn`), is

ASCII Queue name	Description
`"QUE_ECM_IF"`	Name of the ECM_IF-Task process queue

Table 199: Names of Queues in the EtherCAT Master Firmware

The returned handle has to be used as value `ulDest` in all initiator packets the AP-Task intends to send to the `ECM_IF`-Task. This handle is the same handle that has to be used in conjunction with the macros like `TLR_QUE_SENDPACKET_FIFO/LIFO()` for sending a packet to the respective task.

Note: The ECM_IF-Task provides a common access point to all master tasks when the AP-Task is not used (since V4.X).

7.2 Extended status

The content of the channel specific extended status block is specific to the implementation. Depending on the protocol, a status area may or may not be present in the dual-port memory. It is always available in the default memory map (see section 3.2.1 of *netX Dual-Port Memory Manual*).

Offset	Type	Name	Description
0x0050	UINT8	`abExtendedStatus[432]`	Extended Status Area Protocol Stack Specific Status Area

Table 200: Extended Status Block

Extended Status Block Structure

```
typedef __ECM_AP_PACKED_PRE struct __ECM_AP_PACKED_POST ECM_AP_EXTENDED_STATUS_DATA_Ttag
{
  uint32_t ulDcEnabled; /* always set on ECMV3.X */
  uint32_t aulReserved[12];

  uint32_t ulMarker3;

  uint32_t ulValidBufferedDpmInputDataExchangesCount;
  uint32_t ulBlockedBufferedDpmInputDataExchangesCount;

  uint32_t ulValidBufferedDpmOutputDataExchangesCount;
  uint32_t ulBlockedBufferedDpmOutputDataExchangesCount;

  uint8_t abState[ECM_AP_STATE_INFO_STRING_LENGTH];
  uint8_t bCurrentState;
  uint32_t aulLastFiveCommunicationErrors[5];

  uint32_t ulCompleteCyclesCount;
  uint32_t ulCyclesWithLostFramesCount;

  uint32_t ulMarker0;
  uint32_t ulValidSynchInputDataExchangesCount;
  uint32_t ulCompletedSynchInputDataExchangesCount;
  uint32_t ulBlockedSynchInputDataExchangesCount;

  uint32_t ulValidSynchOutputDataExchangesCount;
  uint32_t ulCompletedSynchOutputDataExchangesCount;
  uint32_t ulBlockedSynchOutputDataExchangesCount;

  uint32_t ulMarker1;
  uint32_t ulBufferedBusInputDataExchangesCount;
  uint32_t ulBufferedBusOutputDataExchangesCount;
  uint32_t ulCompletedBusInputDataExchangesCount;

  uint32_t ulMarker2;

  /* only 2 dwords left here */
} ECM_AP_EXTENDED_STATUS_DATA_T;
```

7.3 Legal notes

Copyright

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fission processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

7.4 List of figures

Figure 1: I/O sync mode 1 Timing Diagram ... 21
Figure 2: I/O sync mode 2 timing diagram ... 22
Figure 3: Flow diagram diagnostic log indications handling ... 66
Figure 4: Single fragment handling of download/write SDO Service ... 70
Figure 5: Two fragment handling of download/write SDO Service ... 71
Figure 6: Multiple fragment handling of download/write SDO Service ... 71
Figure 7: Single fragment handling of upload/read SDO Service ... 72
Figure 8: Two fragment handling of upload/read SDO Service ... 72
Figure 9: Multiple fragment handling of upload/read SDO Service ... 73
Figure 10: Flowchart for download / write SDO fragmentation ... 78
Figure 11: Flowchart for upload/ read SDO Fragmentation ... 79
Figure 12: Single fragment handling of GetOdList SDOINFO service ... 80
Figure 13: Two fragment handling of GetOdList SDOINFO service ... 80
Figure 14: Multiple fragment handling of GetOdList SDOINFO service ... 81
Figure 15: Single fragment handling of GetObjDesc SDOINFO Service ... 82
Figure 16: Two fragment handling of GetObjDesc SDOINFO Service ... 82
Figure 17: Multiple fragment handling of GetObjDesc SDOINFO Service ... 83
Figure 18: Single fragment handling of GetEntryDesc SDOINFO Service ... 84
Figure 19: Two fragment handling of GetEntryDesc SDOINFO Service ... 84
Figure 20: Multiple fragment handling of GetEntryDesc SDOINFO Service ... 85
Figure 21: Flowchart for GetOdList fragmentation ... 94
Figure 22: Flowchart for GetObjDesc fragmentation ... 95
Figure 23: Flowchart for GetEntryDesc fragmentation ... 96
Figure 24: Single fragment handling of write file service ... 108
Figure 25: Two Fragment handling of write file service ... 109
Figure 26: Multiple fragment handling of write file service ... 109
Figure 27: Single fragment handling of read file service ... 110
Figure 28: Two fragment handling of read file service ... 110
Figure 29: Multiple fragment handling of read file service ... 111
Figure 30: Flowchart for write file service fragmentation ... 118
Figure 31: Flowchart for read file fragmentation ... 119
Figure 32: Single fragment handling of write IDN service ... 120
Figure 33: Two Fragment handling of write IDN service ... 120
Figure 34: Multiple fragment handling of write IDN service ... 121
Figure 35: Single fragment handling of read IDN service ... 122
Figure 36: Two fragment handling of read IDN service ... 122
Figure 37: Multiple fragment handling of read IDN service ... 123
Figure 38: Flowchart for write IDN service fragmentation ... 130
Figure 39: Flowchart for read IDN fragmentation ... 131
Figure 40: Flow diagram of slave diagnostic information packets ... 157
Figure 41: Example packet flow for using generic bus scan ... 198
Figure 42: Using legacy bus scan ... 202

7.5 List of tables

Table 1: List of Revisions 5
Table 2: Terms, Abbreviations and Definitions 9
Table 3: References 9
Table 4: Bus and Master Parameters, their Meanings and their Ranges of allowed Values 11
Table 5: LED states for the EtherCAT Master 15
Table 6: LED state definitions for the EtherCAT Master protocol 16
Table 7: Auto-increment address related to topology position 18
Table 8: Topology position scheme related to topology position on bus 19
Table 9: HIL_SET_HANDSHAKE_CONFIG_REQ – Set handshake configuration request 23
Table 10: HIL_SET_HANDSHAKE_CONFIG_CNF – Set handshake configuration confirmation 24
Table 11: ECM_IF_CMD_SET_MASTER_TARGET_STATE_REQ – Set master target state request 27
Table 12: Possible values of bTargetState 28
Table 13: ECM_IF_CMD_SET_MASTER_TARGET_STATE_CNF – Set master target state confirmation 28
Table 14: ECM_IF_CMD_GET_MASTER_CURRENT_STATE_REQ – Get master current state request 29
Table 15: ECM_IF_CMD_GET_MASTER_CURRENT_STATE_CNF – Get master current state confirmation 29
Table 16: Possible values of bCurrentState 30
Table 17: Possible values of bTargetState 30
Table 18: Meaning of ulMasterFlags 31
Table 19: ETHERCAT_MASTER_CMD_SET_ECSTATE_REQ – Set master target state request (Legacy) 32
Table 20: Possible values of usNewEcState 32
Table 21: ETHERCAT_MASTER_CMD_SET_ECSTATE_CNF – Set master target state confirmation (Legacy) 33
Table 22: ETHERCAT_MASTER_CMD_GET_ECSTATE_REQ – Get master current state request (Legacy) 34
Table 23: ETHERCAT_MASTER_CMD_GET_ECSTATE_CNF – Get master current state confirmation (Legacy) 34
Table 24: Possible values of usCurrentEcState 35
Table 25: ECM_IF_CMD_SET_SLAVE_TARGET_STATE_REQ – Set slave target state request 36
Table 26: Possible values of bTargetState 36
Table 27: ECM_IF_CMD_SET_SLAVE_TARGET_STATE_CNF – Set slave target state confirmation 37
Table 28: ECM_IF_CMD_GET_SLAVE_CURRENT_STATE_REQ – Get slave current state request 38
Table 29: ECM_IF_CMD_GET_SLAVE_CURRENT_STATE_CNF – Get slave current state confirmation 39
Table 30: Possible values of bCurrentState 40
Table 31: Possible values of bTargetState 40
Table 32: HIL_REGISTER_APP_REQ – Register for status indications request 41
Table 33: HIL_REGISTER_APP_CNF – Register for status indications confirmation 41
Table 34: HIL_UNREGISTER_APP_REQ –Unregister from status indications request 42
Table 35: HIL_UNREGISTER_APP_CNF – Unregister from status indications confirmation 42
Table 36: ECM_IF_MASTER_CURRENT_STATE_IND_T – Master state indication 43
Table 37: Possible values of bCurrentState 44
Table 38: Possible values of bTargetState 44
Table 39: ECM_IF_SLAVE_CURRENT_STATE_IND – Current slave state indication 45
Table 40: usCurrentState values 46
Table 41: ECM_IF_SLAVE_CURRENT_STATE_RES – Response to master state indication 46
Table 42: Structure ECM_DIAG_ENTRY_T 47
Table 43: Structure ECM_DIAG_ENTRY_HEADER_T 48
Table 44: Possible values of usEntryType for diagnostic log 49
Table 45: Structure ECM_DIAG_ENTRY_NEW_STATE_T 50
Table 46: Structure ECM_DIAG_ENTRY_INTERNAL_ERROR_T 51
Table 47: Structure ECM_DIAG_ENTRY_IDENTITY_MISMATCH_T 52
Table 48: Meaning of usCompareFlags 53
Table 49: Structure ECM_DIAG_ENTRY_COE_INITCMD_FAILED_T 53
Table 50: Structure ECM_DIAG_ENTRY_SOE_INITCMD_FAILED_T 54
Table 51: Structure ECM_DIAG_ENTRY_REG_INITCMD_INFO_T 55
Table 52: Structure ECM_DIAG_ENTRY_REG_INITCMD_INFO_T 56
Table 53: Structure ECM_DIAG_ENTRY_ALCONTROL_FAILED_T 57
Table 54: Structure ECM_DIAG_ENTRY_SII_ASSIGN_FAILED_T 58
Table 55: Structure ECM_DIAG_ENTRY_SII_ASSIGN_FAILED_T 58
Table 56: Structure ECM_DIAG_ENTRY_SII_REQUEST_FAILED_T 59
Table 57: Structure ECM_DIAG_ENTRY_SLAVE_WARNING_T 60
Table 58: Available reason codes for ulWarningType 60
Table 59: Structure ECM_DIAG_ENTRY_SLAVE_ERROR_T 61
Table 60: Available reason codes for ulErrorType 61
Table 61: ECM_IF_CMD_READ_DIAG_LOG_ENTRY_REQ – Read diagnostic log entry service 62
Table 62: ECM_IF_CMD_READ_DIAG_LOG_ENTRY_CNF – Read diagnostic log entry confirmation 63

Table 63: ECM_IF_CMD_CLEAR_DIAG_LOG_REQ –Clear diagnostic log request 64
Table 64: ECM_IF_CMD_CLEAR_DIAG_LOG_CNF – Clear diagnostic log confirmation 64
Table 65: ECM_IF_CMD_DIAG_LOG_INDICATIONS_REGISTER_REQ – Register for diagnostic log indications request 67
Table 66: ECM_IF_CMD_DIAG_LOG_INDICATIONS_REGISTER_CNF – Register for diagnostic log indications confirmation 67
Table 67: ECM_IF_CMD_DIAG_LOG_INDICATIONS_UNREGISTER_REQ – Unregister from diagnostic log indications request 68
Table 68: ECM_IF_CMD_DIAG_LOG_INDICATIONS_UNREGISTER_CNF - Unregister from diagnostic log indications confirmation 68
Table 69: ECM_IF_CMD_NEW_DIAG_LOG_ENTRIES_IND – New diagnostic log entries available indication 69
Table 70: ECM_IF_CMD_NEW_DIAG_LOG_ENTRIES_RES – New diagnostic log entries available response 69
Table 71: ECM_IF_CMD_COE_SDO_DOWNLOAD_REQ – Download/write SDO request 74
Table 72: ECM_IF_CMD_COE_SDO_DOWNLOAD_CNF – Download/write SDO confirmation 75
Table 73: ECM_IF_CMD_COE_SDO_UPLOAD_REQ – Upload/read SDO request 76
Table 74: ECM_IF_CMD_COE_SDO_UPLOAD_CNF – Upload/read SDO confirmation 77
Table 75: ECM_IF_CMD_COE_SDOINFO_GETODLIST_REQ – Get object list request 86
Table 76: Possible values of usListType 87
Table 77: ECM_IF_CMD_COE_SDOINFO_GETODLIST_CNF – Get object list confirmation 88
Table 78: ECM_IF_CMD_COE_SDOINFO_GETOBJDESC_REQ – Get object description request 89
Table 79: ECM_IF_CMD_COE_SDOINFO_GETOBJDESC_CNF – Get object description confirmation 90
Table 80: ECM_IF_CMD_COE_SDOINFO_GETENTRYDESC_REQ – Get entry description request 91
Table 81: ECM_IF_CMD_COE_SDOINFO_GETENTRYDESC_CNF – Get entry description confirmation 92
Table 82: Meaning of bRequestedValueInfo and bValueInfo 93
Table 83: ETHERCAT_MASTER_CMD_SDO_DOWNLOAD_REQ – Download/write SDO request (Legacy) 97
Table 84: ETHERCAT_MASTER_CMD_SDO_DOWNLOAD_CNF – Download/write SDO confirmation (Legacy) 98
Table 85: ETHERCAT_MASTER_CMD_SDO_UPLOAD_REQ – Upload/read SDO request (Legacy) 99
Table 86: ETHERCAT_MASTER_CMD_SDO_UPLOAD_CNF – Upload/read SDO confirmation (Legacy) 100
Table 87: ETHERCAT_MASTER_CMD_GET_ODLIST_REQ – Get object list request (Legacy) 101
Table 88: Possible values of ulListType 101
Table 89: ETHERCAT_MASTER_CMD_GET_ODLIST_CNF – Get object list confirmation (Legacy) 102
Table 90: ETHERCAT_MASTER_CMD_GET_OBJECTDESC_REQ – Get object description request (Legacy) 103
Table 91: ETHERCAT_MASTER_CMD_GET_OBJECTDESC_CNF – Get object description confirmation (Legacy) 104
Table 92: ETHERCAT_MASTER_CMD_GET_ENTRYDESC_REQ – Get entry description request (Legacy) 105
Table 93: Parameter ulAccessBitMask 106
Table 94: ETHERCAT_MASTER_CMD_GET_ENTRYDESC_CNF – Get entry description confirmation (Legacy) 107
Table 95: Meaning of ulAccessBitMask and ulValueInfo 107
Table 96: ECM_IF_CMD_FOE_WRITE_REQ – Write file request (FoE, First segment) 113
Table 97: ECM_IF_CMD_FOE_WRITE_REQ – Write file request (FoE, Following segment) 113
Table 98: ECM_IF_CMD_FOE_WRITE_CNF – Write file confirmation 114
Table 99: ECM_IF_CMD_FOE_READ_REQ – Read file request (FoE, First segment) 116
Table 100: ECM_IF_CMD_FOE_READ_REQ – Read file request (FoE, Following segments) 116
Table 101: ECM_IF_CMD_FOE_READ_CNF – Read file confirmation (FoE) 117
Table 102: ECM_IF_CMD_SOE_WRITE_REQ – Write IDN request 124
Table 103: Meaning of bElementFlags 125
Table 104: ECM_IF_CMD_SOE_WRITE_CNF – Write IDN confirmation 126
Table 105: ECM_IF_CMD_SOE_READ_REQ – Read IDN request 127
Table 106: Meaning of bElementFlags 128
Table 107: ECM_IF_CMD_SOE_READ_CNF – Read IDN confirmation 129
Table 108: ECM_IF_CMD_GET_DC_DEVIATION_REQ – Get DC deviation information request 132
Table 109: ECM_IF_CMD_GET_DC_DEVIATION_CNF – Get DC deviation information confirmation 133
Table 110: Meaning of ulDcStatusFlags 134
Table 111: Meaning of sign/magnitude values 134
Table 112: ECM_IF_CMD_RESET_DC_MAX_DEVIATIONS_REQ – Reset DC max deviations request 135
Table 113: ECM_IF_CMD_RESET_DC_MAX_DEVIATIONS_CNF – Reset DC max deviations confirmation 135
Table 114: ECM_IF_CMD_GET_SLAVE_DC_INFO_REQ – Get slave DC information request 136
Table 115: ECM_IF_CMD_GET_SLAVE_DC_INFO_CNF – Get save DC deviation information confirmation 137
Table 116: Meaning of usFlags 138
Table 117: ETHERCAT_MASTER_CMD_GET_DC_DEVIATION_REQ – Get DC deviation request (Legacy) 139
Table 118: ETHERCAT_MASTER_CMD_GET_DC_DEVIATION_CNF – Get DC deviation confirmation (Legacy) 139
Table 119: ECM_IF_CMD_GET_TIMING_INFO_REQ – Get timing information request 140
Table 120: ECM_IF_CMD_GET_TIMING_INFO_CNF – Get timing information Confirmation 140
Table 121: Possible values of WcState bit 141
Table 122: ECM_IF_CMD_GET_WC_STATE_INFO_REQ – Get WcState information request 142
Table 123: Structure ECM_IF_WCSTATE_INFO_ENTRY_T 143
Table 124: ECM_IF_CMD_GET_WC_STATE_INFO_CNF – Get WcState information confirmation 144

Table 125: Possible values of usTransmitType and usReceiveType 145
Table 126: Data field definition of BRD ALStatus 146
Table 127: Data field definition of BRD DcSysTimeDiff 146
Table 128: ExtSync Status data structure 147
Table 129: Parameter ulExtSyncInfoFlags 148
Table 130: Parameter ulDcExtErrorDiffNsSignMag 148
Table 131: ECM_IF_CMD_GET_CYCLIC_CMD_MAPPING_REQ – Get cyclic command mapping request 149
Table 132: Structure ECM_IF_CYCLIC_CMD_MAPPING_ENTRY_T 150
Table 133: ECM_IF_CMD_GET_CYCLIC_CMD_MAPPING_CNF – Get cyclic command mapping confirmation 151
Table 134: Definition of usDirection 152
Table 135: ECM_IF_CMD_GET_CYCLIC_SLAVE_MAPPING_REQ – Get cyclic slave mapping request 153
Table 136: Structure ECM_IF_CYCLIC_SLAVE_MAPPING_ENTRY_T 154
Table 137: ECM_IF_CMD_GET_CYCLIC_SLAVE_MAPPING_CNF – Get cyclic slave mapping confirmation 155
Table 138: Slave connection information 158
Table 139: ulCurrentState in Slave connection information 159
Table 140: HIL_GET_SLAVE_HANDLE_REQ – Get slave handle request 160
Table 141: HIL_GET_SLAVE_HANDLE_CNF – Confirmation of get slave handle request 161
Table 142: ECM_IF_CMD_GET_SLAVE_HANDLE_BIT_LIST_REQ – Get slave handle bit list request 162
Table 143: ECM_IF_CMD_GET_SLAVE_HANDLE_BIT_LIST_CNF – Confirmation of get save handle request 163
Table 144: HIL_GET_SLAVE_CONN_INFO_REQ – Get slave connection information request 164
Table 145: HIL_GET_SLAVE_CONN_INFO_CNF – Confirmation of get slave connection information request 165
Table 146: ETHERCAT_MASTER_CMD_READ_EMERGENCY_REQ – Get slave CoE emergencies request 166
Table 147: ETHERCAT_MASTER_CMD_READ_EMERGENCY_CNF – Confirmation of get slave CoE emergencies request 167
Table 148: Structure of ETHERCAT_MASTER_SLAVE_EMERGENCY_T 167
Table 149: Topology information entry 168
Table 150: ECM_IF_CMD_GET_TOPOLOGY_INFO_REQ – Get topology information request 169
Table 151: ECM_IF_CMD_GET_TOPOLOGY_INFO_CNF – Get topology information confirmation 169
Table 152: ECM_IF_CMD_READ_REGS_REQ – Read ESC registers request 170
Table 153: ECM_IF_CMD_READ_REGS_CNF – Read ESC registers confirmation 171
Table 154: ECM_IF_CMD_WRITE_REGS_REQ – Write ESC registers request 172
Table 155: ECM_IF_CMD_WRITE_REGS_CNF – Write ESC registers confirmation 173
Table 156: ECM_IF_CMD_READ_SII_REQ – Read SII/EEPROM request 174
Table 157: ECM_IF_CMD_READ_SII_CNF – Read SII/EEPROM confirmation 175
Table 158: ECM_IF_CMD_WRITE_SII_REQ – Write SII/EEPROM request 176
Table 159: ECM_IF_CMD_WRITE_SII_CNF – Write SII/EEPROM confirmation 177
Table 160: ETHERCAT_MASTER_CMD_EEPROM_READ_REQ – Read SII/ EEPROM request (Legacy) 179
Table 161: ETHERCAT_MASTER_CMD_EEPROM_READ_CNF – Read SII/EEPROM confirmation (Legacy) 180
Table 162: ETHERCAT_MASTER_CMD_EEPROM_WRITE_REQ – Write SII/EEPROM request (Legacy) 182
Table 163: ETHERCAT_MASTER_CMD_EEPROM_WRITE_CNF – Write SII/EEPROM confirmation (Legacy) 182
Table 164: ECM_IF_CMD_GET_EXT_SYNC_INFO_REQ – Get ExtSync information request 184
Table 165: ECM_IF_CMD_GET_EXT_SYNC_INFO_CNF – Get ExtSync information confirmation 185
Table 166: Parameter ulExtSyncInfoFlags 186
Table 167: ECM_IF_RESET_EXT_SYNC_MAX_DEVIATIONS_REQ – Reset Max Deviations request 187
Table 168: ECM_IF_CMD_RESET_EXT_SYNC_MAX_DEVIATIONS_CNF – Reset Max Deviations confirmation 187
Table 169: ECM_IF_SELECT_SYNC_CONFIG_REQ – Select sync configuration request 189
Table 170: ulSyncModuleId values 189
Table 171: ulTimeSyncModuleId values 190
Table 172: Possible combinations of ulSyncModuleId and ulTimeSyncModuleId 190
Table 173: ECM_IF_SELECT_SYNC_CONFIG_CNF – Set sync configuration confirmation 190
Table 174: ECM_IF_ENUM_SYNC_CONFIG_REQ – Enumerate sync configurations request 191
Table 175: ECM_IF_ENUM_SYNC_CONFIG_CNF – Enumerate sync configurations confirmation 192
Table 176: ulEntryType values 193
Table 177: ECM_IF_GET_ERROR_COUNTERS_REQ – Get error counters request 194
Table 178: ECM_IF_GET_ERROR_COUNTERS_CNF – Get error counters confirmation 195
Table 179: Meaning of ulPortState 196
Table 180: HIL_GET_DEVICE_INFO_REQ – Get device info request 199
Table 181: HIL_GET_DEVICE_INFO_CNF – Get device info confirmation 200
Table 182: ETHERCAT_MASTER_CMD_START_BUS_SCAN_REQ – (Re)start the bus scan request 203
Table 183: ETHERCAT_MASTER_CMD_START_BUS_SCAN_CNF – (Re)start the bus scan confirmation 204
Table 184: ETHERCAT_MASTER_CMD_GET_BUS_SCAN_INFO_REQ – Get results from bus scan request 205
Table 185: ETHERCAT_MASTER_CMD_GET_BUS_SCAN_INFO_CNF – Get results from bus scan confirmation 206
Table 186: EtherCAT Master Tag List Parameters 207
Table 187: EtherCAT Master packet status codes 213
Table 188: EtherCAT Master V4 LLD error codes 216
Table 189: EtherCAT Master V4 EMC error codes 223

Table 190: EtherCAT Master V4 AoE error codes 225
Table 191: EtherCAT Master V4 CoE error codes 228
Table 192: EtherCAT Master V4 EoE error codes 229
Table 193: EtherCAT Master V4 FoE error codes 230
Table 194: EtherCAT Master V4 SoE error codes 233
Table 195: EtherCAT Master V4 ENI error codes 238
Table 196: EtherCAT Master V4 AL Status codes 241
Table 197: EtherCAT Master V4 IF error codes 242
Table 198: EtherCAT Master V4 AP Task error codes 242
Table 199: Names of Queues in the EtherCAT Master Firmware 243
Table 200: Extended Status Block 244

7.6 Contacts

Headquarters

Germany
Hilscher Gesellschaft für
Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr

Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it

Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp

Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us

Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com