

EtherCAT Slave

Protocol API V4.9

DOC110909APIV4.9.1.0EN | Revision V4.9.1.0 | English | Preliminary | Public | 2024-03-07



Table of Contents

1 Introduction	5
1.1 About this document	5
1.2 Functional Overview	5
1.3 System requirements	5
1.4 Intended audience	5
1.5 Technical data	6
1.6 Terms, abbreviations and definitions	7
1.7 References to documents	8
2 Getting started	9
2.1 Stack types	9
2.1.1 Loadable Firmware (LFW)	9
2.1.2 Linkable Object Module (LOM)	9
2.2 Configuring the EtherCAT stack	10
2.2.1 Configuration methods	10
2.2.2 Sequence and priority of configuration evaluation	10
2.2.3 Configuration parameters	10
2.2.4 Application sets the configuration parameters	11
2.2.5 Configuration software	12
2.3 Cyclic data exchange - Process data input and output	12
2.3.1 BusOn / BusOff	13
2.4 Acyclic data exchange	13
2.5 Object dictionary	13
3 Stack structure and stack functions	15
3.1 Structure of the EtherCAT Slave stack	15
3.2 Base component	16
3.2.1 ESM task	16
3.2.2 Slave Information Interface (SII)	19
3.2.3 MBX task	20
3.3 CoE component	21
3.3.1 CoE task	21
3.3.2 SDO task	22
3.3.3 ODV3 task	22
3.4 FoE component	25
3.5 Behavior when receiving a Set Configuration command	26
3.6 Watchdogs	26
3.6.1 DPM watchdog	26
3.6.2 SM Watchdog	26
3.6.3 PDI Watchdog	27
3.7 Usage of PHYs	27
4 Status information	28
4.1 Common status	28
4.2 Extended status	28
5 Requirements to the application	29
5.1 Sequence within the host application	29
5.2 General initialization sequence	31
5.3 Explicit Device Identification	32
5.3.1 Initialization sequence	32
5.3.2 Set firmware parameter	34
5.3.3 Required entry in ESI file for Explicit Device Identification	35
5.4 Complete Access for object data held by application	36
5.5 Dynamic PDO mapping	37
5.5.1 One application registered (application successful)	38
5.5.2 One application registered (application not successful)	39
5.5.3 Multiple applications registered (one application not successful)	40
5.5.4 One application registered Complete Access: (application successful)	40
5.6 Protocol-specific aspects to regard for ODV3 API when using the EtherCAT Slave stack	42
5.6.1 ODV3 access mask and flags	42



5.6.2 Free memory available for ODV3 objects can decrease after firmware update	42
6 Application interface	43
6.1 General.....	44
6.1.1 Register application service	44
6.1.2 Unregister application service.....	44
6.1.3 Set ready service.....	44
6.1.4 Initialization complete service	46
6.1.5 Link status changed service.....	47
6.2 Configuration	48
6.2.1 Set configuration service.....	48
6.2.2 Set extended configuration service.....	60
6.2.3 Set handshake configuration service	61
6.2.4 Set IO Size service.....	61
6.2.5 Set Station Alias service	62
6.2.6 Get Station Alias service.....	63
6.2.7 Relation between Set configuration packet and ESI file.....	63
6.3 EtherCAT state machine.....	65
6.3.1 Register for AL control changed indications service	66
6.3.2 Unregister from AL control changed indications service.....	66
6.3.3 AL control changed service.....	67
6.3.4 AL status changed service	69
6.3.5 Set AL status service	70
6.3.6 Get AL status service.....	71
6.4 CoE.....	72
6.4.1 Send CoE emergency service.....	72
6.5 Packets for Object Dictionary access	73
6.6 Slave Information Interface (SII in virtual EEPROM)	73
6.6.1 SII read service	74
6.6.2 SII write service	74
6.6.3 Register for SII write Indications service	75
6.6.4 Unregister from SII write indications service.....	76
6.6.5 SII write Indication service	76
6.7 Ethernet over EtherCAT (EoE)	78
6.7.1 Register for frame indications service.....	79
6.7.2 Unregister from frame indications service.....	79
6.7.3 Ethernet send frame service	80
6.7.4 Ethernet frame received service	81
6.7.5 Register for IP parameter indications service.....	82
6.7.6 Unregister from IP parameter Indications service	83
6.7.7 Set IP parameter service	84
6.7.8 Get IP parameter service	86
6.8 File Access over EtherCAT (FoE).....	87
6.8.1 Set FoE options service.....	88
6.8.2 FoE register file service	89
6.8.3 FoE unregister file service.....	91
6.8.4 FoE write file service	92
6.8.5 FoE read file service	93
6.8.6 FoE file written Service.....	96
6.8.7 FoE File Write Aborted Service	97
6.9 ADS over EtherCAT (AoE).....	98
6.9.1 AoE register port service.....	99
6.9.2 AoE unregister port service.....	99
6.10 Vendor-specific protocol over EtherCAT (VoE)	100
6.10.1 Mailbox register type service	100
6.10.2 Mailbox unregister type service.....	101
6.10.3 Mailbox service	101
6.10.4 Mailbox send service	102
7 Special topics.....	104
7.1 For programmers.....	104



7.2 Getting the receiver task handle of the process queue.....	104
8 Status and error codes.....	105
8.1 Error LED	105
8.2 SDO abort codes	105
8.2.1 SDO abort codes	106
8.3 Correspondence of SDO abort codes and status / error codes	107
8.4 CoE emergency codes	108
Appendix A: Appendix	109
A.1 List of figures	109
A.2 List of tables.....	110
A.3 List of snippets	114
A.4 Legal Notes	115
A.5 Contacts.....	119

Chapter 1 Introduction

1.1 About this document

This manual describes the application interface of the EtherCAT Slave protocol stack V4 and provides information about how an application has to use the EtherCAT Slave protocol stack.

1.2 Functional Overview

The stack has been written in order to meet the IEC 61158 Type 12 specification. The following features are implemented in the stack:

- EtherCAT Base Component
 - HAL initialization of the associated EtherCAT interface
 - EtherCAT interrupt handling
 - EtherCAT State Machine
 - Mailbox Receive handling
 - Mailbox Send handling
- CANopen over EtherCAT Component
 - Master-to-Slave SDO communication
 - Object dictionary
 - Complete Access (supported from stack version 4.3)
- Ethernet over EtherCAT Component
- File Access over EtherCAT Component
- Slave-to-slave communication only for netX 50, netX 51, and netX 52
- Slave-to-slave communication in same EtherCAT cycle only possible with netX 50, netX 51, netX 52 using LOM

1.3 System requirements

This software package has the following system requirements to its environment:

- netX chip as CPU hardware platform
- operating system for task scheduling required

1.4 Intended audience

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

- Knowledge of the programming language C
- Knowledge of the Hilscher Task Layer Reference Model

Further knowledge in the following areas might be useful:

- Knowledge of the IEC 61158 Part 2-6 Type 12 specification documents
- Knowledge of the IEC 61800-7-300
- Knowledge of the IEC 61800-7-204

Software developers working with Linkable Object Modules should additionally have:

- Knowledge of the use of the realtime operating system rcX

1.5 Technical data

NOTE | For technical details, each firmware comes with a separate, chip specific data sheet document.

The data below applies to EtherCAT Slave firmware and contains some general information.

supported chips	netX50 netX51 netX52 netX100,netX500
Supported protocols	SDO server side protocol (Acyclic communication SDO Master-Slave) (CoE component) CoE emergency messages (CoE component) Ethernet over EtherCAT (EoE component) File Access over EtherCAT (FoE component) AoE (supported from stack version 4.3) SoE (supported from stack version 4.8, SoE and CoE cannot be used at the same time)
Type	Complex Slave
Licensing	As this is a slave protocol stack, there is no license required
Configuration	by packet or database (not supporting all features)
Diagnostic	Firmware supports common diagnostic in the dual-port-memory for loadable firmware.

1.6 Terms, abbreviations and definitions

Term	Description
ADS	Automation Device Specification
AL	Application layer
AoE	ADS over EtherCAT
AP (-task)	Application (-task) on top of the stack
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CoE	CANopen over EtherCAT
COS	Change of State
DC	Distributed Clocks
DL	Data Link Layer
DPM	Dual port memory
E2PROM (EEPROM)	Electrically erasable Programmable Read-only Memory
EoE	Ethernet over EtherCAT
ESC	EtherCAT Slave Controller
ESM	EtherCAT State Machine
ETG	EtherCAT Technology Group
EtherCAT	Ethernet for Control and Automation Technology
FMMU	Fieldbus Memory Management Unit
FoE	File Access over EtherCAT
IEEE	Institute of Electrical and Electronics Engineers
LFW	Loadable firmware
LOM	Linkable object modules
LSB	Least significant byte
MSB	Most significant byte
OD	Object dictionary
ODV3	Object dictionary Version 3
PHY	Physical Interface (Ethernet)
PDO	Process Data Object (process data channel)
RTR	Remote Transmission Request
RxPDO	Receive PDO
SDO	Service Data Object (representing an acyclic data channel)
SHM	Shared memory
SM	Sync Manager
SoE	Servodrive Profile over EtherCAT
SSC	SoE Service Channel
TxPDO	Transmit PDO
VoE	Vendor Profile over EtherCAT
XML	eXtensible Markup Language

1.7 References to documents

This document refers to the following documents. The referenced standard documents are available from ETG. Additionally to these, there is a knowledgebase side for ETG which we suggest to check if specification related questions occur. It contains explanations or clarifications to the EtherCAT specification.

- [1] Hilscher Gesellschaft für Systemautomation mbH: [Dual-Port Memory Interface Manual, Revision 17, English, 2020-06.](#)
- [2] Hilscher Gesellschaft für Systemautomation mbH: [Packet API, netX Dual-Port Memory, Packet-based services, Revision 4, English, 2020-06.](#)
- [3] Hilscher Gesellschaft für Systemautomation mbH: [netX EtherCAT Slave HAL Documentation](#)
- [4] Hilscher Gesellschaft für Systemautomation mbH: [Object Dictionary V3 Protocol API, Revision 4, English, 2017.](#)
- [5] Hilscher Gesellschaft für Systemautomation mbH: [Protocol API, Socket Interface, Packet Interface, Revision 5, English, 2019.](#)
- [6] Hilscher Gesellschaft für Systemautomation mbH: [Protocol API, TCP/IP, Packet Interface, Revision 17, English, 2020-10.](#)
- [7] IEC 61158 Part 2-6 Type 12 documents (also available for members of EtherCAT Technology Group as specification documents ETG-1000)
- [8] IEC 61800-7
- [9] EtherCAT Specification Part 5 – Application Layer services specification. ETG1000.5
- [10] EtherCAT Specification Part 6 – Application Layer protocol specification. ETG1000.6
- [11] EtherCAT Protocol Enhancements. ETG1020
- [12] EtherCAT Indicator and Labeling Specification. ETG1300
- [13] EtherCAT Slave Information Specification ETG2000
- [14] EtherCAT Slave Information Annotation ETG2001
- [15] Slave Information Interface (SII) ETG2010
- [16] Modular Device Profile ETG5001
- [17] Semiconductor Device Profile ETG5003
- [18] Conformance test Specification ETG7000

Chapter 2 Getting started

This chapter describes the basics of the Hilscher EtherCAT Slave stack. This includes information about

- use cases and stack types (LFW/LOM)
- the configuration of the EtherCAT Slave stack
- principles of cyclic and acyclic data exchange
- object dictionary

2.1 Stack types

The EtherCAT Slave protocol stack can be used in two different use cases:

- Loadable Firmware (LFW)
- Linkable Object Modules (LOM)

2.1.1 Loadable Firmware (LFW)

The application and the EtherCAT Slave Protocol Stack run on different processors. While the host application runs on a computer typically equipped with an operating system (such as Microsoft Windows® or Linux), the EtherCAT Slave Protocol Stack runs on the netX processor together with a connecting software layer, the AP task. The connection is accomplished via a driver (Hilscher cifX Driver, Hilscher netX Driver) as software layer on the host side and the AP task as software layer on the netX side. Both communicate via a dual port memory (DPM), into which they both can write and from which they both can read. This situation is shown in Figure 1:

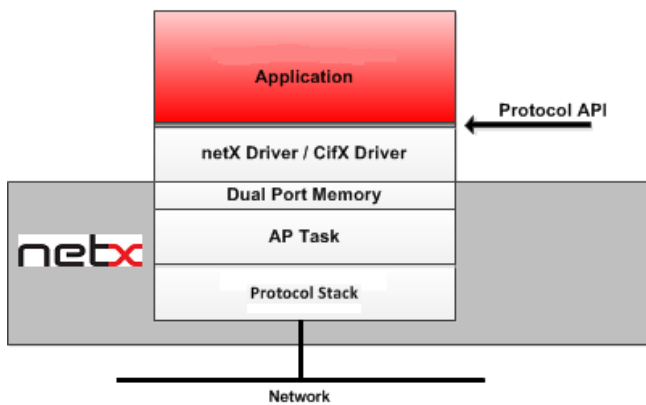


Figure 1. Usecase loadable Firmware

2.1.2 Linkable Object Module (LOM)

Both, the application and the EtherCAT Slave Protocol Stack are executed on netX. There is no need for drivers or a stack-specific AP task. Application and protocol stack are statically linked.

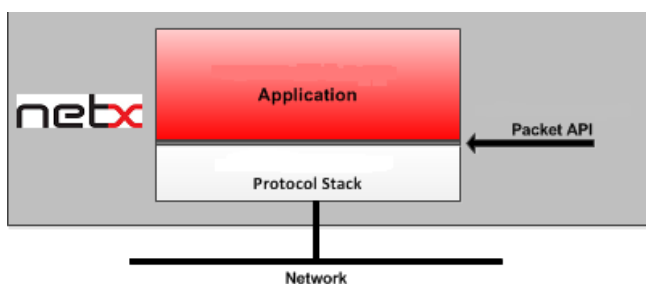


Figure 2. Usecase loadable Firmware

If the stack is used as Linkable Object Module, the user has to create its own configuration file (which among others contains task start-up parameters and hardware resource declarations).

2.2 Configuring the EtherCAT stack

2.2.1 Configuration methods

You can use one of the following methods to configure the EtherCAT Slave stack:

- The application can set the configuration parameters using the [Set configuration service](#) to transfer the parameters within a packet to the stack.
- You can use one of the following configuration softwares to set the configuration parameters.
 - You can use SYCON.net configuration software (which creates a configuration file named CONFIG.NXD)
 - You can use the netX Configuration Tool (which creates a file named INIBATCH.NXD)

2.2.2 Sequence and priority of configuration evaluation

The EtherCAT Slave stack has implemented the following sequence and priority of configuration evaluation:

1. In case the file CONFIG.NXD is available, the stack will use the configuration parameters from this file and starts working.
2. In case the file INIBATCH.NXD is available, the operating system rcX will send the configuration parameters from this file to the EtherCAT Slave stack and the stack starts working.
3. The stack “waits” for the configuration parameters and remains unconfigured. The application has to use the [Set configuration service](#) to configure the EtherCAT Slave and a Channel Init service to activate the configuration parameters.

2.2.3 Configuration parameters

Basic configuration parameters

The basic configuration parameters set the values for e.g. startup behavior of the stack, the vendor id, product code, etc. The application has to set these parameters with the [Set configuration service](#).

Component configuration parameters

The EtherCAT Slave stack consists of several components. Each component has its own parameters (configuration structure).

The following table lists the components of the stack.

Component	Meaning
AoE	ADS over EtherCAT Data structure for configuration of AoE component actually just reserved: ECAT_ESM_CONFIG_AOE_T
CoE	CANopen over EtherCAT Data structure for configuration of CoE component: ECAT_SET_CONFIG_COE_T
EoE	Ethernet over EtherCAT Data structure for configuration of EoE component: ECAT_SET_CONFIG_EOE_T
FoE	File Access over EtherCAT Data structure for configuration of FoE component: ECAT_SET_CONFIG_FOE_T
SoE	Servodrive Profile over EtherCAT Data structure for configuration of SoE component: ECAT_SET_CONFIG_SOE_T
Sync Modes	Synchronization modes Data structure for configuration of Sync Modes component: ECAT_SET_CONFIG_SYNCMODES_T

Component	Meaning
Sync PDI	Process data interface for synchronization Data structure for configuration of Sync PDI component: ECAT_SET_CONFIG_SYNCPDI_T
UID	Unique Identification Data structure for configuration of UID component: ECAT_SET_CONFIG_UID_T
Boot Mbx	Boot mailbox Data structure for configuration of Bootmailbox component: ECAT_SET_CONFIG_BOOTMBX_T
Device Info	Device information Data structure for configuration of Device Info component ECAT_SET_CONFIG_DEVICEINFO_T
Sm Length	Sync Manager Data structure for configuration of Syncmanagers address spaces ECAT_ESM_CONFIG_SMLENGTH_DATA_T

Table 1. Component configuration parameters

These data structures need only be filled with data if they are used and evaluated. This depends on the flags within parameter Component Initialization of the Base Configuration Parameters described above. Each flag controls whether the data structure for a single component is evaluated (flag set) or not (flag equals 0). Please refer to chapter [Set configuration service](#) for a detailed programming reference.

Extended configuration parameters

Since version V4.10 the stack supports an extended configuration packet that offers the opportunity to introduce new configuration parameters without increasing the size of the configuration packet itself. The extended configuration has to be send **after** a standard (basic) configuration packet. Its values apply after sending the Channel Init (Bus on).

The content of an extended configuration packet is defined by its type. It is possible to send several packets with different types. Configurations overwrite configurations of the same type which were send before. Possible types of extended configuration are shown the following list. Details are in chapter [Set extended configuration service](#).

Extended Configuration Type	Meaning
ECAT_SET_CONFIG_STRUCTURE_TYPE_SMS	Changes the standard mailbox start addresses and size. This packet is mainly used for a device replacement which requires different start addresses to keep the ESI file consistent to the former device (not from hilscher).
ECAT_SET_CONFIG_STRUCTURE_TYPE_BOOTMBX	Sets bootstrap mailbox start addresses and size. This packet is mainly used for a device replacement which requires different start addresses to keep the ESIfile consistent to the former device (not from hilscher).

Table 2. Extended configuration parameters

2.2.4 Application sets the configuration parameters

In case the application configures the EtherCAT Slave, the application has to perform the following steps:

1. Configure the device using the [Set configuration service](#). This provides the device with all parameters needed for operation. These include both **basic parameters** for I/O sizes and for identification such as Vendor ID and Product code as well as the **component configuration**. When the stack confirms the Set Configuration to the application, the given configuration has been evaluated completely and prepared for being applied.
If several configuration packets are sent from the application to the stack, the stack uses the last received configuration packet before the application sends a Channel Init.
2. (optional) Configure the device by sending one ore more **extended configurations** using the [Set extended configuration service](#). Sending a standard configuration (1) again after sending extension configurations will delete all extended configuration parameters.
3. Perform the **Channel Init** (for further information, see reference [1]) to activate the configuration parameters. As a result, the stack is ready to start communication with an EtherCAT Master. A Channel Init does not unregister already registered services.

Figure 3 shows the Set Configuration and Channel Init sequence.

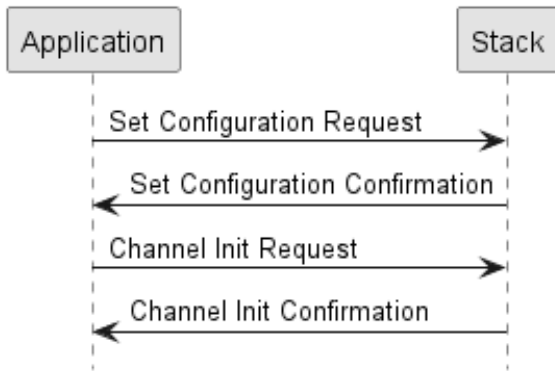


Figure 3. Set Configuration / Channel Init

2.2.4.1 Reconfiguration

It is possible to reconfigure the stack at any time. To do so, simply send a new configuration to the stack followed by a Channel Init request [1]. Sending the new configuration without the Channel Init request will not have an effect to the running communication. The new parameters will be stored in the RAM only. Sending the Channel Init request will stop any communication and take over the new parameters. A Channel Init does not unregister already registered services. As an alternative for complete reconfiguration, specific parameters can be reconfigured separately:

- The IO Size: See section Set IO Size service which contains two parameters only: input length and output length. Thus, it is not necessary to send a complete configuration packet twice
- The Synchronisation parameters: Needed to change the synchronization mode in case, more than one synchronization mode supported by the device. See section `_set_handshake_configuration_service`.

2.2.4.2 Delete configuration

The deletion of the configuration is not possible in EtherCAT because this would stop the physical interface to work and thus break the logical ring structure and cut the communication with the following slaves in the topology.

2.2.4.3 Configuration lock

If the configuration of the stack is locked as described in [1], the following behavior is implemented in the stack:

- new configuration packets are not accepted.
- a Channel Init Request will be rejected.

2.2.5 Configuration software

The configuration via SYCON.net or netX Configuration Tool are described in separate manuals.

2.3 Cyclic data exchange - Process data input and output

This section describes how the application can get access to the cyclic IO data which is exchanged with the EtherCAT Master. The EtherCAT Slave stack provides different ways to exchange this data. Depending on the user's application only one of these methods may be used:

- Use case **loadable firmware**: If the netX chip is used as dedicated communication processor while the user's application runs on its own host processor, I/O data can only be accessed using the mechanism described in the Dual-Port Memory Interface Manual [1].
- Use case **linkable object modules**: If the application is executed on the netX chip together with the EtherCAT Slave Stack there exist two possibilities to access the cyclic I/O data:
 - If the Shared Memory Interface is used, the application has to access the I/O data using the shared memory interface API. As this is basically an emulation of the dual-port memory interface for applications running local on the netX chip, the interface is similar to using the netX as dedicated communication processor.
 - If the user application is not using the shared memory interface, the I/O data is accessed using a function call API. This approach is also known as "packet API". It removes any overhead from the Shared Memory Interface.

EtherCAT uses the concept of a cyclic process data image. Each master or slave of an EtherCAT network has an image of

input and output data. This image is updated using cyclic Ethernet frames. More information on how cyclic data exchange is accomplished with suitable PDO mappings can be found at subsection [PDO mapping for cyclic communication](#).

Input and output data of EtherCAT Slave for netX 100, 500

Offset in ESC	Area	Length (byte)	Type
0x1000	Output block	512	Read/Write
0x2680	Input block	512	Read/Write

Table 3. Input and output data netX 100, 500

Input and output data of EtherCAT Slave for netX 50, 55, 52

Offset in ESC	Area	Length (byte)	Type
0x1000	Output block	1024	Read/Write
0x2680	Input block	1024	Read/Write

Table 4. Input and output data netX 100, 500

2.3.1 BusOn / BusOff

The BusOn/Off bit controls whether the stack is allowed to proceed further than Pre-Operational state. If the bit is set, the stack can be brought into Operational state by the master e.g. TwinCAT. If the bit is cleared, the stack will fall back to Pre-Operational state and notify the master about this by setting the code `ECAT_AL_STATUS_CODE_HOST_NOT_READY` in the AL status Code area.

```
#define ECAT_AL_STATUS_CODE_HOST_NOT_READY 0x8000
```

For a list of available AL status Codes please refer to the `EcsV4_Public.h` file.

2.4 Acyclic data exchange

Acyclic communication: Application to EtherCAT Slave

The EtherCAT Slave stack uses two mailboxes in the dual-port memory to communicate (acyclic communication) with the application.

This acyclic communication via the dual-port memory is done through channels which each have two mailboxes. A Send Mailbox for transfer from host system to firmware or and a Receive Mailbox transfers from firmware to host system. Each mailbox can hold one packet at a time. The netX firmware stores packets that are not retrieved by the host application in a packet queue.

NOTE | The packet queue has limited space and may fill up so new packets maybe lost. To avoid these data loss situations, it is strongly recommended to empty the mailbox frequently, even if packets are not expected by the host application.

Acyclic communication: EtherCAT Master to EtherCAT Slave via Service Data Objects

For acyclic data exchange between an EtherCAT Slave and an EtherCAT Master the EtherCAT mailbox is used. Acyclic data exchange is done via Service Data Objects. These objects are managed by the ODV3 task. For more information refer to the separate ODV3 documentation [4].

2.5 Object dictionary

The EtherCAT Slave uses objects to hold values and device parameters. Those objects can be accessed by the application and also from the EtherCAT Master which is reaching them via the EtherCAT network. The stack can be used with the default object dictionary or with a custom object dictionary.

Default object dictionary

The default object dictionary contains all objects that are necessary to bring the slave to operational state. In the default object dictionary, the objects which define the process input data and the process output data, are handled as single bytes and each byte is represented by a subobject in the object dictionary.

If a configuration software is used to configure the EtherCAT Slave device, the default object dictionary has to be used.

See Section [Default object dictionary](#) for detail of the usecase.

Custom object dictionary

The custom object dictionary can be used to structure the process input data and the process output data with several/different data types e.g. to use data type UINT32. This requires that the application program configures the stack. The custom object dictionary contains only a minimal object dictionary and the application has to add all mandatory objects for EtherCAT and thw objects required by the application.

Section [Custom object dictionary based on minimal object directory](#) describes the minimal object dictionary.

Chapter 3 Stack structure and stack functions

3.1 Structure of the EtherCAT Slave stack

The following figure shows the internal structure of the EtherCAT Slave stack.

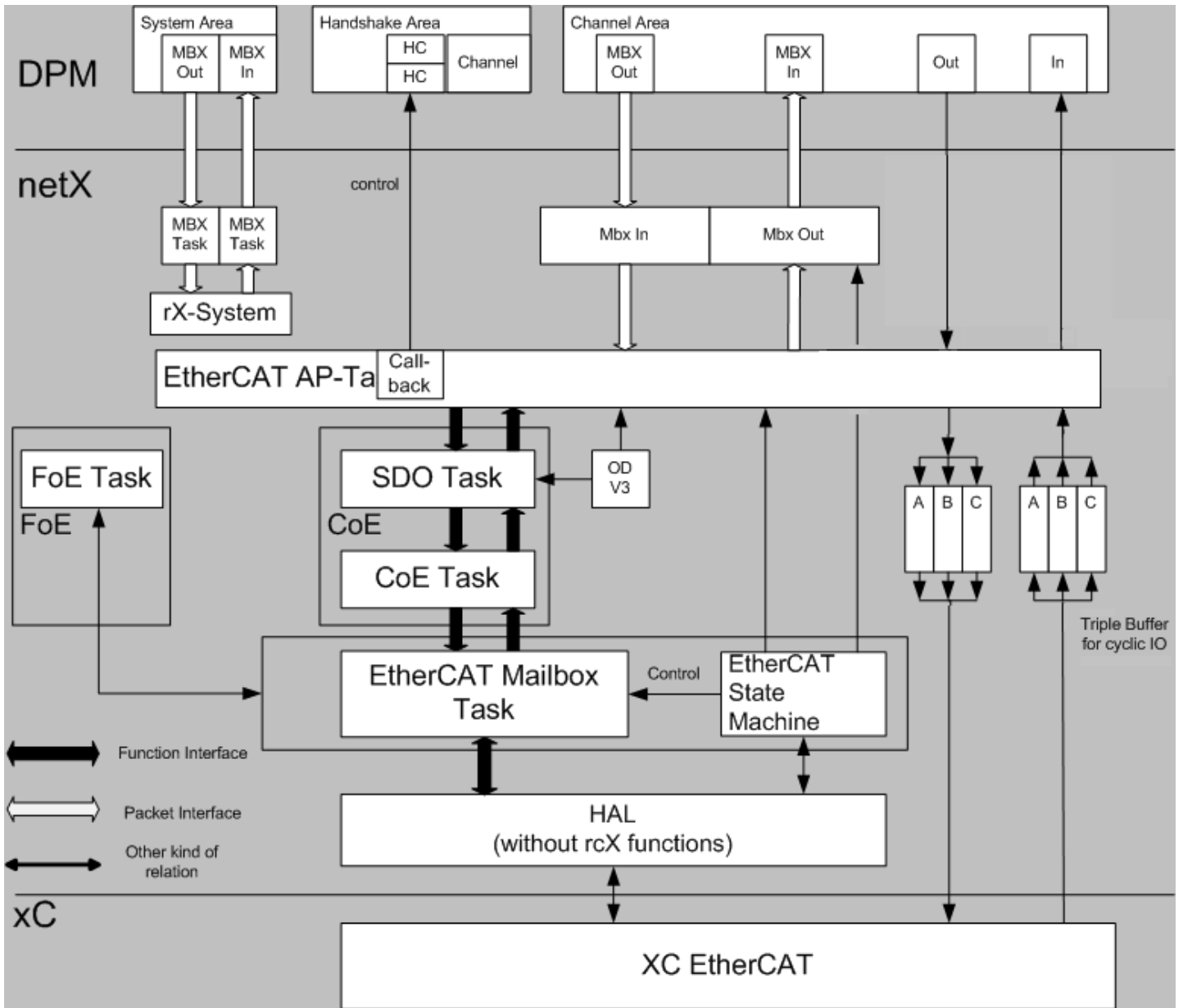


Figure 4. Firmware structure

The single tasks provide the following functionality:

- The AP task represents the interface between the EtherCAT Slave protocol stack and the dual-port memory and is responsible for:
 - Control of LEDs
 - Diagnosis
 - Packet routing
 - Update of the IO data
- The EtherCAT state machine task (ESM task) manages the states and operation modes of the protocol stack, generates AL control events, and sends them to all registered receivers.
- The EtherCAT Mailbox/DL task (MBX task) provides the low-level part of data communication.
- The SDO task is used to perform SDO communication via mailboxes, i.e. acyclic communication such as service requests.

- The CoE task handles the CoE related mailbox messages and routes them to the appropriate tasks. In addition, the CoE task provides a mechanism for sending CoE emergency messages.
- The ODV3 task handles access to the object dictionary (acyclic communication).

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.

You can find information about the various tasks:

- In section [ESM task](#)
- In section [MBX task](#)
- In section [CoE task](#)
- In section [SDO task](#)
- In reference [\[4\]](#) for the ODV3 task

In the use case “loadable firmware”, the dual-port memory is used to exchange information, data and packets. The EtherCAT Slave AP task takes care of mapping the EtherCAT Stack API to the Dual-Port-Memory. The application only accesses the AP.

Overview

The main topics described in this chapter are:

- Base Component
- CoE Component
- EoE Component
- FoE Component

3.2 Base component

3.2.1 ESM task

The ECAT_ESM task (ECAT_ESM Task) coordinates all tasks that have registered themselves with their respective queues as AL control event receivers. Additionally, it notifies the mailbox associated tasks of the current state and sets their operation modes.

3.2.1.1 EtherCAT State Machine (ESM)

Purpose

The states and state changes of the slave application can be described by the EtherCAT State Machine (ESM). The ESM implements the following four states which are precisely described in the EtherCAT specification (see there for reference):

- **Init:** The EtherCAT Slave is initialized in this state. No real process data exchange happens.
- **Pre-Operational:** Initialization of the EtherCAT Slave continues. No real process data exchange happens. The master and the slave communicate acyclically via mailbox to set parameters.
- **Bootstrap** is an optional state, it’s purpose is mainly the firmwareupdate. Like Pre-Operational it supports mailbox communication and no process data.
- **Safe-Operational:** In this state, the EtherCAT Slave can process input data. However, the output data are set to a ‘safe’ state.
- **Operational:** In this state, the EtherCAT Slave is fully operational.

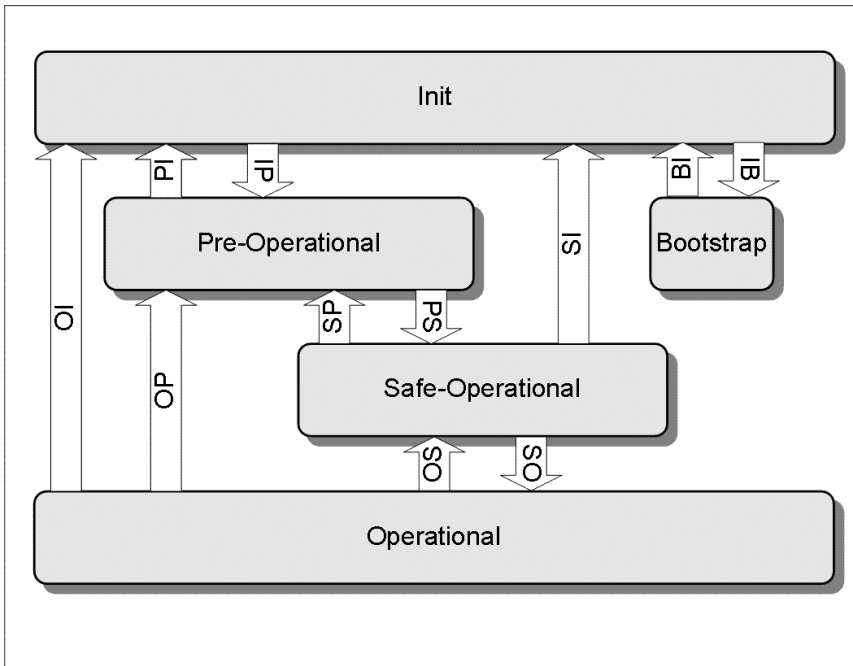


Figure 5. State diagram of EtherCAT State Machine (ESM)

Closely connected to the ESM are the AL control register and the AL status register of the EtherCAT Slave [10].

3.2.1.2 AL control register and AL status register

- The AL control register contains the requested state of the EtherCAT slave.
- The AL status register contains the current state of the EtherCAT slave.

Handling and controlling the EtherCAT State Machine

The AL control register and the AL status register provide a synchronization mechanism for state transitions between the master and the slave. They are precisely described in the EtherCAT specification, see there for more information.

AL status register related

The Hilscher EtherCAT slave stack provides mechanisms for user applications to get informed about state changes of the EtherCAT State Machine (ESM). Furthermore an application can control state changes of the ESM if necessary. Such mechanisms are needed for the realization of complex EtherCAT slaves [9]. If an application wants to get informed about state changes it has to register via **RCX_REGISTER_APP_REQ**. As result the stack will send an **ECAT_ESM_ALSTATUS_CHANGED_IND** to the application.

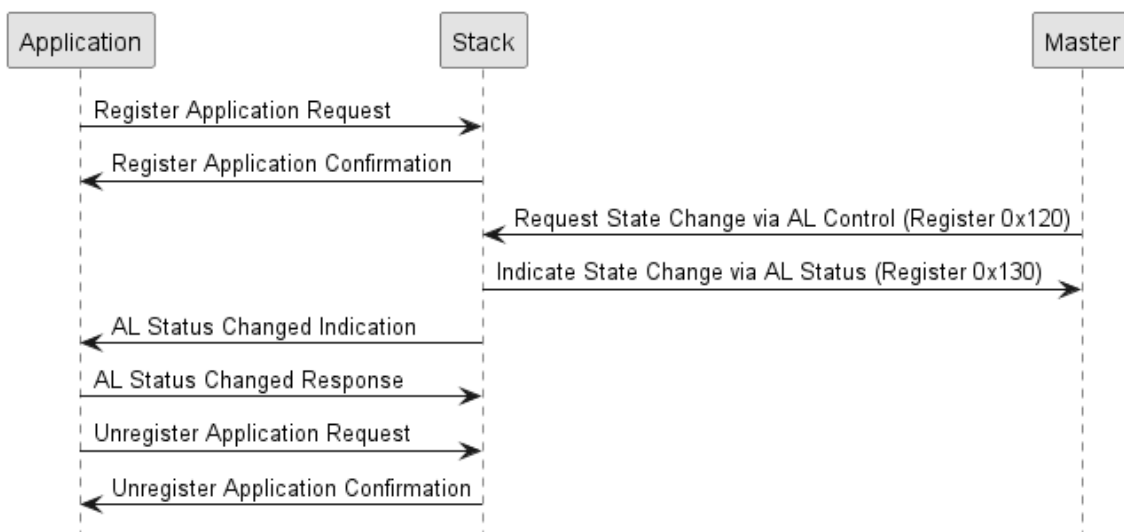


Figure 6. Sequence diagram of state change with indication to application/host

The packets mentioned above indicate that a state change has already happened. An application has no chance to control

or interrupt a transition; it just gets informed about it.
To unregister use the RCX_UNREGISTER_APP_REQ packet.

AL Control register related

If an application additionally wants to control ESM state changes it has to register for AL Confirmed Services.+ Registering for AL Confirmed Services may be necessary e.g. in following cases:

- Servo Drive with use of Distributed Clock (Synchronization)
In Motion Control applications it is of utmost importance that all devices work synchronized. Therefore drives often use a Phased Locked Loop (PLL) to synchronize their local control loop with the bus cycle. Before this has not happened, the device is not allowed to proceed to Operational state [10]. Using AL Confirmed services, an application can delay the start up process and synchronize their local control loop first. After the local PLL has 'locked in' the device may proceed to the Operational state.
- CoE Slaves with dynamic PDO mapping allow a flexible arrangement of process data. The master configures the layout of the process data which the slave has to transmit during cyclic operation. Therefore CoE Slaves often delay the transition to the state Safe-Operational and set up copy lists before eventually proceeding to the requested state. This approach allows the slaves just to process the copy lists in cyclic operation, regardless to the configured mapping, which is very fast.

When using LFW or SHM API, the AL control changed service is based upon a packet mechanism. For registering the service use [Register for AL control changed indications service](#). To unregister use Unregister from AL control changed indications service. After registering for AL control changed service, the stack informs an application via AL control changed indication packet each time when a master has requested a state change of the ESM via AL control register (0x0120). The stack will remain in the current state until the application triggers a state change via a Set AL status request. This enables an application to delay or even interrupt a state change. Furthermore it can signalize errors to the master using AL status Codes (see the EcsV4_Public.h file, or [10]).

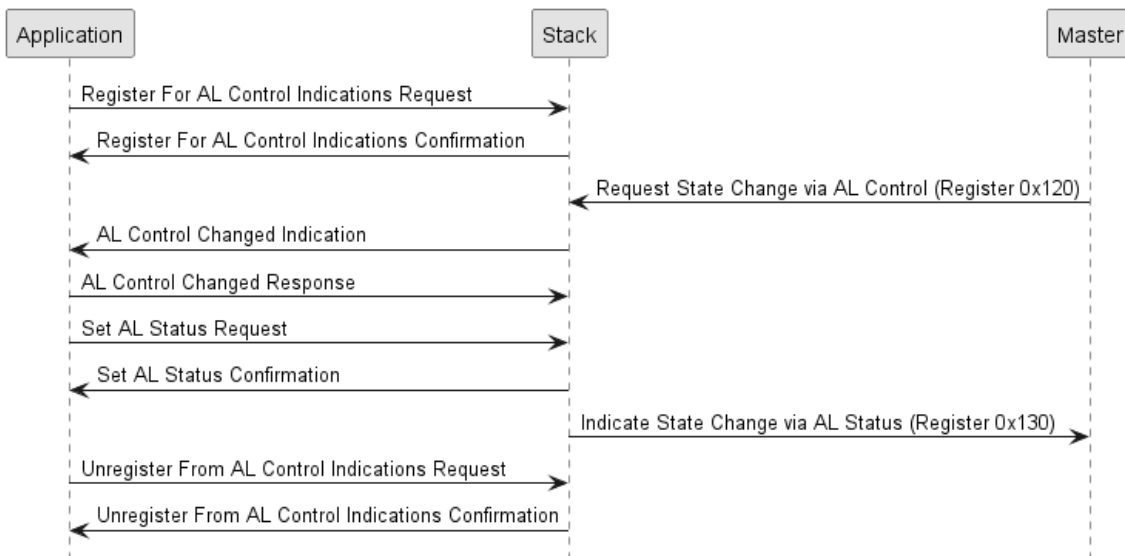


Figure 7. Sequence diagram of EtherCAT state change controlled by application/host

NOTE | There will no indications be sent when switching downwards, for instance when switching from Operational down to Init state.

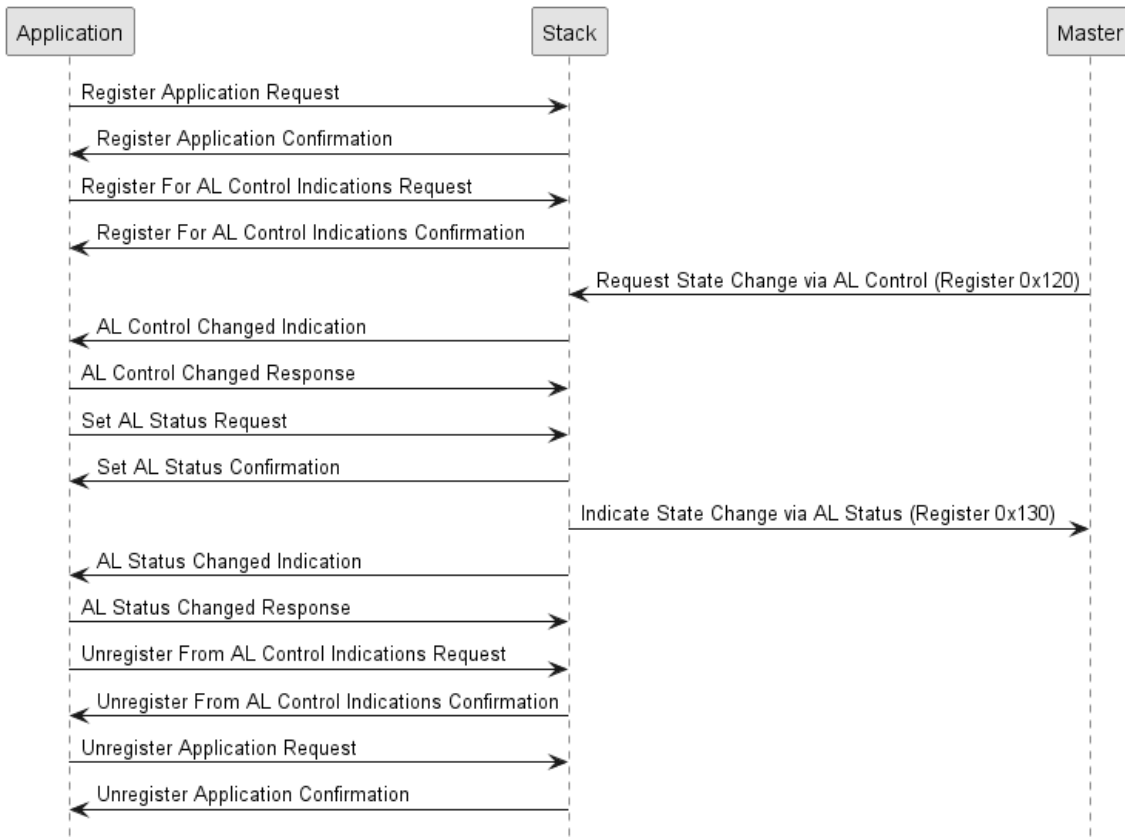


Figure 8. Sequence diagram of state change controlled by application/host with additional AL status changed indications

3.2.2 Slave Information Interface (SII)

As mandatory element, each EtherCAT slave has a slave information interface (SII) [15] which is accessible by the slave. Physically, this is a special storage area for slave-specific data in an EEPROM memory chip. Its size is variable in the range of 1 kBits – 512 kBits (128 – 65536 Bytes). For loadable firmware, the size of the SII is limited to 64 K. Because there is no separate EEPROM in the netX chips, the SII is virtually created in the netX.

After configuration, the firmware writes the content of the SII to the virtual EEPROM. In case the user wants to change the SII data, SII related functions are available as described in section [Slave Information Interface \(SII in virtual EEPROM\)](#). The SII has to be written on every startup of the device and should not overwrite the area before address 0x80 of the SII, which is written by the stack, using the information it got from the configuration.

Structure

The SII can be considered as a collection of persistently stored objects. For instance, these objects may be: * configuration data * device identity * application information data

Masters access the Slaves' SII in order to obtain slave-specific information for instance for administrative and configuration purposes.

The Hilscher EtherCAT Slave Stack provides following packets for Slave Information Interface (SII) interaction:

- SII read service
- SII write service
- Register for SII write Indications service
- Unregister from SII write indications service
- SII write Indication service

The contents stored in the SII can be divided into the following separate groups of parameters:

Address Range	Value/Description
0x0000 - 0x0007	EtherCAT Slave Controller configuration area
0x0008 - 0x000F	Device identity (corresponds to CoE object 1018h)
0x0010 - 0x0013	Delay configuration

Address Range	Value/Description
0x0014 - 0x0017	Configuration data for the Bootstrap Mailbox
0x0018 - 0x001B	Configuration data for the Standard Send/Receive Mailbox
0x001C - 0x003F	Other settings
> 0x003F	Optionally additional information may be present

Table 5. Slave Information Interface structure as defined in IEC 61158, part 6-12

NOTE | The addresses mentioned in the table above relate to 16 bit words.

The optional additional information area (addresses > 0x003F) is organized by different categories. There are standard categories and vendor-specific categories allowed.

The following standard categories are available:

Category	Description	Category Type	Supported by the Hilscher EtherCAT Protocol Stack	Is generated at 'Set Configuration'
NOP	No info	0	Yes	No
STRINGS	String repository for other Categories structure	10	Yes	Yes
Data types	Data Types (reserved for future use)	20	No	No
General	General information structure	30	Yes	Yes
FMMU	FMMUs to be used structure	40	Yes	Yes
SyncM	Sync Manager Configuration structure	41	Yes	Yes
TXPDO	TxPDO description structure	50	Yes	No
RXPDO	RxPDO description structure	51	Yes	No
PDO Entry	PDO Entry description structure	-	Yes	No

Table 6. Available standard categories

All categories have a header containing among others the length information of the rest of the data of the category. Unknown categories may be skipped during evaluation. In general, each of these categories categories is structured as follows:

Parameter	Address	Data Type	Value/Description
1st Category Header	0x40	UNSIGNED15	Category Type
	0x40	UNSIGNED1	Reserved for vendor-specific purposes
	0x41	UNSIGNED16	Length String1
1st Category data	0x42	Category dependent	String1 Data
2nd Category Header	0x40 + x	UNSIGNED15	Category Type
	0x40	UNSIGNED1	Reserved for vendor-specific purposes
	0x41	UNSIGNED16	Length String2
2nd Category data	0x42	Category dependent	String2 Data
...			

Table 7. Slave Information Interface Categories

Hilscher does not define any additional vendor-specific categories of its own. More details about the SII structure can be obtained from or [15] which summarizes the most important Information. Other references are the standard document IEC 61158, part 6-12, section 5.4 "SII coding" in [10], section "Application layer service definition" contains additional information [9]

3.2.3 MBX task

Purpose

On the first hand, the ECAT_MBX task handles all mailbox messages sent by the master and sends them further to the registered queues according to the type they specified to receive. The respective parts of the EtherCAT stack e.g. CoE or

FoE hook to this task to perform their services. On the other hand, the ECAT_MBX task handles all mailbox messages to be sent to the master. Additionally, its state is controlled by the ESM task according to the requested state changes. The respective parts of the EtherCAT stack e.g. CoE or FoE hook to this task to perform their services.

The ECAT_MBX task provides the basis for application level protocols such as:

- CoE (CANopen over EtherCAT)
- FoE (File transfer over EtherCAT)
- SoE (Servodrive over EtherCAT)

3.3 CoE component

The main topics described in this chapter are:

- CoE task
- SDO task
- Object Dictionary V3

Purpose

CoE (CANopen over EtherCAT) can be used for two purposes

1. It can be used for acyclic communication, which is mainly applied for accessing and configuring service data such as communication parameters or device-specific parameters. These service data are stored as service data objects (SDO) within an object dictionary (OD). The EtherCAT Slave protocol stack V4 from Hilscher uses the Object Dictionary V3, which is described in [5]
2. It can be used to provide an easy migration path from CANopen to EtherCAT. CoE emulates a CAN-based environment working on EtherCAT and allows the use of CAN profiles.

In detail, the CoE functionality allows:

- SDO download: Acyclic data transfer from the master to a slave
- SDO upload: Acyclic data transfer from a slave to the master
- SDO information service: reading SDO object properties (object dictionary) from a slave
- CoE emergency Requests

The host can initialize uploads, downloads and information services. Emergencies are generated by slaves. The master collects them and shows them via the slave diagnosis. Also cyclic communication is affected from CoE, as the communication parameters related to PDOs can be configured via SDO to specific object dictionary entries. For more information see [PDO mapping for cyclic communication](#). For details see [9] and [10]

3.3.1 CoE task

The ECAT_COE task is the main handler of all CoE related mailbox messages and routes them to the tasks associated with those inside the CoE component. In addition, the ECAT_COE task provides a mechanism for sending CoE emergency messages.

3.3.1.1 CoE Emergencies

CoE emergencies are sent from the slaves to the master when abnormal states or conditions occur. A CoE emergency message contains a standard CANopen emergency frame consisting of

- Error code (2 bytes)
- Error register (1 byte)
- Data (5 bytes)

Additional data may be added to the CoE emergency message. The master collects the CoE emergencies and stores up to five emergencies per slave. If further emergencies occur, they are dropped. The existence of at least one emergency is represented in the slave diagnosis of the master. The host can read out these emergencies. The host decides whether it deletes the emergencies or they remain in the master.

- See section [Send CoE emergency service](#) for more information about the CoE emergency Service.
- See section [CoE emergency codes](#) for a list of CoE emergency codes and their meanings.

3.3.2 SDO task

The SDO task does not have any packets for the host application to communicate with. The complete packet interface for the SDO functionality of the EtherCAT Slave protocol stack V4 is provided by ODV3 and described in an own separate manual [4].

3.3.3 ODV3 task

This task acts as a connection to the object dictionary V3 described in [4]. It basically provides the following functionality:

- Basic services for reading and writing objects
- Information services for retrieving object-related information
- Management services for creating, maintaining and deleting objects

The following topics also need to be taken into account:

- Access rights
- Complete Access
- CoE communication area for EtherCAT
- Custom object dictionary based on minimal object directory
- Description of objects of minimal object dictionary

3.3.3.1 Access rights

Access rights that apply for the EtherCAT Slave protocol stack V4 are listed in [4]. Additionally, the following additional combinations have been defined:

```

ECAT_OD_READ_INIT = (0x4000) /*internal, not in ethercat spec*/
ECAT_OD_WRITE_INIT = (0x8000) /*internal, not in ethercat spec*/
ECAT_OD_READ_ALL = (ECAT_OD_READ_PREOP|ECAT_OD_READ_SAFEOP|ECAT_OD_READ_OPERATIONAL|ECAT_OD_READ_INIT)
ECAT_OD_WRITE_ALL = (ECAT_OD_WRITE_PREOP|ECAT_OD_WRITE_SAFEOP|ECAT_OD_WRITE_OPERATIONAL|ECAT_OD_WRITE_INIT)
ECAT_OD_ECAT_ALL =
(ECAT_OD_SETTINGS|ECAT_OD_BACKUP|ECAT_OD_MAPPABLE_IN_TXPDO|ECAT_OD_MAPPABLE_IN_RXPDO|ECAT_OD_READ_PREOP|ECA
T_OD_READ_SAFEOP|ECAT_OD_READ_OPERATIONAL|ECAT_OD_WRITE_PREOP|ECAT_OD_WRITE_SAFEOP|ECAT_OD_WRITE_OPERATIONAL|
ECAT_OD_MAPPABLE_IN_SAFE_INPUTS|ECAT_OD_MAPPABLE_IN_SAFE_OUTPUTS|ECAT_OD_MAPPABLE_IN_SAFETY_PARASET)
ECAT_OD_ACCESS_ALL = (ECAT_OD_READ_ALL | ECAT_OD_WRITE_ALL)
    
```

(The vertical bar | stands for logical OR)

3.3.3.2 CoE communication area for EtherCAT

Index (hex)	Object code	Name	Data type	M/O/C
1000	VAR	Device Type	UNSIGNED32	M
1001	VAR	Error Register	UNSIGNED8	O
1008	VAR	Manufacturer Device Name	VisibleString	O
1009	VAR	Manufacturer Hardware Version	VisibleSting	O
100A	VAR	Manufacturer Software Version	VisibleSting	O
1018	RECORD	Identity Object	Identity(0x23)	M
1600	RECORD	1st receive PDO Mapping	PDO Mapping (0x21)	C
1601	RECORD	2st receive PDO Mapping	PDO Mapping (0x21)	C
...

Table 8. Abstract of the CoE Communication Area (0x1000 - 0x1FFF)

See [10] for the complete list.

3.3.3.3 Custom object dictionary based on minimal object directory

Using the `ECAT_SET_CONFIG_COEFLAGS_USE_CUSTOM_OD` configuration flag, the user can enable or disable working with a custom object dictionary. If this configuration flag is not set a default object dictionary will be created by the stack. If this configuration flag is set only a minimal object dictionary will be created. The following contains a list of the objects contained in this minimal object dictionary. Note that without providing additional objects by a user

application an EtherCAT master will not be able to bring the slave to Operational state.

The stack always creates the following, regardless of using a custom object dictionary or not.

Index (hex)	Subindex	Object	Comment
1000	00	Device Type	
1018	00	Identity Object	Fix value set to 4
1018	01	Vendor ID	
1018	02	Product code	
1018	03	Revision number	
1018	04	Serial number	

Table 9. Minimal object dictionary

Definitions of the object 0x1000 and 0x1018 can be found in [10]

NOTE | Each object to be used for process data transfer has to be byte aligned.

3.3.3.4 Default object dictionary

The stack creates a default object dictionary if the configuration flag `ECAT_SET_CONFIG_COEFLAGS_USE_CUSTOM_OD` is set to zero.

The default object dictionary contains all objects that are necessary to bring the slave to Operational state. The online objects which are created by the stack match with the objects described in the ESI file. The RxPDO and TxPDO objects described in the ESI file are the maximum possible PDO's which can be transferred/created. After the configuration was sent to the stack, it creates a set of process data objects according to the configured process data length. (This set is a subset of the process data objects in the ESI file.) If the process data size is changed by a Set IO Size command, the present objects are deleted and a new set of objects is created. For every byte of process data, a single subobject is created in the OD. The order is not changeable and process data is always copied in this order.

The object dictionary created in this way is not sufficient for every user. To serve special needs it is possible to create a custom object dictionary see [Custom object dictionary based on minimal object directory](#). A custom object dictionary is necessary if PDO objects with different sizes to 1 byte are required. The default objects cannot be changed and additional processdata objects can not be added to the default object dictionary. If only additional SDO objects are needed, they can be added to the default objects created by the stack. For this case we recommend using the object range 0x4000 to 0x5FFF in order to avoid conflicts with process data objects.

The following table shows the objects created by the default OD.

Index (hex)	Subindex	Object	Comment
1000	00	Device Type	
100A	00	Manufacturer Software Version	
1018	00	Identity Object	Fix value set to 4
1018	01	Vendor ID	
1018	02	Product code	
1018	03	Revision number	
1018	04	Serial number	
1600	00	RxPDO	Number of mapped process data objects, value range 1...200 (not present if output data is zero) Direction: master → slave
1601	00	RxPDO	Number of mapped process data objects, value range 1...200 for data bytes 200 – 399 (not present if output data ≤ 200 bytes)
160x
1A00	00	TxPDO	Number of mapped process data objects, value range 1...200 (not present if output data is zero) Direction: slave → master

Index (hex)	Subindex	Object	Comment
1A01	00	TxPDO	Number of mapped process data objects, value range 1...200 for data bytes 200 – 399 (not present if output data \leq 200 bytes)
1A0x
1C00	00	Sync Manager communication types	Number of elements (max 8 sync managers are defined in default OD)
1C00	01	Sync Manager 0	Value: 0x01
1C00	02	Sync Manager 1	Value: 0x01
...
1C10	00	Sync Manager 0 PDO assignment	0 because Receive Mailbox
1C11	00	Sync Manager 1 PDO assignment	0 because Transmit Mailbox
1C12	00	Sync Manager 2 PDO assignment	Number of assigned mapping objects (not present if output data is zero) Direction: master \rightarrow slave
1C12	01	Subindex 001	0x1600 (not present if output data = zero)
1C12	02	Subindex 002	0x1601 (only present if output data exceeds 200 byte)
...
1C13	00	Sync Manager 3 PDO assignment	Number of assigned mapping objects (not present if input data is zero) Direction: slave \rightarrow master
1C13	01	Subindex 001	0x1A00 (not present if input data = zero)
1C13	02	Subindex 002	0x1A01 (only present if input data exceeds 200 byte)
...
2000	00	Outputs	Number of elements, value 0..200 (only present if output data configured)
2000	01	1 Byte Out (0)	Acyclically read value of this process data byte
...
3000	00	Inputs	Number of elements, value 0..200 (only present if input data configured)
3000	01	1 Byte In (0)	Acyclically read value of this process data byte
...

Table 10. Default object dictionary

3.3.3.5 PDO mapping for cyclic communication

The process data objects (PDOs) provide the interface to the application objects. They are assigned to the entries in the object dictionary. The process of assignment is denominated as PDO mapping and is practically accomplished via a specific mapping structure in the object dictionary (for EtherCAT: Sync Manager PDO assignment (Objects 0x1C10 – 0x1C2F)).

This mapping structure can be found at: * 0x1600-0x17FF (0x1600 for the first RxPDO) * 0x1A00-0x1BFF (0x1A00 for the first TxPDO)

The following figure explains the relationship between object dictionary (left upper part), PDO mapping structure (right upper part) and the resulting PDO containing the application objects to be mapped (lower part).

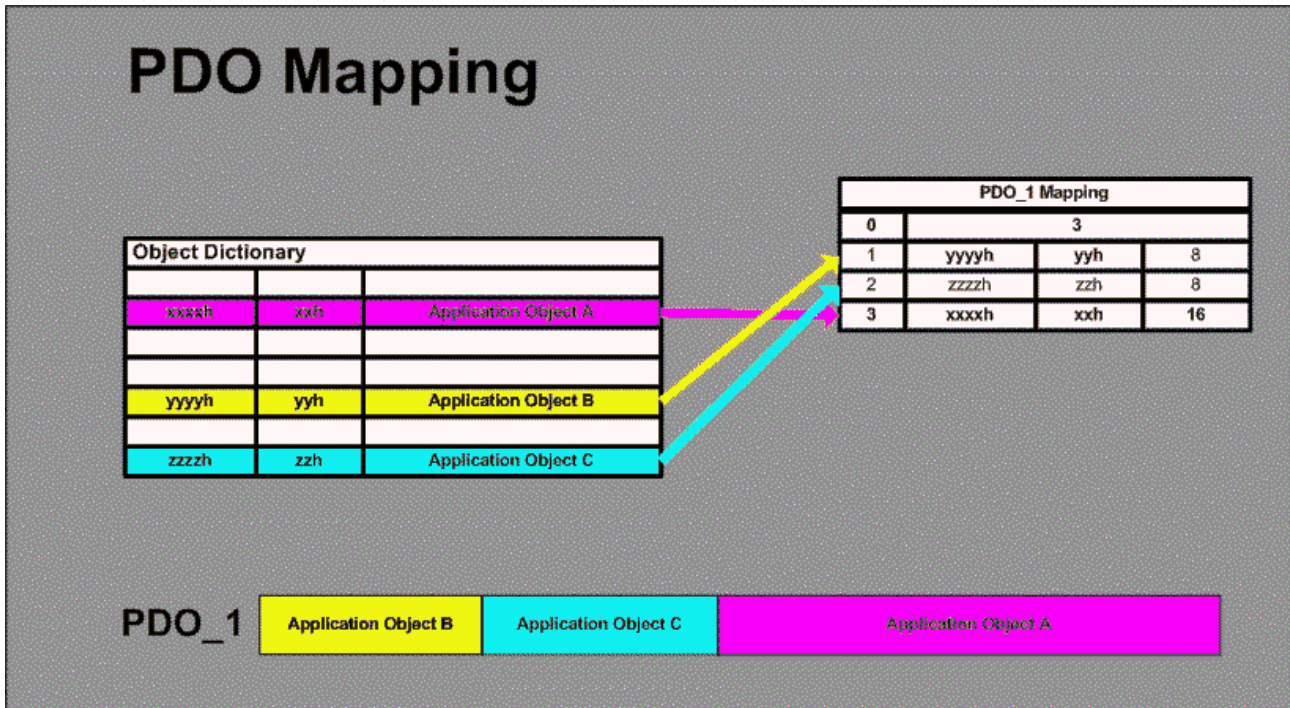


Figure 9. Mapping scheme for a PDO

One PDO mapping entry requires 32 bit. It consists of: * 16 bit containing the index of the object dictionary entry containing the application object to be mapped * 8 bit containing the subindex of the object dictionary entry containing the application object to be mapped * 8 bit containing the length information (in Bit).

The use of a mapping structure must be configured. In EtherCAT, this is done by the Sync Manager PDO assignment (Objects 0x1C10 – 0x1C2F). Usually 0x1C12 contains the RxPDO's (e.g. 0x1600) and 0x1C13 the TxPDO's (0x1A00)

Sometimes it is necessary to change the mapping after startup of the device (e.g. for modular devices). For information on this topic refer to [Section 5.5](#).

3.3.3.6 Complete Access

The SDO Complete Access mechanism allows to read out a whole object with all subobjects at once. The ODV3 task translates those accesses, so they appear as single accesses to the application side and no special handling is required. A common use case is to handle the download of startupparameters for dynamic process data. For information on this topic refer to [One application registered Complete Access: \(application successful\)](#).

3.4 FoE component

The EtherCAT standard defines various mailbox protocols. One of these protocols is File Access over EtherCAT (FoE). This protocol is similar to the well-known Trivial File Transfer Protocol (TFTP). By the help of FoE it is possible to exchange files between an EtherCAT master and an EtherCAT slave device. When downloading a file via FoE to a Hilscher EtherCAT slave the file will be copied to the local file system of the device. For a slave supporting FoE this support has to be indicated in ESI and SII. FoE can be used in any state the mailbox is activated, these are all states except Init state.

FoE can be used to read files or store files physically in the filesystem (only system volume is possible) or with the virtual file option, which means that the application holds the data and handles the read, write operations (no data on filesystem). The virtual option can be a solution if the file does not fit in the filesystem. Names for files which use the filesystem are restricted to the 8.3 convention, virtual filenames can be longer.

A very important use case for FoE is a firmware download to an EtherCAT slave in order to update the used slave firmware.

A firmware download/update needs following steps

- If not done: Download of EtherCAT slave firmware (not by FoE)
- Slave stack configuration according to ESI (FoE set in ESI)
- Slave shall be connected to EtherCAT master

- Slave shall be set at least to EtherCAT state Pre-Operational
- Firmware file (*.nxf) shall be downloaded by FoE from master to slave
- Slave stack will check file
 - OK: firmware will be used after next power cycle
 - error: Error “Illegal” (see ETG1000.6, Table 92 – Error codes of FoE) will be reported by slave to master

3.5 Behavior when receiving a Set Configuration command

The following rules apply for the behavior of the EtherCAT Slave protocol stack when receiving a set configuration command:

- The configuration packets name is
 - *ECAT_SET_CONFIG_REQ* for the request and
 - *ECAT_SET_CONFIG_CNF* for the confirmation.
- The configuration data are checked for consistency and integrity.
- In case of failure all data are rejected with a negative confirmation packet being sent.
- In case of success the configuration parameters are stored internally (within the RAM).
- The parameterized data will be activated only after a channel initialization has been performed.
- No automatic registration of the application at the stack happens.
- The confirmation packet *ECAT_SET_CONFIG_CNF* only transfers simple status information, but does not repeat the whole parameter set.

If you allowed the automatic start of the communication (can be chosen within the Set Configuration Request packet) the device will allow to advance the ESM state beyond Pre-Operational state. Otherwise, setting of the BusOn bit via ApplicationCOS (see [1]) is required.

If a watchdog error occurs ([Watchdogs](#)) prior to setting the BusOn bit via ApplicationCOS, this will prohibit advancing to ESM states beyond Pre-Operational. You can recognize this situation by the unusual characteristic signal of the LEDs and an [AL control changed indication packet](#) with indicated EtherCAT states Init or Pre-Operational being sent to the host. In this case a channel reset is required. If you intend to use the DPM interface, also refer to the related DPM manual [1].

3.6 Watchdogs

Three watchdogs exist in the context of the EtherCAT Slave stack.

- The DPM Watchdog monitors the communication between the host application and the stack via the dual-port memory.
- The SM Watchdog monitors the process data received from the EtherCAT network.
- The PDI Watchdog allows the master to monitor whether the EtherCAT Slave is still running

3.6.1 DPM watchdog

The DPM Watchdog is relevant for LFW users only. The application and the EtherCAT Slave uses two watchdog cells in the dual-port-memory for each communication channel (details see [1]). The watchdog time is configured with the basic configuration parameters of the [Set configuration service](#). If the DPM Watchdog expires, the EtherCAT Slave will return to the Pre-Operational state. The stack notifies the master with the AL status Code:

```
#define ECAT_AL_STATUS_CODE_DPM_HOST_WATCHDOG_TRIGGERED 0x8002
```

The application has to use Channel Init service, because the EtherCAT Slave requires a Channel Init to leaves this state. For a list of available AL status Codes, see in `EcsV4_Public.h` file.

3.6.2 SM Watchdog

The EtherCAT Slave uses the SyncManager Watchdog to monitor the receiving of process data. This watchdog is only related to output data direction mapped in RxPDO's, meaning data send from the master to the device, usually transferred by Syncmanager 2. The default value of the SM watchdog is set to 100 ms, value 0 deactivates the watchdog.

In case the EtherCAT Slave has input data only, the watchdog will not be started at all. If the EtherCAT Slave receives no process data within the configured time, the AL status Code *ECAT_AL_STATUS_CODE_SYNC_MANAGER_WATCHDOG* is set and the slave falls back to the SafeOP state. The AP

task starts to trigger the SM Watchdog with the first reading of the buffer and triggers again with each next reading. It is reset with every PDO cycle from the master. The EtherCAT Master or a configuration tool can change the watchdog time by writing the related registers (0x420, 0x400 = divider for both wdgs.) in the EtherCAT Slave. You can add an entry (OEM customization) in the ESI file, see Reg0420 in [13]. The typical events causing a watchdog timeout are unplugging the network cable or long cycle times.

3.6.3 PDI Watchdog

The PDI watchdog has a default value of 100ms, value 0 deactivates this watchdog. The EtherCAT Master or a configuration tool can configure the related registers by writing the related registers (0x410, 0x400 = divider for both wdgs, 0x110 = status) in the EtherCAT Slave.

3.7 Usage of PHYs

The usage of PHYs is as follows:

- PHY 0 means in-port.
- PHY 1 means out-port.

Chapter 4 Status information

The EtherCAT Slave provides status information in the dual-port memory. The status information has a common block (protocol-independent) and an EtherCAT Slave specific block (extended status).

4.1 Common status

For a description of the common status block, see reference [1].

4.2 Extended status

Offset	Type	Name	Description
0x0050	UINT32	ulNextSync0Time	32-bit time value of the next Sync0 event. The value is updated every Sync0 interrupt if enabled (see section Sync PDI configuration parameter).
0x0054	UINT64	ulSMCycleTimeNanoSec	64-bit time value measured between two SM2 interrupts. The value is updated every SM2 interrupt

Table 11. Extended status block



Chapter 5 Requirements to the application

5.1 Sequence within the host application

The next figure shows the sequence within a host application program.

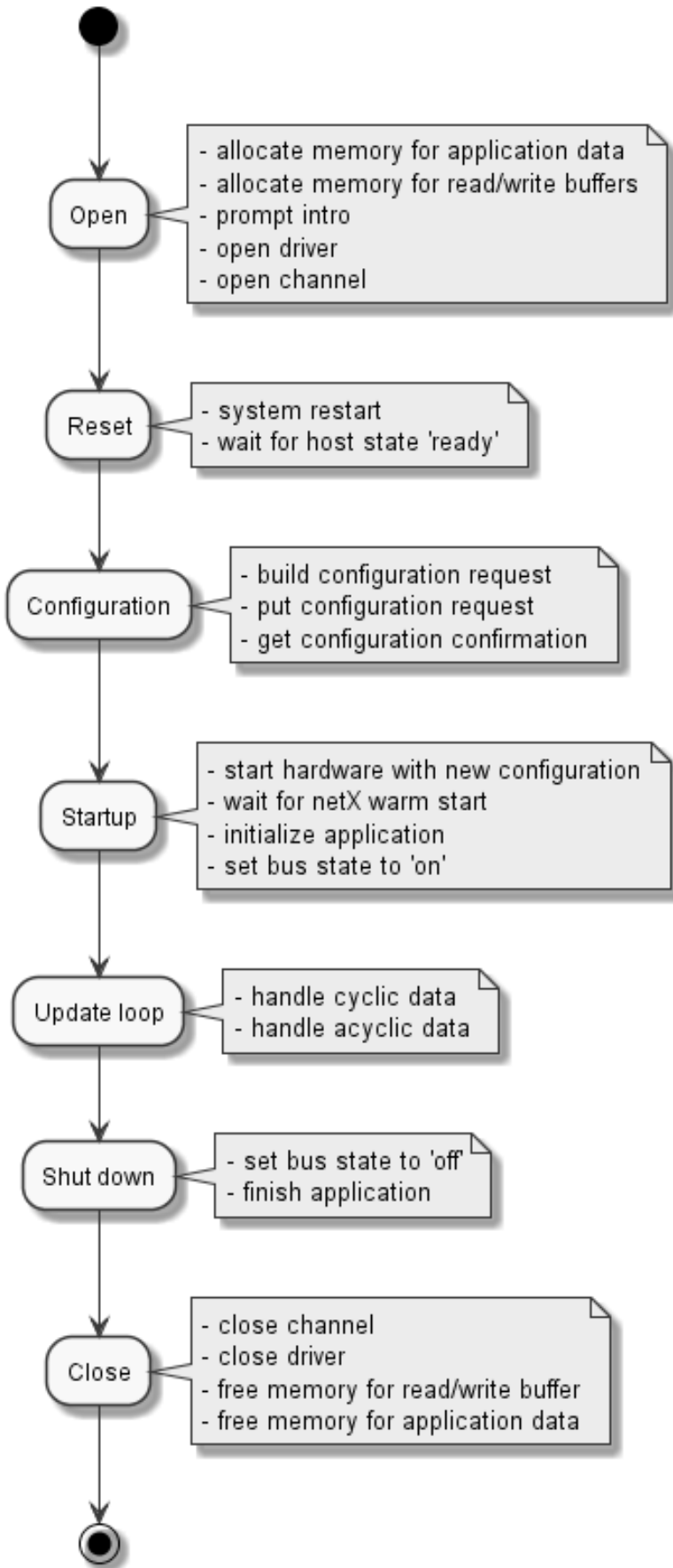


Figure 10. Sequence within the application

5.2 General initialization sequence

The next figure explains the general initialization sequence. Note, that all needed registration requests are done **after the channel initialization** and **before the bus is switched on**. The only exception is the register application request, which precedes the configuration and channel initialization.

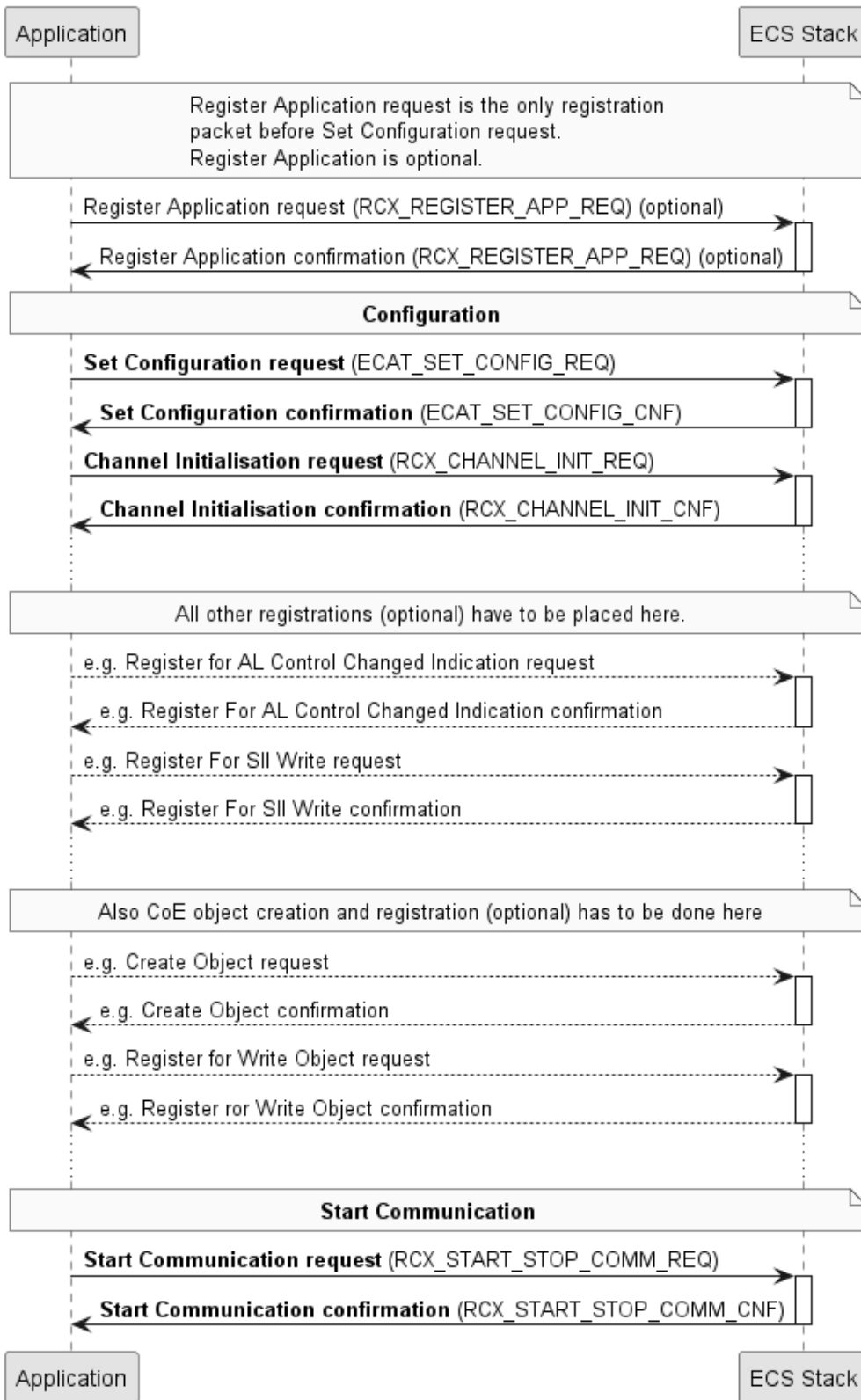


Figure 11. Initialization sequence with placing of registrations and object dictionary creation

Before the communication is started, the application has to read the IO Channel. This is necessary to be able to reach the Op state within the EtherCAT protocol stack:

Before the stack accepts switching to Operational state (Op), it needs to have access to the cyclic process data on the bus, which has been sent from the EtherCAT master to the EtherCAT slave. Therefore, the application needs to read the

data at minimum once after the master has started to send it (e.g. with XChannellIORead). The master starts sending process data at the state change from Preop2 to SafeOP. This behavior is not present for input only devices (which only transfers data from device to bus) or devices that have not configured any output data.

5.3 Explicit Device Identification

This section describes the correct handling for the usage of an Explicit Device Identification (optional)

5.3.1 Initialization sequence

The next figure shows the initialization sequence diagram. Prerequisite for correct operation is a power on or system start. The PHYs will be disabled after power on or system start.

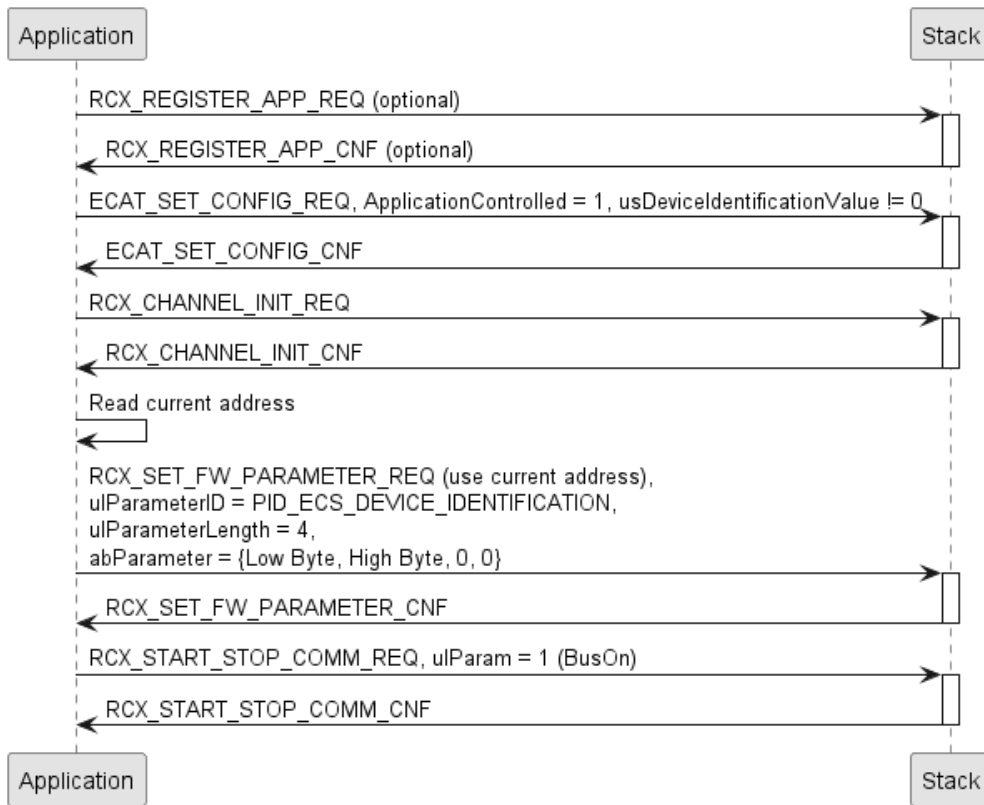


Figure 12. Initialization sequence for Explicit Device Identification

Remarks: *RCX_START_STOP_COMM_REQ* can be replaced with BusOn via CommCOS *RCX_CHANNEL_INIT_REQ* can be replaced with Channellnit via CommCOS.

The application has to set the Device Identification Value **before** the BusOn is to be executed. The Device Identification Value is handled according to the Explicit Device Identification via ESC registers ALSTATUS / ALSTATUSCODE. For details on the functionality of those registers within the stack, see [11]

Details concerning the Device Identification Value from Figure 12 are as follows.

The address value which is set on the slave via ID selector (e.g. an address from a rotary switch or a display) is read out by the master via the so called Requesting ID mechanism. This mechanism needs to be activated in the slave and also needs to be inactive if the feature is not used. The address value of the switch can be sent to the slave by using the *RCX_SET_FW_PARAMETER_REQ_T* packet:

- The address switch has to be polled by the user application in order to get the address.
- Setting a value unequal to zero with the parameter *usDeviceIdentificationValue* *ECAT_SET_CONFIG_UID* in the SetConfiguration request activates the address handling by the stack (answering if the master requests the address). The value zero deactivates the handling.
- After the address handling is activated, the address switch has to be polled and the actual value has to be given to the stack to get the correct information before the slave starts to communicate over the network. Therefore, the command *RCX_SET_FW_PARAMETER_REQ* has to be sent before *BUS_ON*. The stack writes the address in register 134. This mechanism makes sure that the address is set after every cold start of the device.

- Additionally, it is necessary to poll the switch frequently while the device is running and send the Command *RCX_SET_FW_PARAMETER_REQ* to the stack, when the address has changed. (This is optional since conformance test version 7000.2 V1.2.6) The stack will not provide the new address to the master until the stack requests it again.
- If the address handling is switched off, because the device does not support it, the address should not be sent by *RCX_SET_FW_PARAMETER_REQ*, because this request with a value unequal to zero also activates the address handling. (Also, be aware that there is no entry *<IdentificationReg134>* in the ESI file if the address is not supported.)
- Setting the address value with the parameter *usDeviceIdentificationValue* (SetConfiguration request) only without polling is also possible and can make sense e.g. for mechanical engineering, but it is no longer sufficient for fulfilling the conformance requirements.
- Beginning with protocol stack version 4.9, the following legacy behavior according to ETG1020 is supported concerning simultaneous handling of rotary switch and Configured Station Alias: Only in case both values are set before Bus on, the rotary switch value is copied to register 12 of the ESC and Configured Station Alias is set to 0. An emergency error is generated. After going back to state Init, the slave can be brought to state Op again. (See [11] or ETG's knowledgebase website, which provides details).
- Legacy behavior in handling of explicit device IDs according to ETG1020 only applies for cases where both device ID values are set - the one originating from bus side and the local one. This means: if the rotary switch value is set on device startup to a value not 0, it obtains priority over the value originating from bus side. However, this only applies if set on first startup (Figure 12 is obeyed) Please note the difference between the terms Configured Station Alias (Section 6.2.5) and Explicit Device ID (Section 5.3):
- Configured Station Alias or Configured Station Address is a 16 Bit value designating a station and can be used by the master (when activated) to send datagrams to a slave as done with the first station (Node) Address. It is set from master side and stored in the EEPROM.
- An Explicit Device ID is a value which is used for identification purpose and never used by the master as address value for sending datagrams. It can be set in two ways. For instance by means of a rotary switch locally on the device. Or by setting the Configured Station Alias via bus (means storing it in EEPROM) and doing a powercycle on the device. After power on the device copies the value to register 0x12 of the ESC. This register can also be used for Explicit Device Identification purpose. So Explicit Device Identification Value is an overall term and can alternatively use one of the mechanisms.

NOTE | NetX52 based devices need some additional handling on the application to support the legacy mechanism. Please ask our support for an Application Note.

5.3.2 Set firmware parameter

With this parameter the packet is used to send an address value to the stack. Send each time the value changes.

Packet parameters

Value	Name	Description
0x30009001	PID_ECS_DEVICE_IDENTIFICATION	ulParameterID switch or display address value

Table 12. Set device identification value

ulParameterLength

ulParameterLength = 4.

abParameter

Field	Meaning
abParameter[0]	Low Byte of Device Identification Value
abParameter[1]	High Byte of Device Identification Value
abParameter[2]	set to zero
abParameter[3]	set to zero

Table 13. abParameter

Packet description

Structure <i>RCX_SET_FW_PARAMETER_REQ_T</i>				Type: Request
Variable	Type	Value/Range	Description	
structure HIL_PACKET_HEADER_T				
ulDest	UINT32		Destination queue-Handle	
ulSrc	UINT32		Source Queue-Handle	
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet	
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process	
ulLen	UINT32	12	Packet data length in bytes	
ulId	UINT32	0 ... $2^{32} - 1$	Packet Identification as unique number generated by the Source Process of the Packet	
ulSta	UINT32	0	Status code of the packet	
ulCmd	UINT32	0x2F86	RCX_SET_FW_PARAMETER_REQ - Command	
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons	
ulRout	UINT32	x	Routing, do not touch	
Structure <i>RCX_SET_FW_PARAMETER_REQ_DATA_T</i>				
ulParameterID	UINT32	0x30009001	PID_ECS_DEVICE_IDENTIFICATION	
ulParameterLength	UINT32	4	Length of parameter	
abParameter	UINT8[4]		See description of abParameter	

Table 14. Request packet *RCX_SET_FW_PARAMETER_REQ_T*

Confirmation packet

Packet description

Structure <i>RCX_SET_FW_PARAMETER_CNF_T</i>				Type: Confirmation
Variable	Type	Value/Range	Description	
structure HIL_PACKET_HEADER_T				
ulDest	UINT32		Destination queue-Handle	

Structure <code>RCX_SET_FW_PARAMETER_CNF_T</code>			Type: Confirmation
<code>ulSrc</code>	UINT32		Source Queue-Handle
<code>ulDestId</code>	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
<code>ulSrcId</code>	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
<code>ulLen</code>	UINT32	0	Packet data length in bytes
<code>ulId</code>	UINT32	0 ... $2^{32} - 1$	Packet Identification as unique number generated by the Source Process of the Packet
<code>ulSta</code>	UINT32		See section Status and Error Codes
<code>ulCmd</code>	UINT32	0x2F87	<code>RCX_SET_FW_PARAMETER_CNF</code> - Command
<code>ulExt</code>	UINT32	0	Extension not in use, set to zero for compatibility reasons
<code>ulRout</code>	UINT32	x	Routing, do not touch

 Table 15. Confirmation packet `RCX_SET_FW_PARAMETER_CNF_T`

Example

The following example shows how to set a value as identification value:

```
void FillOutFwParamDeviceIdentPacket(TLR_UINT32 ulSrc, RCX_SET_FW_PARAMETER_REQ_T* ptPkt, TLR_UINT16 usIdentValue)
{
    ptPkt->tHead.ulCmd = RCX_SET_FW_PARAMETER_REQ;
    ptPkt->tHead.ulExt = 0;
    ptPkt->tHead.ulSta = 0;
    ptPkt->tHead.ulSrcId = 0;
    ptPkt->tHead.ulSrc = ulSrc;
    ptPkt->tHead.ulLen = 12;
    ptPkt->tHead.ulRout = 0;
    ptPkt->tHead.ulId = 0;
    ptPkt->tHead.ulDestId = 0;
    ptPkt->tHead.ulDest = 0x20; /* addressed communication channel */
    ptPkt->tData.ulParameterID = PID_ECS_DEVICE_IDENTIFICATION;
    ptPkt->tData.ulParameterLength = 4;
    ptPkt->tData.abParameter[0] = usIdentValue & 0xFF;
    ptPkt->tData.abParameter[1] = usIdentValue >> 8;
    ptPkt->tData.abParameter[2] = 0;
    ptPkt->tData.abParameter[3] = 0;
}
```

5.3.2.1 No implementation for netX devices with rotary switches

For netX devices with natively implemented rotary switches such as the COMX51CA-RE\R, no implementation must be done and also no implementation is allowed. This means: Sending `RCX_SET_FW_PARAMETER_REQ` with `ulParameterID = PID_ECS_DEVICE_IDENTIFICATION` is forbidden!

For Set Configuration Packet:

- The Component `ECAT_SET_CONFIG_UID_T` parameter `usDeviceidentificationValue` is ignored (overwritten) by the firmware.
- The `ECAT_SET_CONFIG_UID_T` parameter `usStationAlias` (which is not allowed for devices that need a certification) is overwritten when rotary switch value is unequal to 0, which means activation. So the `usStationAlias` parameter should not be used if it cannot be ensured that the switches are set to zero (legacy handling applies, see ETG knowledgebase for more Information).

5.3.3 Required entry in ESI file for Explicit Device Identification

Beginning with EtherCAT Slave protocol stack version 4.9 supports the Legacy Mode Mechanism. If the Explicit Device ID is set at the device via an ID selector (not set by the Master), the ESI file has to be adapted to support the Legacy Mode Mechanism. For instance, you have to do this when using the rotary switches or a display for address setting. To adapt the ESI file, add the following entry there:

```

<Info>
  <IdentificationAdo>#x12</IdentificationAdo>
  <IdentificationReg134>1</IdentificationReg134>
</Info>

```

5.4 Complete Access for object data held by application

This section describes the sequence of packets for the use case "object dictionary holds object and subobjects descriptions, application holds data" for a Complete Access. For a description of the use case, see reference [4]. This section is not for the use case dynamic PDO mapping, therefore see Section 5.5. The ODV3 task translates SDO up- and downloads with Complete Access from an EtherCAT Master into single accesses to the application. The main difference of the Complete Access compared to the standard access is the order when the object validation indications are sent: with Complete Access, the validation indications are implicit. If a Complete Access request from EtherCAT Master fails, the application has to restore the former object values. The following example shows how to handle the packets for a write object request correctly to support both single access and Complete Access.

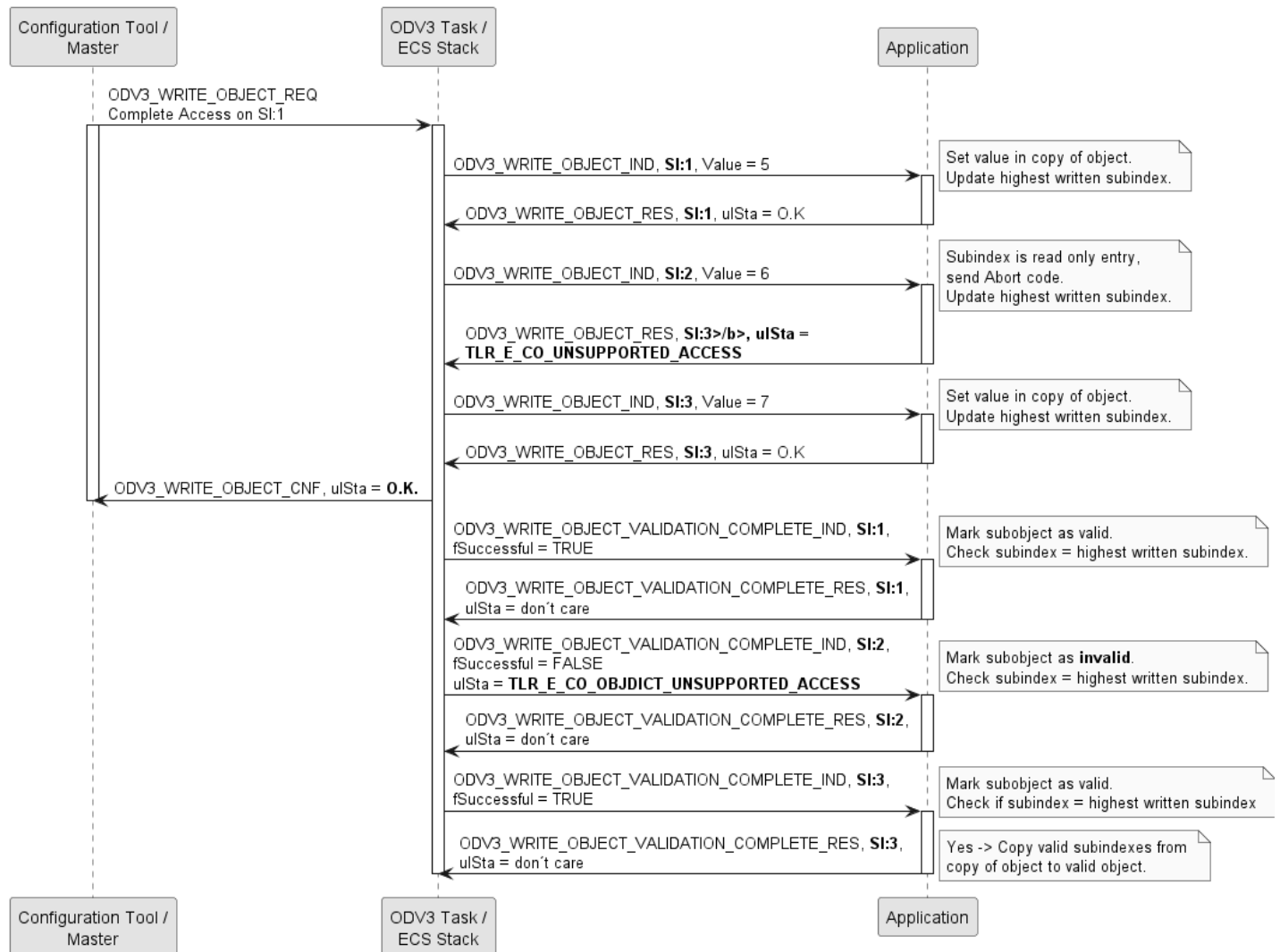


Figure 13. SDO download with Complete Access (successful)

The application has to save new values using a copy of the values in order to be able to restore the former value of the objects in case of an error. After the application has received the validation of the last written subindex, the application has to take over the (new) values of successful answered values only. In the case of an abort of the Complete Access request, the ODV3 task will send all validation indications with `fSuccessful` set to `False`. The example shows that the application answers the write indication on subindex 2 with the error code `TLR_E_CO_OBJDICT_UNSUPPORTED_ACCESS`. This does not lead to an error for the Complete Access download request itself. The value of the subindex just remains unchanged. In the case, all subindexes report an error, the complete access will fail. Also in case subindex zero of an object is read only and the master or configuration tool tries to write more

subindexes as subindex zero of this object contains, causes for example a failure of the Complete Access request.

5.5 Dynamic PDO mapping

Dynamic PDO mapping means that the process data configuration of the EtherCAT Slave device can be changed via EtherCAT. The EtherCAT Master or a configuration tool can use * PDO assignment (Sync Manager objects e.g. 0x1C12/0x1C13), or * PDO configuration (e.g. 0x1600/0x1A00) or both in order to dynamically change the PDO mapping. A common use case for dynamic PDO mapping are modular devices. As a result of the required functionality for the complete device, the user application will be more complex if the user application has to support both PDO assignment and PDO configuration. This section describes how the application has to support the dynamic PDO mapping

functionality and which sequences of packets occur.

The following figures show the handling for PDO assignment. Please observe the sequence of packets and make sure that the Set IO Size request reaches the slave stack before the process data length evaluation takes place. The sequence of writing the assignment objects 0x1C12 and 0x1C13 may differ from configuration tool to configuration tool because the sequence is not defined. For details on the download order, see [14].

For the dynamic PDO mapping, the user application has to response on multiple indications of the *ODV3_WRITE_OBJECT* service from the EtherCAT Slave stack. As soon as the user application receives "Writing subindex 0 with the value of the highest subindex", the end of the dynamic mapping for each PDO is indicated. The user application will receive only one *ODV3_WRITE_OBJECT_IND* if no process data for a direction has been configured.

NOTE | The EtherCAT Slave stack sends the *ODV3_WRITE_OBJECT_VALIDATION_COMPLETE* packet only if the application has registered to get this indication for the particular object or in case Complete Access is used.

For more information about the dynamic PDO mapping, see chapter 10 in [11] and [14]. The following subsections show some usecases.

5.5.1 One application registered (application successful)

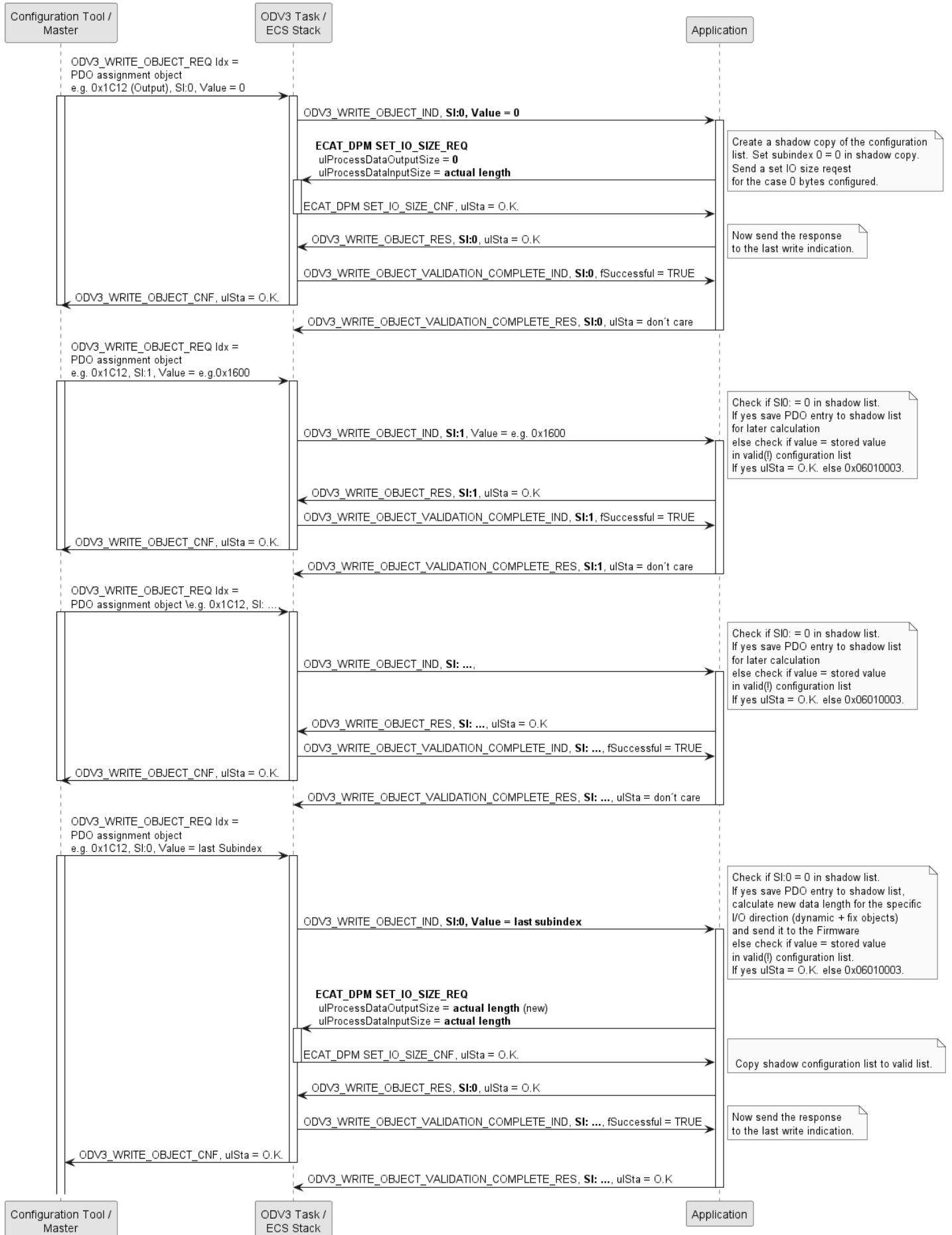


Figure 14. Dynamic PDO assignment: One application registered for write indications (successful)

The user application has to create two lists. One list contains the current configuration and the other list contains a copy (shadow list) of the original configuration in order to save the new configuration. As soon as subindex 0 is set to zero, the user application has to copy the current configuration into the shadow list. Only at the end of correct configuration sequence, the user application can copy the shadow list back to the current configuration list. In case of error which means that at least one 'ODV3_WRITE_OBJECT_CNF with ulSta unequal to 0' occurs, the current configuration list stays valid and the process data length can be restored.

As soon as the application receives 'ODV3_WRITE_OBJECT_IND with subindex 0 has value zero', the application has to send the first *ECAT_DPM_SET_IO_SIZE_REQ* of the sequence to the EtherCAT Slave. The *ECAT_DPM_SET_IO_SIZE_REQ* contains the length of input and the length of output. The length for the current written I/O direction gets value 0, the length for the other direction gets the current value. After the application has received the confirmation *ECAT_DPM_SET_IO_SIZE_CNF*, the application has to send the response *ODV3_WRITE_OBJECT_RES* for subindex 0. The application has to send the *ECAT_DPM_SET_IO_SIZE_REQ* at the beginning, because of the possibility that no more *ODV3_WRITE_OBJECT_REQ* from master follow in case the configured data size is zero. In case, the device has additionally fix configured process data in the specific direction (which is not downloaded by the configuration tool), this first *ECAT_DPM_SET_IO_SIZE_REQ* with value zero should not be send or instead send with the length of the fix process data. The *ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND* packets have no effect because there is no other application registered.

The application receives values for the subindexes within the write indication. If the object is deactivated (subindex 0 is zero which means "writing allowed") the application has to save the new value in the shadow list. If not, the application has to compare the current value with the new value. If they match, this is an allowed request otherwise this is an unallowed access (0x06010003 = Subindex cannot be written, SI0 must be 0 for write access). This behaviour allows a master to check the configuration.

The end of object writing is indicated by 'ODV3_WRITE_OBJECT_IND sets subindex 0 to a value unequal to zero'. The application has to calculate the process data length and send the *ECAT_DPM_SET_IO_SIZE_REQ* to the EtherCAT slave. This request has to contain the length of input and the length of output and one length of both has a new value. As soon as the application has received the *ECAT_DPM_SET_IO_SIZE_CNF*, the application has to send the *ODV3_WRITE_OBJECT_RES* for subindex 0.

Only this sequence ensures that the EtherCAT Slave stack uses the new data size for the next process data evaluation which takes place before changing the operating mode to Safe-Operational.

5.5.2 One application registered (application not successful)

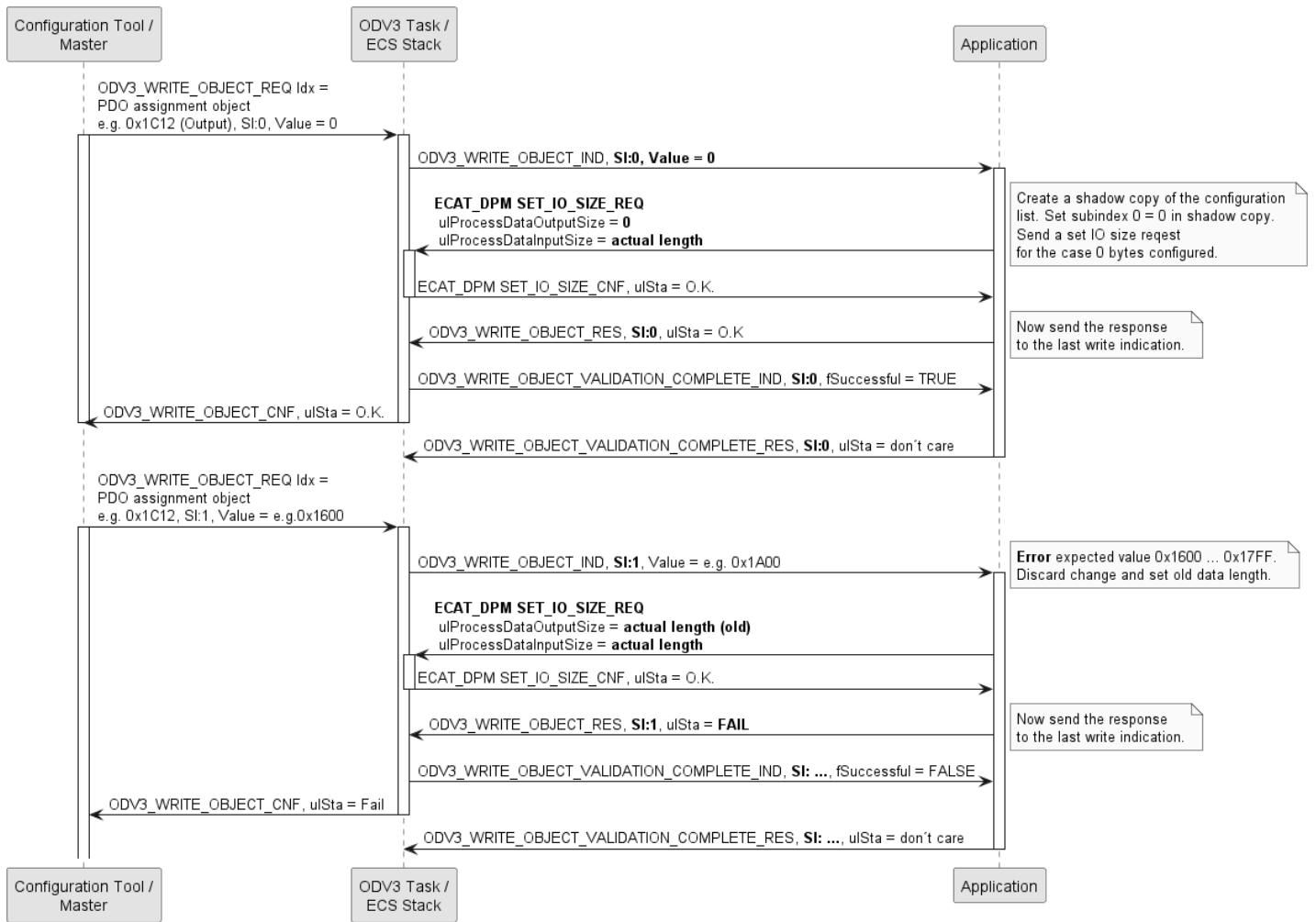


Figure 15. Dynamic PDO assignment: One application registered for write indications (not successful)

Until the first confirmation `ODV3_WRITE_OBJECT_CONF` for subindex 0, the sequence is the same as described in section [Figure 14](#). If one of the following `_ODV3_WRITE_OBJECT_IND` fails, indicated by 'ulSta unequal to zero' from the application, the application has to restore the former I/O size or has to set a default size. An example for this case can be that a written value is out of range.

5.5.3 Multiple applications registered (one application not successful)

In case of multiple applications are registered, the application has to check the `ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND` packets. If at least one other registered application answers on a `ODV3_WRITE_OBJECT_IND` packet with an ulSta unequal to zero, all applications will get the `ODV3_WRITE_OBJECT_VALIDATION_COMPLETE_IND` packet with the respected error code. In this case, the application has to restore the former I/O size or has to set a default size as [Figure 15](#) shows.

5.5.4 One application registered Complete Access: (application successful)

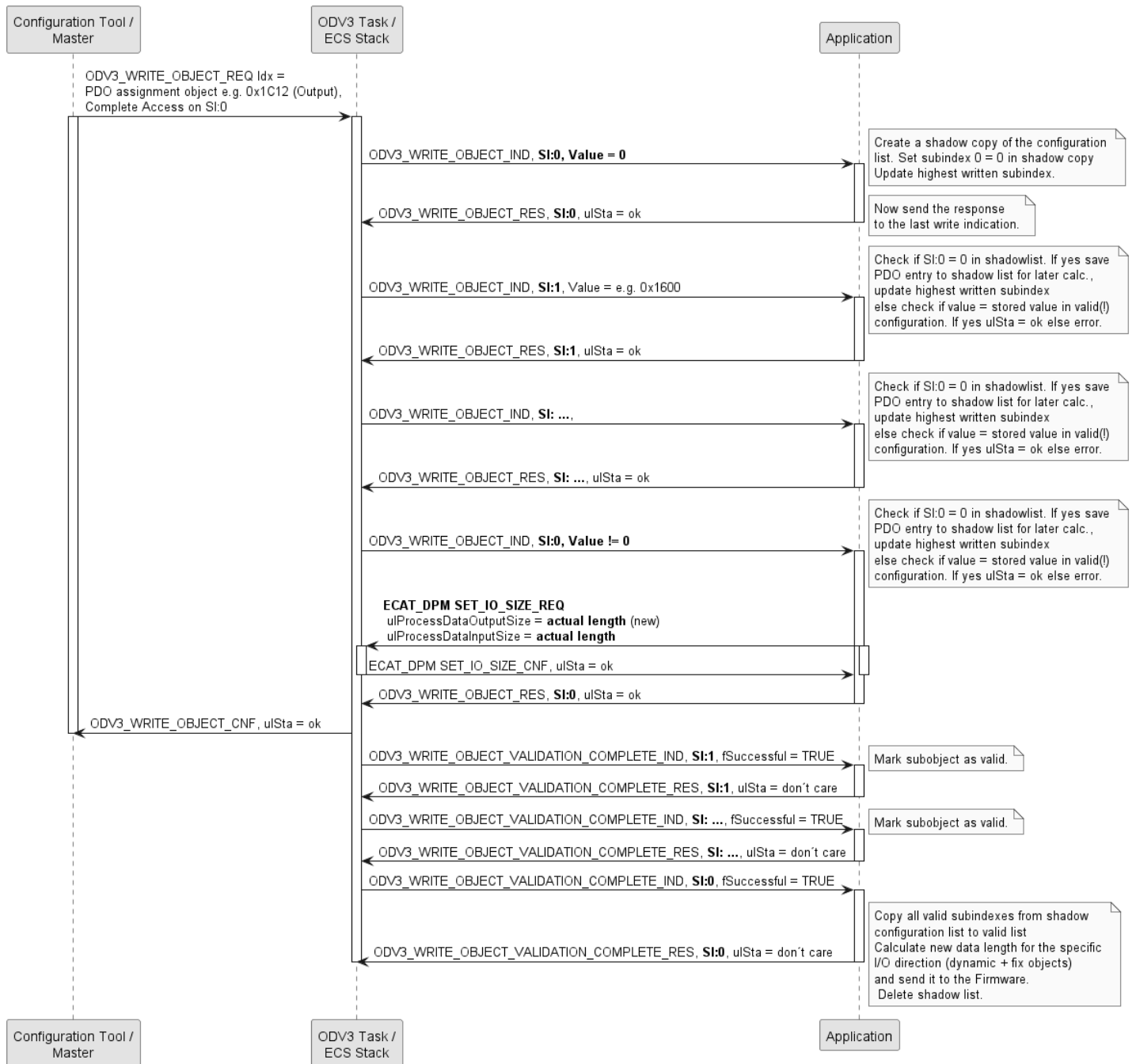


Figure 16. Dynamic PDO assignment with Complete Access: One application registered for write indications (successful)

The ODV3 task splits a single Complete Access from master into several requests. To get the subindexes in the correct order for PDO mapping, the application has to set the flag `ODV3_ACCESS_FLAGS_SUBINDEX_0_WRITE_0_FIRST` during object creation. The application also has to set the flag `ODV3_INDICATION_FLAGS_ON_WRITE_INVALIDATED`, see [4].

The application has to copy the current configuration list with all its values to a second list (shadow list). In the shadow list, the application saves the new configured data. At the end of a correct configuration sequence, the application copies the valid subobjects of the new configuration back to the current configuration list. In Case, the Complete Access fails (`ODV3_WRITE_OBJECT_CNF ulSta` is not zero), the configuration of the current list is still valid and process data length can be restored or is unchanged.

The sequence starts with setting subindex 0 to zero. In comparison to single access, the application does not need to send a `ECAT_DPM_SET_IO_SIZE_REQ` packet, because Set IO Size follows after the validation indication. The application receives values for the subindexes with the following write indications. In case the object is deactivated (subindex 0 = zero and writing is allowed), the application has to save the new values to the shadow list. If not, the application has to compare the current value with the new value. If they match, this is an allowed request otherwise this is an unallowed access (`0x06010003` = Subindex cannot be written, SI0 must be 0 for write access). The indication for subindex zero unequal to zero is checked to match highest written subindex.

All validation indications which are following show whether the written value is valid or whether an error has occurred. The validation for subindex 0 indicates the end of the configuration for the application. In case the validation is successful, the application has to copy the valid entries to the current configuration list. The application has to calculate the process data length and send the Set IO Size request to the EtherCAT Slave which contains the length of input and the length of output. One length has a new value.

Additional considerations

The main difference of the Complete Access compared to the standard access is that all write object validation packets are sent after the last write object response that has reached the ODV3 task. The amount of validation packets coming straight after the last write object response can be very high. As a result, the application has to take the following points into account:

* For LFW users: The application has to empty the mailbox frequently because otherwise the buffers in the packet queue for the mailbox runs out of packets. Depending on the amount of validation packets, it may necessary for the application to poll the mailbox with a higher frequency while the startup parameters are downloaded. Afterwards the application can decrease the frequency for a standard acyclic packet handling. * The master tries to change the EtherCAT state right after the download of the last startup parameter has been finished. Which means after the last write object confirmation reaches the master. The state change request leads to a process data validation in the slave. To avoid that the process data evaluation is done before the new data size is set, the validation responses from application side can be neglected (they are not evaluated by the ECS stack). If problems occur, it is also possible to define a timeout for the state changes in the ESI file.

5.6 Protocol-specific aspects to regard for ODV3 API when using the EtherCAT Slave stack

5.6.1 ODV3 access mask and flags

If the objects are administered by the application itself ("undefined mechanism") and the object list is also returned by the application, you have to take care of the ODV3 `usObjAccessMask` / `usObjAccessCompare` at the ODV3 List request. Objects in datatype area, which means objects less than 0x1000, are marked by the flag `ODV3_ACCESS_FLAGS_IS_DATATYPE_AREA`.

5.6.2 Free memory available for ODV3 objects can decrease after firmware update

Usually, a firmware update requires an increased amount of resources to be reserved for the firmware. Consequently, the resources available for your application might decrease after a firmware update. This also applies for the memory required by the EtherCAT stack to store ODV3 objects.

If the amount of memory for ODV3 objects your application uses is very close to the upper limit, the application might not work anymore after a firmware update and an error message such as "Out of Memory" might be issued. In this case, the amount of objects that can be held by the ODV3 component of the EtherCAT stack can be reduced. For example, if a new component is added to the firmware like a TCP stack. So, it is not guaranteed that the object dictionary will still fit on the netX after firmware update. Especially on netx52 targets, the amount of free memory is very small. A solution for this problem is to use the undefined mechanism when creating objects, see [4] for details.

Chapter 6 Application interface

The following chapters define the application interface of the EtherCAT Slave stack. The application itself has to be developed as a task according to the Hilscher's Task Layer Reference Model. In the following, the application task is named AP task.

The AP task's process queue is keeping track of all its incoming packets. It provides the communication channel for the underlying EtherCAT Slave Stack. Once, the EtherCAT Slave Stack communication is established, events received by the stack are mapped to packets that are sent to the AP task's process queue. On the one hand, every packet has to be evaluated in the AP task's context and corresponding actions be executed. On the other hand, Initiator-Services that are requested by the AP task itself are sent via predefined queue macros to the underlying EtherCAT Stack queues via packets as well.

All tasks belonging to the EtherCAT stack are grouped together according to their functionality they provide. The following overview shows the different tasks that are available within the EtherCAT stack.

EtherCAT component	Task	Description
Base component	<i>ECAT_ESM</i> task	This task provides the EtherCAT state machine and controls all related tasks.
	<i>ECAT_MBX</i> task	This task provides the mailbox of an EtherCAT slave.
CoE component	<i>ECAT_COE</i> task	This task splits the CoE messages according to their rule in the CANopen over EtherCAT.
	<i>ECAT_SDO</i> task	This task handles all SDO-based communications inside the EtherCAT CoE component.
	<i>ODV3</i> task	This task performs all accesses to the object dictionary (such as reading, writing, creating, deleting and maintaining objects). Its packet interface is described in [4]
EoE component	<i>ECAT_EOE</i> task	This task handles the Ethernet over EtherCAT.
FoE component	<i>ECAT_FOE</i> task	This task handles the File Access over EtherCAT.

Table 16. EtherCAT Slave stack components

The EtherCAT Slave Stack consists of several tasks dealing with certain aspects of the EtherCAT mailbox messages and cyclic communication. These can be accessed using the following queue names:

ASCII Queue Name	Description
'ECAT_ESM_QUE'	<i>ECAT_ESM</i> task queue name <i>ECAT_ESM</i> task handles all ESM states and AL control Events
'ECAT_COE_QUE'	<i>ECAT_COE</i> task queue name sending of CoE message will go through this queue
'ECAT_SDO_QUE'	<i>ECAT_SDO</i> task queue name <i>ECAT_SDO</i> task handles all SDO communications of the CoE Stack part
'ECAT_FOE_QUE'	<i>ECAT_FOE</i> task queue name <i>ECAT_FOE</i> task handles all File Access over EtherCAT communications
'ECAT_EOE_QUE'	<i>ECAT_EOE</i> task queue name <i>ECAT_EOE</i> task handles all EOE communications

Table 17. Summary of all queue names, which may be used by an AP task

The packets, which can be sent to those queues, will be detailed in the particular chapters. Furthermore, there is an *ECAT_DPM* task, which is not associated with a queue as it is only necessary for direct access to the DPM.

6.1 General

Service	Command	Command Code
Register application service	RCX_REGISTER_APP_REQ	0x2F10
	RCX_REGISTER_APP_CNF	0x2F11
Unregister application service	RCX_UNREGISTER_APP_REQ	0x2F12
	RCX_UNREGISTER_APP_CNF	0x2F13
Set ready service	ECAT_ESM_SETREADY_REQ	0x1980
	ECAT_ESM_SETREADY_CNF	0x1981
Initialization complete service	ECAT_ESM_INIT_COMPLETE_IND	0x198E
	ECAT_ESM_INIT_COMPLETE_RES	0x198F
Link status changed indication	RCX_LINK_STATUS_CHANGE_IND	0x198E
	RCX_LINK_STATUS_CHANGE_RES	0x198F

Table 18. Overview over the general packets of the EtherCAT Slave stack

6.1.1 Register application service

This service is described in DPM Interface Manual for netX based Products, see [1]. After registration, the stack will generate an initial AL status changed indication. When an application has been registered for indications with *RCX_REGISTER_APP_REQ*, the EtherCAT Slave stack may produce the following indications:

- AL status changed indication (*ECAT_ESM_ALSTATUS_CHANGED_IND*)
- Link status changed indication (*RCX_LINK_STATUS_CHANGE_IND*)
- Initialization complete indication (occurs only in context of linkable object modules)

Other indications of the EtherCAT Slave Stack will only be sent by the stack if an application has registered itself for that indication. This service is only informative, while the [Register for AL control changed indications service](#) goes beyond.

NOTE | It is required that the application returns all indications it receives as valid responses to the stack. It is not allowed to change any field in the packet header except *ulSta*, *ulCmd* and *ulLen*. Otherwise the stack will not be able to assign the response successfully.

6.1.2 Unregister application service

Using this service the application can unregister with the EtherCAT Slave stack: the stack will no longer generate indications. The service is described in DPM Interface Manual for netX based Products, see [1].

6.1.3 Set ready service

This service is used to notify the *ECAT_ESM* task of initialization completion of up to 32 tasks each represented by one bit of variable *ulReadyBits*. The lower 20 bits are reserved for the EtherCAT task and cannot be used by any application. The upper 12 bits are free to be used by the application. The *ECAT_ESM* task will wait for all required ready bits. It will not enable any state changes before all bits have been set.

NOTE | This service can only be used in the context of linkable object. It is also necessary to register the application with [Register application service](#) if the application shall receive the corresponding Initialization complete indication. At least one bit of variable *ulReadyBits* must be set.

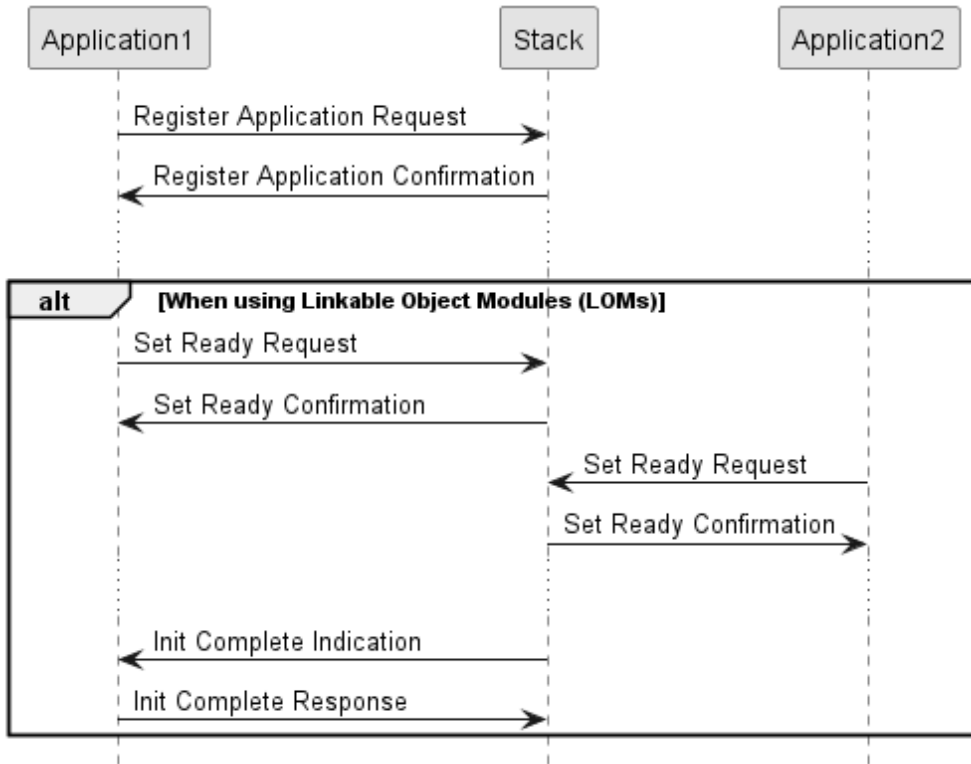


Figure 17. Set ready service request

As application 2 has not registered for indications (via *RCX_REGISTER_APP_REQ*) only application 1 receives the Initialization Complete Indication. The following ready waits bits are defined:

Value	Name	Description
0xffff0000	ECAT_READYWAIT_APPLICATION_MASK	ready wait bits
0x000fffff	ECAT_READYWAIT_STACK_MASK	
0x00008000	ECAT_READYWAIT_CYCLIC_DPM	
0x00100000	ECAT_READYWAIT_APP_TASK_1	
0x00200000	ECAT_READYWAIT_APP_TASK_2	
0x00400000	ECAT_READYWAIT_APP_TASK_3	
0x00800000	ECAT_READYWAIT_APP_TASK_4	
0x01000000	ECAT_READYWAIT_APP_TASK_5	
0x02000000	ECAT_READYWAIT_APP_TASK_6	
0x04000000	ECAT_READYWAIT_APP_TASK_7	
0x08000000	ECAT_READYWAIT_APP_TASK_8	
0x10000000	ECAT_READYWAIT_APP_TASK_9	
0x20000000	ECAT_READYWAIT_APP_TASK_10	
0x40000000	ECAT_READYWAIT_APP_TASK_11	
0x80000000	ECAT_READYWAIT_APP_TASK_12	

Table 19. Bitmask ulReadyBits of ECAT_ESM_SETREADY_REQ_DATA_T

As seen above up to 12 application tasks can set a ready bit. Notice that *ECAT_READYWAIT_CYCLIC_DPM* is used by the stack. The 'stack area' of the 32 ready waits bits covers the lower 20 bits, the 'application area' covers the upper 12 bits.

6.1.3.1 Set ready request packet

This request has to be sent from the application to the stack in order to cause the stack to wait until the ready bit of a task of the application has been set. As long as the ready bit has not been set, no state change of the stack happens.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	4
ulSta	uint32_t	0
ulCmd	uint32_t	0x1980
tData	ECAT_ESM_SETREADY_REQ_DATA_T	
ulReadyBits	uint32_t	see Bitmask ulReadyBits

Table 20. ECAT_ESM_SETREADY_REQ_T

6.1.3.2 Set ready confirmation packet

This confirmation will be sent from the stack to the application every time it receives a *ECAT_ESM_SETREADY_REQ* packet.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1981

Table 21. ECAT_ESM_SETREADY_CNF_T

6.1.4 Initialization complete service

This service indicates the completion of the initialization. It is used together with the Set ready service.

NOTE | This service can only be used in the context of linkable object. It is also necessary to register the application with [Register application service](#) in order to receive an Initialization complete indication. At least one bit of variable ulReadyBits must be set.

6.1.4.1 Init complete indication packet

This indication will be sent from the stack to the application when all bits which should be set in ready wait bits are set.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x198E

Table 22. ECAT_ESM_INIT_COMPLETE_IND_T

6.1.4.2 Init complete response packet

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	

Variable	Type	Description
uIDest	uint32_t	0x20
uLen	uint32_t	0
uSta	uint32_t	0
uCmd	uint32_t	0x198F

Table 23. ECAT_ESM_INIT_COMPLETE_RES_T

6.1.5 Link status changed service

This service indicates a link status change for a specific port e.g. cable plugged/unplugged in an Ethernet port. The stack polls the port status cyclically to generate the messages. This request is available from firmware/stack V4.4.0.2.

NOTE | It is necessary to register the application with [Register application service](#) in order to receive a link status changed indication.

6.1.5.1 Link status changed indication

This indication will be sent from the stack to every registered application.

6.1.5.2 Link status changed response

The application has to send this response to the stack after receiving the link status changed indication.

6.2 Configuration

Service	Command	Command Code
Set configuration service	ECAT_SET_CONFIG_REQ	0x2CCE
	ECAT_SET_CONFIG_CNF	0x2CCF
Set handshake configuration service	RCX_SET_HANDSHAKE_CONFIG_REQ	0x2F34
	RCX_SET_HANDSHAKE_CONFIG_CNF	0x2F35
Set IO Size service	ECAT_DPM_SET_IO_SIZE_REQ_T	0x2CC0
	ECAT_DPM_SET_IO_SIZE_CNF_T	0x2CC1
Set Station Alias service	ECAT_DPM_SET_STATION_ALIAS_REQ	0x2CC6
	ECAT_DPM_SET_STATION_ALIAS_CNF	0x2CC7
Get Station Alias service	ECAT_DPM_GET_STATION_ALIAS_REQ	0x2CC8
	ECAT_DPM_GET_STATION_ALIAS_CNF	0x2CC9

Table 24. Configuration packets overview

6.2.1 Set configuration service

The application has to use the set configuration service to configure the stack on startup.

NOTE As described in Dual-Port memory manual [1], it is required to send a channel initialization request to the EtherCAT Slave stack after the set configuration request is performed. The stack will not use the configuration until the channel initialization request is received.

For detailed information on the packet sequence, see [Section 2.2](#).

If this message has not been sent to the stack, the slave will not proceed further than to Pre-Operational state. If the master requests Safe-Operational, the slave will notify the master with the following code in the AL status code: `_ECAT_AL_STATUS_CODE_IO_DATA_SIZE_NOT_CONFIGURED_` (0x8001). For a list of available AL status codes please refer to the `EcsV4_Public.h` file.

Static PDO mapping vs. dynamic PDO mapping

This configuration service fully appropriate only for static PDO mapping. In case of dynamic PDO mapping, the application must send additionally a `[_set_io_size_request]` packet each time the input or output length has changed.

6.2.1.1 Set config request packet

The application has to send this request to the protocol stack in order to configure the stack with configuration parameters. The following applies: Configuration parameters will be stored internally in RAM. (In case of any error no data will be stored at all.) A channel initialization request is required to activate the configuration parameters.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	size of tData
ulSta	uint32_t	0
ulCmd	uint32_t	0x2CCE
tData	ECAT_SET_CONFIG_REQ_DATA_T	
tBasicCfg	ECAT_SET_CONFIG_REQ_DATA_BASIC_T	ECAT_SET_CONFIG_REQ_DATA_BASIC_T
tComponentsCfg	ECAT_SET_CONFIG_REQ_DATA_COMPONENTS_T	ECAT_SET_CONFIG_REQ_DATA_COMPONENTS_T

Table 25. ECAT_SET_CONFIG_REQ_T

The structures of the data part of the request [ECAT_SET_CONFIG_REQ_DATA_BASIC_T](#) and [ECAT_SET_CONFIG_REQ_DATA_COMPONENTS_T](#) of the data part and the possible defines are described in the following two separate subsections to keep it clear.

6.2.1.2 Basic configuration data structure

Variable	Type	Value/Range	Description
ulSystemFlags	UINT32	0, 1	Behavior at system start: 0 = automatic (default) 1 = application controlled For a description, see below this table.
ulWatchdogTime	UINT32	0, 20 – 65535	DPM Watchdog time in ms: 0 = off, default: 1000 Time for the application program for retriggering the EtherCAT slave watchdog. A value of 0 indicates that the watchdog timer is switched off. The watchdog will only be active after first triggering.
ulVendorId	UINT32	0 ... $2^{32} - 1$	Vendor ID Vendor IDentification number of the manufacturer of an EtherCAT device. Default: 0xE0000044 denoting device has been manufactured by Hilscher Vendor id, product code, and revision number for Hilscher, see Table 28
ulProductCode	UINT32	0 ... $2^{32} - 1$	Product code Product code of the device. Default: 0x00020004
ulRevisionNumber	UINT32	0 ... $2^{32} - 1$	Revision number Revision number of the device as specified by the manufacturer.
ulSerialNumber	UINT32	0 ... $2^{32} - 1$	Serial number Serial number of the device. Default: 0 Value 0 forces the stack to read the serial number from the security memory or Flash device label in the device. If security memory or Flash device label is present but can't be accessed correctly, value 0 is used.
ulProcessDataOutput Size	UINT32	netX 100/500 ¹⁾ : 0...512 – ulProcessDataInputSize netX 50/51/52 ²⁾ :+ 0...1024	Process data output size (in bytes) Default: 4 Byte netX100/500 only: The sum of input and output data is limited to 512 Bytes*.
ulProcessDataInput Size	UINT32	netX 100/500 ¹⁾ : 0...512 – ulProcessDataOutputSize netX 50/51/52 ¹⁾ :+ 0...1024	Process data input size (in bytes) Default: 4 Byte netX100/500 only: The sum of input and output data is limited to 512 Bytes*.
ulComponent Initialization	UINT32	Bit mask	Component initialization bit mask, enables or disables certain components of the EtherCAT Slave stack. For a list, see below.
ulExtensionNumber	UINT32	0 ... $2^{32} - 1$	Currently not used Number which identifies an additional configuration structure default: 0.

¹⁾ netX 100/500: The sum of roundup(input data length) and roundup(output data length) may not exceed 512 Bytes (where roundup() means round up to the next multiple of 4. If either the input data length or the output data length exceeds 256 Bytes, the device description file delivered with the device requires modifications in order to work properly, also ECAT_SET_CONFIG_SMLENGTH has to be set.
²⁾ netX 50/51/52: The sum of input data length and output data length may not exceed 2048 Bytes and 1024 in each direction.

Table 26. Basic configuration data ECAT_SET_CONFIG_REQ_DATA_BASIC_T

Starting with version 4.6.0 the EtherCAT Slave stack supports simultaneous setting of input and output data length to 0. The use case for is for example modular devices: Set both input and output length to 0 and use the [Set IO Size service](#) to set the calculated input and output data length.

Parameter ulSystemFlags

The start of the device can be performed either application controlled or automatically. The following flags are defined:

Value	Name	Description
0x00000000	ECAT_SET_CONFIG_SYSTEMFLAGS_AUTOSTART	Network connections are opened automatically without taking care of the state of the host application. Communication with an EtherCAT master after starting the EtherCAT Slave is allowed without BUS_ON flag, but the communication will be stopped if the <i>BUS_ON</i> flag changes state to 0.
0x00000001	ECAT_SET_CONFIG_SYSTEMFLAGS_APP_CONTROLLED	The channel firmware is forced to wait for the host application Application to wait for the <i>BUS_ON</i> flag in the communication change of state register. For further information, see [1]. Communication with EtherCAT Master is allowed only with the <i>BUS_ON</i> flag

Table 27. Flags for ulSystemFlags

Automatic

IMPORTANT | If the master sets the slave to Operational state when Automatic has been chosen, probably the application will not be initialized completely.

Application controlled

IMPORTANT | If the initialization of the slave application is to be controlled by the slave application itself, Application controlled must be chosen. The master is only able to change the state of the slave in case of the slave application setting the *BUS_ON* flag.

IMPORTANT | If Application controlled (1) is chosen and a watchdog error occurs, the stack will not be able to reach the Operational or the Safe-Operational state. In this case, a channel reset is required.

For more information concerning the bus startup parameter, see section Controlled or Automatic Start in [1].

Parameter ulVendorId, ulProductCode and ulRevisionNumber

The values for the parameters ulVendorId, ulProductCode and ulRevisionNumber can be taken from the XML file, which is bundled with the particular firmware. The following default value sets for the identification data has been defined:

Firmware	Vendor ID	Product code	Actual revision number
cifX	0xE0000044	0x00000001	0x00060004
comX100	0xE0000044	0x00000003	0x00060004
comX51	0xE0000044	0x0000002B	0x00060004
comX51 Rotary	0xE0000044	0x00000041	0x00060004
netIC50	0xE0000044	0x0000000B	0x00020004
netIC52	0xE0000044	0x00000033	0x00020004
netJACK51	0xE0000044	0x0000002C	0x00060004
netJACK100	0xE0000044	0x00000022	0x00060004
netRapid51	0xE0000044	0x0000003A	0x00060004
netRapid52	0xE0000044	0x00000030	0x00060004
NXIO50	0xE0000044	0x0000000F	0x00060004
NXIO100	0xE0000044	0x00000002	0x00060004
netX100	0xE0000044	0x0000000C	0x00060004
netX500	0xE0000044	0x00000009	0x00060004
netX50	0xE0000044	0x0000000A	0x00060004
netX51	0xE0000044	0x00000028	0x00060004
netX52	0xE0000044	0x0000002E	0x00060004

Table 28. Values for the parameters ulVendorId, ulProductCode and ulRevisionNumber

Parameter ulComponentInitialization

The value *ulComponentInitialization* is used to enable or disable certain component parameter evaluation of the EtherCAT Slave stack. If a bit is set, the related data structure is evaluated in the EtherCAT slave stack.

The following flags are defined for ulComponentInitialization:

```

#define ECAT_SET_CONFIG_COE                0x00000001
#define ECAT_SET_CONFIG_EOE                0x00000002
#define ECAT_SET_CONFIG_FOE                0x00000004
#define ECAT_SET_CONFIG_SOE                0x00000008
#define ECAT_SET_CONFIG_SYNCMODES         0x00000010
#define ECAT_SET_CONFIG_SYNCPDI           0x00000020
#define ECAT_SET_CONFIG_UID                0x00000040
#define ECAT_SET_CONFIG_AOE                0x00000080
#define ECAT_SET_CONFIG_BOOTMBX           0x00000100
#define ECAT_SET_CONFIG_DEVICEINFO         0x00000200
#define ECAT_SET_CONFIG_SMLENGTH           0x00000400
/* 0x00000800--0x00004000 in use by ECSv5 */
    
```

The flags have the following meaning:

Bit	Description
0	CoE parameter evaluation 0 - disabled 1 - enabled
1	EoE parameter evaluation 0 - disabled 1 - enabled
2	FoE parameter evaluation (component activated by default in most targets) 0 - disabled 1 - enabled
3	SoE parameter evaluation 0 - disabled 1 - enabled
4	Synchronization modes parameter evaluation 0 - disabled 1 - enabled
5	Sync PDI parameter evaluation 0 - disabled 1 - enabled
6	Unique identification parameter evaluation 0 - disabled 1 - enabled
7	AoE parameter evaluation 0 - disabled 1 - enabled
8	Bootstrap Mailbox parameter evaluation 0 - disabled 1 - enabled
9	Device Info parameter evaluation 0 - disabled 1 - enabled
10	Sm Length parameter evaluation 0 - disabled 1 - enabled
11 -31	Reserved

Table 29. Parameter ulComponentInitialization

6.2.1.3 Components configuration data structure

Variable	Type
	Description
tCoECfg	ECAT_SET_CONFIG_COE_T CoE configuration parameters
tEoECfg	ECAT_SET_CONFIG_EOE_T EoE configuration parameters
tFoECfg	ECAT_SET_CONFIG_FOE_T FoE configuration parameters
tSoECfg	ECAT_SET_CONFIG_SOE_T SoE configuration parameters
tSyncModesCfg	ECAT_SET_CONFIG_SYNCMODES_T Sync modes configuration parameters
tSyncPdiCfg	ECAT_SET_CONFIG_SYNCPDI_T Sync PDI configuration parameters
tUidCfg	ECAT_SET_CONFIG_UID_T Unique identification configuration parameters
tBootMxbCfg	ECAT_SET_CONFIG_BOOTMBX_T Bootmailbox configuration parameter
tDeviceInfoCfg	ECAT_SET_CONFIG_DEVICEINFO_T Device info configuration parameter
tSmLengthCfg	ECAT_SET_CONFIG_SMLENGTH_T Syncmanager configuration parameter

Table 30. ECAT_SET_CONFIG_REQ_DATA_COMPONENTS_T

AoE configuration parameter

The AoE component, does not have a configuration parameter.

CoE configuration parameter

Variable	Type
	Description
bCoeFlags	uint8_t Flags for CoE configuration
bCoeDetails	uint8_t CoE details refer to value 'CoE details' of category 'General' in the SII)
ulOdIndicationTimeout	uint32_t Timeout for object dictionary indications in milliseconds
ulDeviceType	uint32_t Device type in object 0x1000 of object dictionary
usReserved	uint16_t

Table 31. ECAT_SET_CONFIG_COE_T

The following flags for CoE configuration are defined:

6.2.1.4 Object dictionary creation mode:

0 - Object dictionary shall be created with default objects +
 1 - Object dictionary shall not be created with default objects, only minimal object dictionary (contains objects 0x1000 and 0x1018) is created, the user has to provide objects

ECAT_SET_CONFIG_COEFLAGS_USE_CUSTOM_OD	0x01
----------------------------------------	------

Value	Name	Description
0x01	ECAT_SET_CONFIG_COEDetails_ENABLE_SDO	Enable SDO
0x02	ECAT_SET_CONFIG_COEDetails_ENABLE_SDOINFO	Enable SDO information
0x04	ECAT_SET_CONFIG_COEDetails_ENABLE_PDOASSIGN	Enable PDO assign
0x08	ECAT_SET_CONFIG_COEDetails_ENABLE_PDOCONFIGURATION	Enable PDO configuration
0x10	ECAT_SET_CONFIG_COEDetails_ENABLE_UPLOAD	Enable PDO upload at startup
0x20	ECAT_SET_CONFIG_COEDetails_ENABLE_SDOCOMPLETEACCESS	Enable SDO complete access

Table 32. Flag for bCoeDetails

The flags for CoE details refer to the value “CoE details“ of the category “General” in the SII [15]. They will be directly copied from the configuration request packet to the SII. If the CoE component of the stack is not configured by user given parameters (*ECAT_SET_CONFIG_COE* not used) the following default value applies:

- Enable SDO
- Enable SDO Information
- Enable PDO upload at startup
- Enable SDO complete access

EoE configuration parameter

Variable	Type	Description
ulReserved	uint32_t	

Table 33. ECAT_SET_CONFIG_EOE_T

FoE configuration parameter

The FoE component is activated by default to allow firmware updates, even if it is not set here. Set it here to match the entry in the ESI file or adapt the ESI file. This might influence the opportunities in some masters, but will not deactivate the functionality in the slave. To have control over the downloads (e.g. deny), use the components options. The FoE configuration data structure contains the following parameter:

Variable	Type	Description
ulTimeout	uint32_t	FoE timeout in milliseconds has to be unequal to 0 (default: 1000)

Table 34. ECAT_SET_CONFIG_FOE_T

NOTE | For all targets supporting a file system, FoE is activated by default. For targets with no file system, e.g. CIFX targets, FoE is deactivated by default.

SoE configuration parameter

Variable	Type
	Description
ulIdnIndicationTimeout	uint32_t Timeout currently not supported

Table 35. ECAT_SET_CONFIG_SOE_T

NOTE | CoE and SoE cannot be used at the same time!

Sync modes configuration parameter

Variable	Type
	Description
bPDInHskMode	uint8_t Input process data handshake mode: only Buffered Host Controlled supported
bPDInSource	uint8_t Input process data trigger source (which triggers the input handshake cell) For values and modes, see Flag for bSyncSource .
usPDInErrorTh	uint16_t Threshold for input process data handshake handling errors <i>Note</i> : this is the error threshold of the EtherCAT sync manager for the (master) outputs (usually SM2)
bPDOutHskMode	uint8_t Output process data handshake mode: only Buffered Host Controlled supported
bPDOutSource	uint8_t Output process data trigger source (which triggers the output handshake cell) For values and modes, see Flag for bSyncSource .
usPDOutErrorTh	uint16_t Threshold for output process data handshake handling errors <i>Note</i> : this is the error threshold of the EtherCAT sync manager for the (master) inputs (usually SM3)
bSyncHskMode	uint8_t Synchronization handshake mode: only Device Controlled mode supported
bSyncSource	uint8_t Synchronization source for the special sync handshake cell (may be used for an additional sync decoupled from process data) For values and modes, see Flag for bSyncSource
usSyncErrorTh	uint16_t Threshold for synchronization handshake handling errors

Table 36. ECAT_SET_CONFIG_SYNCMODES_T

The following flags are defined:

Value	Name
	Description
0x00	ECAT_DPM_SYNC_SOURCE_FREERUN no synchronization in use
0x02	ECAT_DPM_SYNC_SOURCE_SYNC0 SYNC0 signal used as synchronization trigger
0x03	ECAT_DPM_SYNC_SOURCE_SYNC1 SYNC1 signal used as synchronization trigger
0x22	ECAT_DPM_SYNC_SOURCE_SM2 SM2 used as synchronization trigger
0x23	ECAT_DPM_SYNC_SOURCE_SM3 SM3 used as synchronization trigger

Table 37. Flag for bSyncSource

The application can be synchronized with the the EtherCAT bus cycle. For information on the handshake mechanism, see [1].

When starting with process data exchange, the application has to do a handshake with the netX once to receive the first process data. In order to do the first handshake toggle, the application has to call the xChannellIORead function once. A common use case is to synchronize the process data exchange on SyncManager2 event with activated handshake mode. If so, the following applies. After the slave has received a frame, this triggers an interrupt on which the slave copies the output process data (master to salve) received from the bus into the triple buffer of the slave controller. As soon as data is copied, the output valid mark is set and the AP task copies data into the local memory. The input handshake bit in the dual-port memory is toggled and the local cycle of the application starts. After the slave has copied the received data, the slave gives back the input handshake bit back to the netX. Now the slave copies its input data (slave to master) to the triple buffer and gives the output handshake bit to the netX that copies the data to the bus and gives the handshake back again. A synchronization for input process data exchange (slave to master) is not necessary because this can be done by the application right after output data is received and thus as fast as possible.

The values for the discussed use case SM2 synchronous are:

Parameter	Description	Setting
bPDIInHskMode	Handshakebits which show that Data is copied to DPM	<i>RCX_IO_MODE_BUFF_HST_CTRL</i>
bPDIInSource	Process data trigger source for inputs (master → slave)	<i>ECAT_DPM_SYNC_SOURCE_SM2</i>
bPDIOutHskMode	Handshakebits which show that Data is copied to DPM	<i>RCX_IO_MODE_BUFF_HST_CTRL</i>
bPDIOutSource	Process data trigger source for outputs (slave → master)	<i>ECAT_DPM_SYNC_SOURCE_FREERUN</i>
bSyncSource	for special sync handshake cell, not needed	<i>ECAT_DPM_SYNC_SOURCE_FREERUN</i>
bSyncHskMode	for special sync handshake cell, not needed	<i>RCX_SYNC_MODE_OFF</i>

Table 38. Example for SM2 synchronous mode

A similar configuration is used for DC synchronous mode, which eliminates the jitter of the bus cycle. The only difference is to set the bPDIInSouce to *ECAT_DPM_SYNC_SOURCE_SYNC0*. The synchronization on SM3 event makes sense in case only inputs are transmitted.

NOTE All values mentioned above have no influence on the real physical sync signal generation by the ESC. Whether it is active or not and which sync signal. This is done by the following parameters in *ECAT_SET_CONFIG_SYNCPDI_T* and from the master side which additionally has to activate signals by writing to ESC registers.

If the slave requires to support multiple synchronisation modes, the application can use the [Set handshake configuration service](#) to reconfigure the synchronization mode.

Sync PDI configuration parameter

Variable	Type
	Description
bSyncPdiConfig	uint8_t Sync PDI configuration (Esc register 0x151) Value 0 - 255 (default 0xCC)
usSyncImpulseLength	uint16_t Sync impulse length (in units of 10 ns) Value 0 - 5535 (default 1000)
bReserved	uint8_t

Table 39. ECAT_SET_CONFIG_SYNCPDI_T

Even if a sync signal in the slave is activated through the configuration of the EtherCAT master, but the application does not need the sync interrupt for synchronisation, the Interrupt has to be activated. This is necessary because the stack uses the interrupt to monitor the presence of a sync signal. Otherwise, the stack cannot reach Operational state. Starting with version 4.7.0, the Sync Interrupts bits 3 and 7 are always enabled by default in loadable firmware.

The following flags are defined for bSyncPdiConfig:

Value	Name	Description
0x01	ECAT_SET_CONFIG_SYNCPDI_SYNC0_OUTPUT_TYPE_MASK	SYNC0 Output type 0 - Push Pull 1 - OpenDrain Note: netX100/500 firmware ignores this bit. They always work as Push Pull.
0x02	ECAT_SET_CONFIG_SYNCPDI_SYNC0_POLARITY_MASK	SYNC0 Polarity 0 - low active 1 - high active
0x04	ECAT_SET_CONFIG_SYNCPDI_SYNC0_OUTPUT_ENABLE_MASK	SYNC0 Output enable/disable 0 - disabled 1 - enabled
0x08	ECAT_SET_CONFIG_SYNCPDI_SYNC0_IRQ_ENABLE_MASK	SYNC0 mapped to PDI-IRQ 0 - disabled 1 - enabled
0x10	ECAT_SET_CONFIG_SYNCPDI_SYNC1_OUTPUT_TYPE_MASK	SYNC1 Output type 0 - Push Pulld 1 - OpenDrain Note: netX100/500 firmware ignores this bit. They always work as Push Pull.
0x20	ECAT_SET_CONFIG_SYNCPDI_SYNC1_POLARITY_MASK	SYNC1 Polarity 0 - low active 1 - high active
0x40	ECAT_SET_CONFIG_SYNCPDI_SYNC1_OUTPUT_ENABLE_MASK	SYNC1 Output enable/disable 0 - disabled 1 - enabled
0x80	ECAT_SET_CONFIG_SYNCPDI_SYNC1_IRQ_ENABLE_MASK	SYNC1 mapped to PDI-IRQ 0 - disabled 1 - enabled

Table 40. Definitions for parameter bSyncPdiConfig of ECAT_SET_CONFIG_SYNCPDI_T

Unique identification configuration parameter

Variable	Type	Description
usStationAlias	uint16_t	0x00 not evaluated here, handling by firmware 0x01 - 0xFF
usDeviceIdentification Value	uint16_t	0x00 switch off handling, activate handling 0x01 - 0xFF

Table 41. ECAT_SET_CONFIG_UID_T

The value of *usStationAlias* will be written into the EEPROM and the register 0x12 of the ESC. The station alias address can be written by a configuration tool to the EEPROM and is transferred to the ESC register at startup of the device. If it is set by configuration parameter, this possibility is no longer available, because the value configured by the tool will be

overwritten by the value of *usStationAlias*. So for most use cases the parameter should be set to zero. The Configured Station Alias can also be changed by an application using the [Set Station Alias service](#).

If it is possible to set an address from the device side (via a rotary switch, a display or by other ways), a value unequal to zero must be set in *usDeviceIdentificationValue* to activate the address handling in the slave stack. Otherwise, set the value to zero to deactivate the handling. If the mechanism is activated, sending the packet RCX_SET_FW_PARAMETER_REQ is needed in addition to update the address each time it has changed and also once before the bus is switched on. See [Explicit Device Identification](#) for details.

Boot mailbox configuration parameter

Variable	Type	Description
usBootstrapMbx Size	uint16_t	0x00 switch off Bootstrapmailbox, 128 - max (size is chip dependent)

Table 42. ECAT_SET_CONFIG_BOOTMBX_T

The bootstrap mailbox size has a default value of 128 Byte which is defined in the configuration file. If the component parameter evaluation is enabled by setting the flag in *ulComponentInitialization*, this value can be changed by the configuration parameter. If the configuration parameter *usBootstrapMbxSize* is set to zero, it deactivates the Bootstrap Mailbox. If the parameter is set to a value different from zero, it overwrites the default value. The minimum possible value is 128 Byte. The maximum configurable size is chip dependent e.g. 3200 bytes - processdata size (three times counted because of triple buffer) for netX 50/51/52 or 896 bytes - processdata size (three times) per direction for netX 100/500. Mailboxes always have the same size for both directions and the size has to be 4 byte aligned.

Device info configuration parameter

Variable	Type	Description
bGroupIdxLength	uint8_t	length of char array <i>_szGroupIdx</i> , values 0 not set, 1 - 127 length
szGroupIdx[127]	char	ascii code of the group name, length 127 byte
bImageIdxLength	uint8_t	set to 0, parameter is not evaluated by firmware
szImageIdx[255]	char	length 255 byte, set to 0
bOrderIdxLength	uint8_t	length of char array <i>_szOrderIdx</i> , values 0 not set, 1 - 127 length
szOrderIdx[127]	char	ascii code of the order name, length 127 byte
bNameIdxLength	uint8_t	length of char array <i>_szNameIdx</i> , values 0 not set, 1 - 127 length
szNameIdx[127]	char	ascii code of the name of the device, length 127 byte

Table 43. ECAT_SET_CONFIG_DEVICEINFO_T

The Device Info configuration parameter data structure *ECAT_SET_CONFIG_DEVICEINFO_T* for the SII relate to entries in the ESI file as shown in following table.

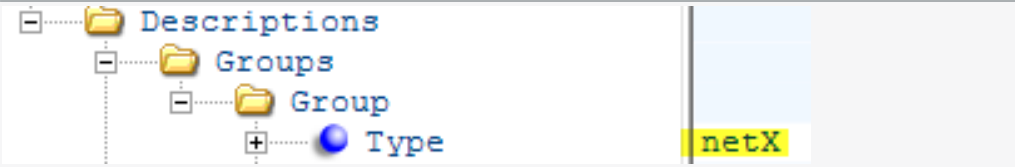
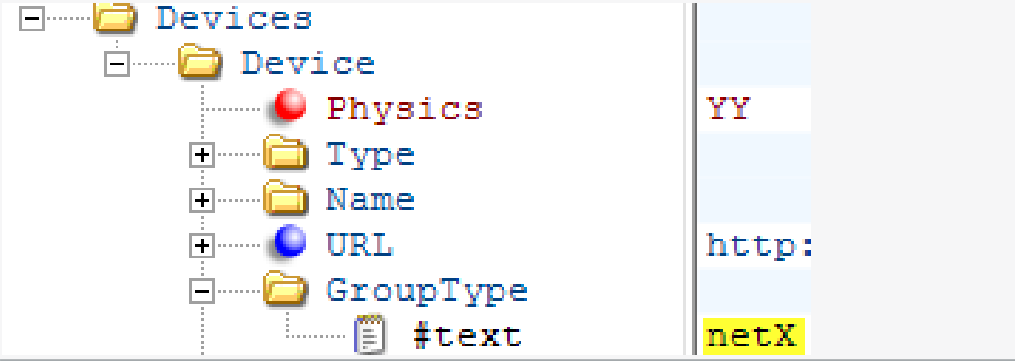
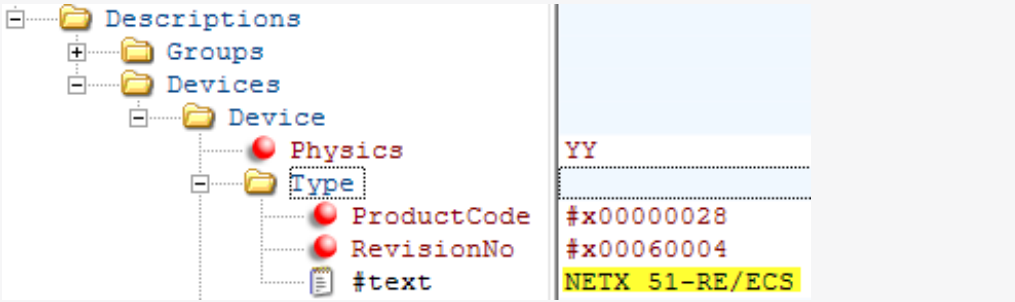
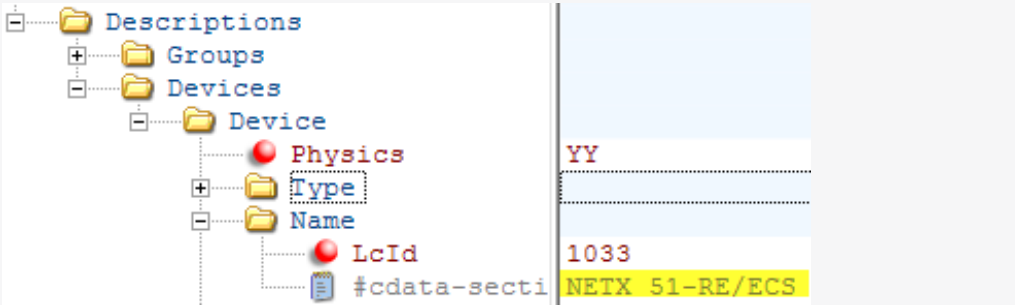
Parameter	Meaning
szGroupIdx[127]	ASCII code of the group name of the device SII entry Grouldx related to ESI entry 
	Further used to add the device to the group: 
szImagIdx[255]	not evaluated by firmware
szOrderIdx[127]	ASCII code of the order name of the device SII entry OrderIdx related to ESI entry: 
szNameIdx[127]	ASCII code of the name of the device SII entry NameIdx related to ESI entry: 

Table 44. Device info configuration parameters

If the component parameter evaluation is enabled by setting the appropriate flag in *ulComponentInitialisation*, the Device Info can be set by the configuration parameters. If not, the Hilscher default values of the target will be used (as in example esi file). It is possible to set only the needed values and deactivate parameters by setting their length to zero.

NOTE Despite the length information, the parameters have to be set in the maximum array length, even if the parameter length is shorter than possible or if the length is set to zero. The strings can be filled up with zeros.

Sm length configuration parameter

Starting with version 4.7.0, the SyncManager mailboxes can be configured for SM0 and SM1 as well as the start addresses for SM2 and SM3.

Variable	Type	Description
usMailboxSize	uint16_t	used for the output (SM0) and for the input mailbox (SM1) as well, min is 128, max is available process-memory byte size / 2
usSM2StartAddress	uint16_t	sets the start address of the address space for output data, min value 0x1100, max value chip dependent
usSM3StartAddress	uint16_t	sets the start address of the address space for input data, min value 0x1104, max value chip dependent

Table 45. ECAT_SET_CONFIG_SMLENGTH_T

The mailbox size for SM0 and SM1 have a default value of 128 bytes. The SM2/3 default start addresses depend on the chip. To change the default settings, the component parameter evaluation has to be set to enabled (flag in *ulComponentInitialization*).

If the configuration parameter *usMailboxSize* is set to a value less than 128 bytes, the minimum possible value of 128 bytes is used or refused (depending on the version). The maximum configurable size is chip dependent and also depends on the needed process data size. Mailboxes always have the same size for both directions and the size has to be 4 byte aligned.

- The calculation for netX 50/51/52: 3200 bytes minus processdata size (three times counted because of triple buffer) for each direction
- The calculation for netX 100/500: 896 bytes minus processdata size (three times) per direction, but maximum 780 Byte.

The configuration parameter *usSM2StartAddress* defines the start address for the output (master/network → slave) process data image. The address must be set directly after the mailbox data image to utilize space. If e.g. the mailbox has the default value of 128 Byte, the start address has to be 0x1100, because the mailboxes start at 0x1000 and have a length of 2 * 0x80 byte.

The configuration parameter *usSM3StartAddress* defines the start address for the input (slave → master/network) process data image. The address can be set directly after the output data image to utilize space. If e.g. the output image is 256 byte long and *usSM2StartAddress* starts at 0x1100, the start address for the input image has to be at minimum 0x1400, because the process data uses triple buffers. The rest of the address space can be used for input data. If the example values are used with netX 500, this is 768 (1792 - 2 * 128 - 3 * 256) byte, which means 256 bytes usable because of the triple buffer.

It is necessary to configure both syncmanager addresses even for devices which only have input data. As well as the mailboxes also the processdata syncmanagers have to be 4 byte aligned. This means the tripplebuffers for netX100, 500, 51, 52, must each be 4 byte aligned. Values have to follow the calculatuion: $usSM2StartAddress + 3 * ulProcessDataOutputSize + 3$ & $(-3 \leq usSM3StartAddress$.

If this component is used, the configured values are automatically written to the virtual EEPROM by the Ethercat stack. The values for the default syncmanager length are defined by the amount of configured process data (set in *ECAT_SET_CONFIG_REQ_DATA_BASIC_T* of the configuration packet). This replaces the standard value 200 bytes used for Hilscher devices. Take care to adapt the ESI file to the values you use in *ECAT_ESM_CONFIG_SMLENGTH_T*.

6.2.1.5 Set config confirmation packet

The stack sends this confirmation to the application.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x2CCF

Table 46. ECAT_SET_CONFIG_CNF_T

6.2.2 Set extended configuration service

The application can use this service for additional configuration parameters that are not included in the standard configuration (basic and component parameters). The first entry in the packets data part is the structure type that defines the content of the following data with additional size. Actually there are the following types defined as enumeration.

Value	Name	Description
4	ECAT_SET_CONFIG_STRUCTURE_TYPE_SMS	
5	ECAT_SET_CONFIG_STRUCTURE_TYPE_BOOTMBX	
6	ECAT_SET_CONFIG_STRUCTURE_TYPE_END	
		no valid type, just for check

Table 47. ECAT_SET_CONFIG_EXT_STRUCTURE_TYPE_E

Structuretype: Standard syncmanager start addresses

Variable	Type	Description
usMailboxSize	uint16_t	used for the output (SM0) and for the input mailbox (SM1) as well, min is 128, max is 1522 Byte
usStdMbxSm0StartAddress	uint16_t	sets the start address of the space for mbx out data, min value 0x1000
usStdMbxSm1StartAddress	uint16_t	sets the start address of the space for mbx in data, min value 0x1000
usSM2StartAddress	uint16_t	sets the start address of the address space for output data, min value 0x1000
usSM3StartAddress	uint16_t	sets the start address of the address space for input data, min value 0x1000

Table 48. ECAT_SET_CONFIG_EXT_DATA_TYPE_SMS_T

Structuretype: Bootstrap mailbox start addresses

Variable	Type	Description
usBootstrapMbxSize	uint16_t	0 switches off special size for Mailbox for BOOT state value 128 - 1522 Mailboxes always have the same size for both directions
usBootMbxSm0StartAddress	uint16_t	0 use default rule sets the start address of the space for mbx out data in boot state, min value 0x1000
usBootMbxSm1StartAddress	uint16_t	0 use default rule sets the start address of the space for mbx in data in boot state, min value 0x1000

Table 49. ECAT_SET_CONFIG_EXT_DATA_TYPE_BOOTMBX_T

Extended config: Main packet structure

Variable	Type	Description
tExtCnf_SMS	ECAT_SET_CONFIG_EXT_DATA_TYPE_SMS_T	configures syncmanager protected start address
tExtCnf_BootMbx	ECAT_SET_CONFIG_EXT_DATA_TYPE_BOOTMBX_T	configures mailbox if slave is in boot state

Table 50. ECAT_SET_CONFIG_EXT_STRUCTUREDATA_T

Variable	Type	Description
usStructureType	uint16_t	Type defining following data, use ECAT_SET_CONFIG_STRUCTURE_TYPE_E
tStructureData	ECAT_SET_CONFIG_EXT_STRUCTUREDATA_T	

Table 51. ECAT_SET_CONFIG_EXT_REQ_DATA_T

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
tData	ECAT_SET_CONFIG_EXT_REQ_DATA_T	

Table 52. ECAT_SET_CONFIG_EXT_REQ_T

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	no data part

Table 53. ECAT_SET_CONFIG_EXT_CNF_T

6.2.3 Set handshake configuration service

The application can use this service to (re)configure the mode of operation of the process data and synchronization handshake. This service is optional and only needed if the configuration can be changed.

The handshake configuration is also adjustable with the [Sync Configuration Parameter](#) of the set configuration service. Section [Sync modes configuration parameter](#) contains the values range and a brief description of the parameters. The EtherCAT slave supports the Host Controlled handshake mode. The application **must not send the set handshake configuration request when the slave is in a process data exchange mode**: SafeOP or OP. Especially when switching from a DC mode to a non-DC mode it can come to a deadlock situation when process data exchange is not stopped from application side (cifx API's XChannelIORead/Write commands) before switching the mode. In this case, the stack can miss the toggling of handshake bits from application side and never toggles the bits back.

NOTE | If DC and a non-DC modes are supported, the actual setting from master side can be determined by *usSyncControl* value of the [AL control changed indication packet](#).

For [Set handshake configuration request](#) and [Set handshake configuration confirmation](#) see [\[2\]](#).

6.2.4 Set IO Size service

The application can use this service to change the process data input length and/or the process data output length. This service does not affect any other parameter.

The main use case for this service is to set new data length for dynamic process data configuration. Section [Dynamic PDO mapping](#) shows the sequences and when the application has to use the Set IO Size service.

The application must not use this service, as soon as the slave has reached SafeOp or Op state and is exchanging data.

6.2.4.1 set io size request packet

The application can use this service to change the size of the I/O image.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	8
ulSta	uint32_t	0
ulCmd	uint32_t	0x2CCC0
tData	ECAT_DPM_SET_IO_SIZE_REQ_DATA_T	
ulProcessDataOutputSize	uint32_t	Process Data Output Length 0 - 512 (netX100/500), 0 - 1024 (netX50/netX51)
ulProcessDataInputSize	uint32_t	Process Data Input Length 0 - 512 (netX100/500), 0 - 1024 (netX50/netX51)

Table 54. ECAT_DPM_SET_IO_SIZE_REQ_T

6.2.4.2 set io size confirmation packet

The stack will send this confirmation to the application.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x2CCC1

Table 55. ECAT_DPM_SET_IO_SIZE_CNF_T

6.2.5 Set Station Alias service

This service is used to set a Station Alias to the register 0x0012 in the EtherCAT Slave. The value to be set in the register is represented with the variable *usStationAlias* of the request packet.

In the past, the application had to use several packets in order to set the Station Alias Address. Now the EtherCAT Slave stack executes the Station Alias address handling. Starting with version 4.5 (starting with version 4.6 for cifX cards), the Station Alias address (Second Station Address) is saved non-volatile and afterwards set to the ESC register by the EtherCAT stack. As a result, the application does not need to handle the Station Alias address anymore compared to earlier EtherCAT Slave stack versions. The netX 52 firmware has not implemented this feature yet and the application has to do the Station Alias address handling.

In case the the Station Alias address handling is implemented in the application, the application overwrites the values set by the firmware (SII and ESC register value). We recommend to remove the Station Alias address handling from the application (except for netx52).

6.2.5.1 set station alias confirmation packet

This request has to be sent from the application to the stack in order to set a station alias

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	2
ulSta	uint32_t	0
ulCmd	uint32_t	0x2CCC6
tData	ECAT_DPM_SET_STATION_ALIAS_REQ_DATA_T	

Variable	Type	Description
usStationAlias	uint16_t	Configured Station Alias also called Second Station Address

Table 56. ECAT_DPM_SET_STATION_ALIAS_REQ_T

6.2.5.2 set station alias confirmation packet

The stack will send this confirmation to the application.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x2CC7

Table 57. ECAT_DPM_SET_STATION_ALIAS_CNF_T

6.2.6 Get Station Alias service

This service is used to request a formerly set Station Alias from the protocol stack. The desired Station Alias is delivered in variable *usStationAlias* of the confirmation packet.

6.2.6.1 Get Station Alias request packet

This request has to be sent from the application to the stack

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x2CC8

Table 58. ECAT_DPM_GET_STATION_ALIAS_REQ_T

6.2.6.2 Get Station Alias confirmation packet

The stack will send this confirmation to the application.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	2
ulSta	uint32_t	0
ulCmd	uint32_t	0x2CC9
tData	ECAT_DPM_GET_STATION_ALIAS_CNF_DATA_T	
usStationAlias	uint16_t	Configured Station Alias also called Second Station Address

Table 59. ECAT_DPM_GET_STATION_ALIAS_CNF_T

6.2.7 Relation between Set configuration packet and ESI file

The [Set configuration service](#) and the ESI file are strongly connected with one another. In order to clarify this, we explain for a specific ESI file, which of its parts corresponds with which part of the Set configuration packet. The following figure shows the ESI file and its relevant parts:


```

<?xml version="1.0" encoding="utf-8" ?><EtherCATInfo Version="1.11" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespace
  <Vendor FileVersion="0033">
    ① <Id>#x00000044</Id>
    ② <Name>Hilscher Gesellschaft für Systemintegration mbH</Name>
    <ImageData6x14>424DE600000000000000760000002800000010000000E0000000100080000000000E0000000120B0000120B0000EA0000000000000000
  </Vendor>
  <Descriptions>
    <Groups>
      <Group SortOrder="0">
        ⑤ <Type>netX</Type>
        <Name LcId="1033">netX</Name>
        <ImageData6x14>424DE600000000000000760000002800000010000000E00000001000400000000070000000130B00000000007000000000
      </Group>
    </Groups>
    <Devices>
      <Device Physics="YY" ⑤>
        ③ <Type ProductCode="#x0000002E" RevisionNo="#x00060004">NETX 52-RE/ECS</Type>
        <Name LcId="1033">[CDATA[NETX 52-RE/ECS]]</Name>
        <GroupType>netX</GroupType>
        <Fmmu>Outputs</Fmmu> ④
        <Fmmu>Inputs</Fmmu> ⑥
        <Fmmu>MBoxState</Fmmu>
        <Sm MinSize="128" MaxSize="128" DefaultSize="128" StartAddress="#1000" ControlByte="#x36" Enable="1">MBoxOut</Sm>
        <Sm MinSize="128" MaxSize="128" DefaultSize="128" StartAddress="#1080" ControlByte="#x32" Enable="1">MBoxIn</Sm>
        <Sm MinSize="0" MaxSize="1024" DefaultSize="200" StartAddress="#1100" ControlByte="#x74" Enable="1">Outputs</Sm>
        <Sm MinSize="0" MaxSize="1024" DefaultSize="200" StartAddress="#1D00" ControlByte="#x30" Enable="1">Inputs</Sm>
        <RxPdo Sm="2">
          <Index>#x1600</Index> ⑧
          <Name>1. RxPDO</Name> ⑦
          <Entry>
            ⑨ <Mailbox DataLinkLayer="1">
              <CoE SdoInfo="1" PdoMapping="1" CompleteAccess="1" ></CoE>
              <FoE/>
            </Mailbox>
            <Dc>
              ⑩ <OpMode>
                <Name>DcOff</Name>
                <Desc>DC unused</Desc>
                <AssignActivate>#x0000</AssignActivate>
              </OpMode>
              <OpMode>
                <Name>DcSync0</Name>
                <Desc>DC Sync0 for synchronization</Desc>
                <AssignActivate>#x0300</AssignActivate>
                <CycleTimeSync0 Factor="1">0</CycleTimeSync0>
                <ShiftTimeSync0>0</ShiftTimeSync0>
              </OpMode>
            </Dc>
            <Eeprom>
              ⑪ <ByteSize>4096</ByteSize>
              <ConfigData>060000CCE8030000</ConfigData>
              <BootStrap>0010800080108000</BootStrap>
            </Eeprom>
            ⑫ <ImageData6x14>424DE600000000000000760000002800000010000000E00000001000400000000070000000130B00000000007000000000
          </Entry>
        </RxPdo>
      </Device>
    </Devices>
  </Descriptions>
</EtherCATInfo>
    
```

Figure 18. Relation between Set Configuration packet and ESI file

The following elements of the XML-based ESI file relate with specific parameters within the configuration process:

1. Element `<Vendor><Id>`:
This element corresponds to the `ulVendorId` element of the Set Configuration packet. To be set within Set Configuration packet and changed within ESI file.
2. Element `<Vendor><Name>`:
To be changed within ESI file only.
3. Element `<Descriptions><Devices><Device><Type>`:
To be set within Set Configuration packet and changed within ESI file. Affects element contents. Take this element from structure `ECAT_SET_CONFIG_DEVICEINFO_T` (element `szOrderIdx`).
Attribute *Product code*:
This element corresponds to the `ulProductCode` element of the Set Configuration packet. Attribute *Revision number*:
This element corresponds to the `ulRevisionNumber` element of the Set Configuration packet.
4. Element `<Descriptions><Devices><Device><Name>`:
Element contents (CDATA entry)
Take this element from structure `ECAT_SET_CONFIG_DEVICEINFO_T` (element `szNameIdx`). This element corresponds to the `ul DeviceName` element of the Set Configuration packet To be set within Set Configuration packet and changed within ESI file.
5. Element `<Descriptions><Groups><Group><Type>`:
Subelement `<Type>`:
Take this element from structure `ECAT_SET_CONFIG_DEVICEINFO_T` (element `szGroupIdx`). To be set within Set Configuration packet and changed within ESI file.

Subelement *<Name>*:

To be changed within ESI file only.

Subelement *<ImageData16x14>*:

To be changed within ESI file only.

6. Element *<Descriptions><Devices><Device><GroupType>*:

Element contents To be changed within ESI file only.

7. Element *<Descriptions><Devices><Device><Sm>*:

Attribute *Default Size*

This entry is mandatory for SM0 und SM1, otherwise optional. It is automatically set, if sync mode is activated within structure *ECAT_SET_CONFIG_SMLENGTH_T*. To be set within *Set Configuration packet* and changed within ESI file.

8. Element *<Descriptions><Devices><Device><RxPdo>*:

The PDOs have to be specified manually here. Specify attributes Index, Name, Entry.

9. Element *<Descriptions><Devices><Device><Mailbox>*:

Take information from structure *ECAT_SET_CONFIG_COE_T* (for instance the element *bCoeFlags* contains the flag information such as *SdoInfo*, *PdoUpload* and *CompleteAccess*).+ To be set within *Set Configuration packet* and changed within ESI file. Affects subelement *<CoE>* and subelement *<FoE>*. ESI elements *<CoE>*, *<FoE>*, ... need to be added in correct order.

10. Element *<Descriptions><Devices><Device><Dc>*:

Take information from structure *ECAT_SET_CONFIG_SYNCMODES_T*. To be set within *Set Configuration packet* and changed within ESI file. Affects subelement *<OpMode>*.

11. Element *<Descriptions><Devices><Device><Eeprom>*:

Subelement *<ConfigData>*:

Usually, there is no need for changes here.

Subelement *<BootStrap>*:

If *Set Configuration packet* deactivates boot mailbox (i.e. *usBootstrapMbx Size =0*), remove ESI entry.

12. Element *<Descriptions><Devices><Device><ImageData16x14>*:

Element contents To be changed within ESI file only

6.3 EtherCAT state machine

Service	Command	Command Code
Set configuration service	ECAT_SET_CONFIG_REQ	0x2CCE
	ECAT_SET_CONFIG_CNF	0x2CCF
Register for AL control changed indications service	ECAT_ESM_REGISTER_FOR_ALCONTROL_INDICATIONS_REQ	0x1B18
	ECAT_ESM_REGISTER_FOR_ALCONTROL_INDICATIONS_CNF	0x1B19
Unregister from AL control changed indications service	ECAT_ESM_UNREGISTER_FROM_ALCONTROL_INDICATIONS_REQ	0x1B1A
	ECAT_ESM_UNREGISTER_FROM_ALCONTROL_INDICATIONS_CNF	0x1B1B
AL control changed service	ECAT_ESM_ALCONTROL_CHANGED_IND	0x1B1C
	ECAT_ESM_ALCONTROL_CHANGED_RES	0x1B1D
AL status changed service	ECAT_ESM_ALSTATUS_CHANGED_IND	0x19DE
	ECAT_ESM_ALSTATUS_CHANGED_RES	0x19DF
Set AL status service	ECAT_ESM_SET_ALSTATUS_REQ	0x1B48
	ECAT_ESM_SET_ALSTATUS_CNF	0x1B49
Get AL status service	ECAT_ESM_GET_ALSTATUS_REQ	0x2CD0
	ECAT_ESM_GET_ALSTATUS_CNF	0x2CD1

Table 60. Overview over the EtherCAT state machine related packets of the EtherCAT Slave stack

6.3.1 Register for AL control changed indications service

In EtherCAT, usually the master controls the state of all slaves. The master can request state changes from the slave. Each time the master requests such a state change of the EtherCAT State Machine (ESM), an indication [AL control changed indication packet](#) is received at the slave informing it about the master's state change request. Then the slave can decide on its own whether to perform or deny the state change requested by the master. However, in order to receive these indications, it is necessary that the application first has to register for the AL control changed indications service. For more information on this service, also refer to [Handling and controlling the EtherCAT State Machine](#)

6.3.1.1 Register for AL control changed indication request packet

This request has to be sent from the application to the stack in order to register for the reception of AL control changed indications signaling a state change request by the EtherCAT Master. Starting with stack version V4.3.16, this packet is extended with a data part and now supports the mechanism to activate indications for state changes from BOOT to INIT. The former packet still works for backward compatibility. This mechanism is compliant to the Semiconductor specification ETG5003-2. + After successful registration on state change requests, the ESM task of the stack will send `ECAT_ESM_ALCONTROL_CHANGED_IND_T` to the registered application.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	4
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B18
tData	ECAT_ESM_REGISTER_FOR_ALCONTROL_INDICATIONS_REQ_DATA_T	
fEnableBootToInitHandling	uint32_t	0 disables the indication mechanism, other enables values 0 - 2up32 -1

Table 61. ECAT_ESM_REGISTER_FOR_ALCONTROL_INDICATIONS_REQ_T

6.3.1.2 Register for AL control changed indications confirmation packet

This confirmation will be sent from the stack to the application. It confirms that the stack is ready to process AL control changed indications.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B19

Table 62. ECAT_ESM_REGISTER_FOR_ALCONTROL_INDICATIONS_CNF_T

6.3.2 Unregister from AL control changed indications service

This service unregisters from AL control Changed Indications. The stack will not generate AL control changed indications any more. For more information on this service, also refer to [Handling and controlling the EtherCAT State Machine](#)

6.3.2.1 Unregister from AL control changed indications request packet

This request has to be sent from the application to the stack in order to unregister from the reception of AL control changed indication. + After unregistration, on state change requests the ESM task will discontinue sending AL control changed indications to the unregistered application.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0

Variable	Type	Description
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B1A

Table 63. ECAT_ESM_UNREGISTER_FROM_ALCONTROL_INDICATIONS_REQ_T

6.3.2.2 Unregister from AL control changed indications confirmation packet

This confirmation will be sent from the stack to the application. It confirms that the stack is informed about no longer receiving AL control Changed indications.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B1B

Table 64. ECAT_ESM_UNREGISTER_FROM_ALCONTROL_INDICATIONS_CNF_T

6.3.3 AL control changed service

In EtherCAT, usually the master controls the state of all slaves. Therefore, the EtherCAT Master can request state changes from the slave. Then the slave can decide on its own whether to perform or deny the state change requested by the master.

Each time the master requests such a state change of the EtherCAT State Machine (ESM), an indication must be inform the application at the slave about the master's state change request. This is done by the AL control Changed Indication service.

For more information on this service, also refer to [Handling and controlling the EtherCAT State Machine](#)

NOTE | It is necessary to register the application by using the [Register for AL control changed indications service](#) in order to receive an AL control changed indication.

Some additional Hints on synchronisation modes You have to use the objects 0x1C32 (Sync Manager 2) or 0x1C33 (Sync Manager 3) for choosing and adjusting the synchronization mode of the EtherCAT Slave (free running, synchronized to SM2/3 event or synchronized to Distributed Clocks Sync Event), if the device supports more than freerun. For more information, see [11].

This request has to be confirmed either by the AP Task or in case of LOM by user tasks.)

6.3.3.1 AL control changed indication packet

This indication is sent by the stack when the master requests a state change of the ESM. The structure tAlControl contains AL control register dependent information.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	None
ulSta	uint32_t	0
ulCmd	uint32_t	00x1B1C
tData	ECAT_ESM_ALCONTROL_CHANGED_IND_DATA_T	
tAlControl	ECAT_ALCONTROL_T	Structure representing the AL control register ECAT_ALCONTROL_T , value 0 - 0xFFFF
usErrorLed	uint16_t	LED error state, value 0 - 8 for the current state of the error LED, meaning see chapter Error LED
usSyncControl	uint16_t	Sync Control, value 0 - 0xFFFF, PDI (sync signal) activation reflects ESC reg. 0x0980 [3].

Variable	Type	Description
usSyncImpulseLength	uint16_t	Length of Sync Impulse, value 0 - 0xFFFF (in units of 10 nanoseconds), ESC reg. 0x9A0 [3], the real cycle time has to be calculated: Cycle Time of SYNC1 = ((DcCycTime1 div DcCycTime0) + 1) * DcCycTime0. * Shift Time of SYNC1 = DcCycTime1 mod DcCycTime0)
uISync0CycleTime	uint32_t	Sync0 Cycle Time (in units of 1 nanoseconds), ESC register 0x0151 [3]
uISync1CycleTime	uint32_t	Sync1 Cycle Time (in units of 1 nanoseconds), ESC register 0x0151 [3]
bSyncPdiConfig	uint8_t	Sync PDI Configuration, value 0 - 0xFF, [3]

Table 65. ECAT_ESM_ALCONTROL_CHANGED_IND_T

The following structure *ECAT_ALCONTROL_T* is bitwise packet representing the AL control register described in the IEC 61158-6-12 norm.

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECAT_ALCONTROL_tag
{
    uint8_t uState          : 4; /*!< :4 Bits used */
    uint8_t fAcknowledge    : 1; /*!< :1 Bits used */
    uint8_t reserved       : 3; /*!< :3 Bits used */
    uint8_t bApplicationSpecific : 8; /*!< :8 Bits used */
} ECAT_ALCONTROL_T;
```

The variables in *ECAT_ALCONTROL_T* have the following meaning: The lowest four bits of the first byte of the structure contain the state which is requested by the master. Following values are possible:

Value	Name
0x01	ECAT_AL_STATE_INIT
0x02	ECAT_AL_STATE_PRE_OPERATIONAL
0x03	ECAT_AL_STATE_BOOTSTRAP_MODE
0x04	ECAT_AL_STATE_SAFE_OPERATIONAL
0x08	ECAT_AL_STATE_OPERATIONAL

Table 66. State definitions for AlControl uState

Value	State
1	Init state
2	Pre-Operational state
3	Bootstrap state
4	Safe-Operational state
8	Operational state

Table 67. Coding of EtherCAT state

The master will set the flag *fAcknowledge* to 0x01 if the state change happens because of a previous error situation of the slave. The master tries to reset this error situation with this state change. In case of a regular state change (e.g. during system Startup), the flag *fAcknowledge* will be set to 0x00. For more information regarding *fAcknowledge* see [10].

According to [10] the last bits of the structure are reserved, respectively application specific.

6.3.3.2 AL control changed response packet

This response has to be sent from the application to the stack.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
uIDest	uint32_t	0x20

Variable	Type	Description
ulLen	uint32_t	None
ulSta	uint32_t	0
ulCmd	uint32_t	00x1B1D

Table 68. ECAT_ESM_ALCONTROL_CHANGED_RES_T

6.3.4 AL status changed service

With this service the stack indicates to the application that the AL status (register 0x0130) of the EtherCAT Slave has changed. The new EtherCAT State and the change bit is indicated.

NOTE | It is necessary to register the application by *RCX_REGISTER_APP_REQ* in order to receive an AL status changed indication.

For more information on this service, also refer to section Handling and controlling the EtherCAT State Machine, especially Figure [Sequence diagram of state change with indication to application/host](#) and Figure [Sequence diagram of state change controlled by application/host with additional AL status changed indications](#).

6.3.4.1 AL status changed indication

This indication is sent to an application each time a change of AL status has happened. An Application registers for this packet via *RCX_REGISTER_APP_REQ*. The structure *ECAT_ALSTATUS_T* is quite similar to those defined in reference [10].

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	6
ulSta	uint32_t	0
ulCmd	uint32_t	00x19DE
tData	ECAT_ESM_ALSTATUS_CHANGED_IND_DATA_T	
tAlStatus	ECAT_ALSTATUS_T	Structure representing the AL status register ECAT_ALSTATUS_T
usErrorLed	uint16_t	Error LED state, meaning see section Error LED and [10]
usAlStatusCode	uint16_t	AL status Code, for listings of supported general and vendor specific AL status Codes see EcsV4_Public.h file

Table 69. ECAT_ESM_ALSTATUS_CHANGED_IND_T

The following structure *ECAT_ALSTATUS_T* is bitwise packet representing the AL status register described in the IEC 61158-6-12 norm.

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST ECAT_ALSTATUS_Ttag
{
    uint8_t uState           : 4; /*!< :4 Bits used */
    uint8_t fChange         : 1; /*!< :1 Bits used */
    uint8_t reserved        : 3; /*!< :3 Bits used */
    uint8_t bApplicationSpecific : 8; /*!< :8 Bits used */
} ECAT_ALSTATUS_T;
```

The variables in *ECAT_ALSTATUS_T* have the following meaning: The lowest four bits of the first byte of this structure are mapped to variable *uState* in the following manner:

Value	Name	Description
0x01	ECAT_AL_STATE_INIT	
0x02	ECAT_AL_STATE_PRE_OPERATIONAL	
0x03	ECAT_AL_STATE_BOOTSTRAP_MODE	

Value	Name	Description
0x04	ECAT_AL_STATE_SAFE_OPERATIONAL	
0x08	ECAT_AL_STATE_OPERATIONAL	

Table 70. State definitions for AIStatus uState

If the flag *fChange* is set to 0x01, the cause of the state change was the slave itself, which means that the state change happened without request of the master because of an error situation of the slave itself. To get more information check the *usAIStatusCode* field.

According to reference [10] the last bits of the structure are reserved, respectively application specific.

6.3.4.2 AL status changed response

This response has to be sent from the application to the stack after receiving an AL status Changed Indication.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x19DF

Table 71. ECAT_ESM_ALSTATUS_CHANGED_RES_T

6.3.5 Set AL status service

The request is used in the following cases:

1. Signaling an error to the master
 - For signaling an error to the master, the *usAIStatusCode* has to be set to the appropriate error code, see section [[_al_status_codes](#)].
2. Signaling to continue the EtherCAT state machine as reaction to an AL control changed indication
 - If it signals to continue the EtherCAT state machine as reaction to a *ECAT_ESM_ALCONTROL_CHANGED_REQ*, the *usAIStatusCode* has to be set to zero and the field *uState* in *tAIStatus* must be set to the state given in the equivalent *ECAT_ESM_ALCONTROL_CHANGED_IND* field *tAIControl.uState*.

For more information on this service, also refer to section [AL control register and AL status register](#).

6.3.5.1 Set AL status request

This request has to be sent from the application to the stack in order to trigger or request an ESM state transition.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	4
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B48
tData	ECAT_ESM_SET_ALSTATUS_REQ_DATA_T	
bAIStatus	uint8_t	AL status as formatted in EtherCAT AL status register, see uState in ECAT_ALSTATUS_T , values 1 - 4, 8 NOTE: The application does not have to set the error bit in case of a failure. If <i>usAIStatusCode</i> is used, the error is implicit.
bErrorLedState	uint8_t	Error LED states as described in section Error LED , values 1 - 8

Variable	Type	Description
usAlStatusCode	uint16_t	AL status code to set or 0 for success. For more information about the available AL status codes see the EcsV4_Public.h file or the EtherCAT specification.

Table 72. ECAT_ESM_SET_ALSTATUS_REQ_T

6.3.5.2 Set AL status confirmation packet

This confirmation will be sent from the stack to the application after a Set AL status request has been issued.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B49

Table 73. ECAT_ESM_SET_ALSTATUS_CNF_T

6.3.6 Get AL status service

This service allows to retrieve the current contents of the AL status register.

6.3.6.1 Get AL status request packet

This request has to be sent from the application to the stack in order to retrieve the current contents of the AL status register

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x2CD0

Table 74. ECAT_ESM_GET_ALSTATUS_REQ_T

6.3.6.2 Get AL status confirmation packet

The stack will send this confirmation to the application if the current contents of the AL status register have been requested.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	4
ulSta	uint32_t	0
ulCmd	uint32_t	2CD1
tData	ECAT_ESM_GET_ALSTATUS_CNF_DATA_T	
bAlStatus	uint8_t	AL status as formatted in EtherCAT AL status register, see uState in ECAT_ALSTATUS_T , values 1 - 4, 8
bErrorLedState	uint8_t	Error LED states as described in section Error LED , values 1 - 8
usAlStatusCode	uint16_t	AL status code to set or 0 for success. For more information about the available AL status codes see the EcsV4_Public.h file or the EtherCAT specification.

Table 75. ECAT_ESM_GET_ALSTATUS_CNF_T

6.4 CoE

Service	Command	Command Code
Send CoE emergency service	ECAT_COE_SEND_EMERGENCY_REQ	0x1994
	ECAT_COE_SEND_EMERGENCY_CNF	0x1995

Table 76. Overview over the CoE packets of the EtherCAT Slave stack

6.4.1 Send CoE emergency service

This service allows sending a CoE emergency mailbox message to notify about internal device errors. Since this is a one-way service, there is no defined response from the remote station. The emergency message can only be transferred if the mailbox is active (all states except Init). The station address *usStationAddress* can be used for two purposes:

- For addressing a master, it is always set to the value 0.
- For addressing a slave, additional preparations at the master are necessary. For more information on this topic, refer to the master's documentation. Set *usStationAddress* to the value that has been assigned to the respective slave to be addressed by the EtherCAT Master.

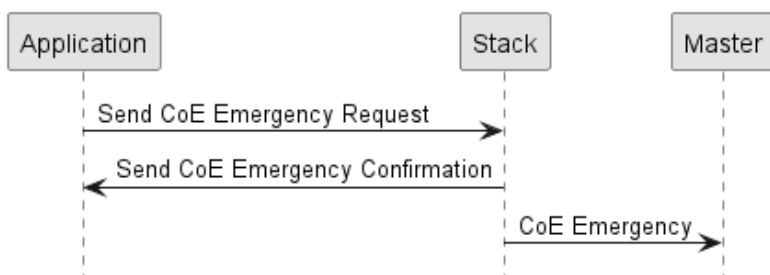


Figure 19. Send CoE emergency service

6.4.1.1 Send CoE emergency request

The application has to send this request to the EtherCAT Slave protocol stack in order to signal an emergency event within the slave to the master. For a list of possible values of *bErrorRegister* see below.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	12
ulSta	uint32_t	0
ulCmd	uint32_t	0x1994
tData	ECAT_COE_SEND_EMERGENCY_REQ_DATA_T	
usStationAddress	uint16_t	The station address is assigned to the slave by the master during ESM State Init and further on used to identify the slave.
usPriority	uint16_t	Priority of the mailbox message, value 0-3, 0 lowest, 3 highest
usErrorCode	uint16_t	Error code as defined by IEC 61158 Part 2-6 Type 12 (or ETG 1000.6). value 0 - 0xFFFF, See CoE emergency codes or Table 50 of reference [9]
bErrorRegister	uint8_t	Error register as defined by IEC 61158 Part 2-6 Type 12 (or ETG 1000.6)
abDiagnosticData[5]	uint8_t	Diagnostic Data specific to error code

Table 77. ECAT_COE_SEND_EMERGENCY_REQ_T

#	Name	Bit mask
D0	Generic error	0x0001
D1	Current error	0x0002
D2	Voltage error	0x0004
D3	Temperature error	0x0008
D4	Communication error	0x0010
D5	Device profile specific error	0x0020

D6	Reserved	0x0040
D7	Manufacturer specific error	0x0080

 Table 78. Bit Mask *bErrorRegister*

The following rules apply for the relationship between *usErrorCode*, *bErrorRegister* and *abDiagnosticData*:

1. At error codes (hexadecimal values) *10xx* bit D0 (Generic error) of bit mask *bErrorRegister* should be set, otherwise reset.
2. At error codes (hexadecimal values) *2xxx* bit D1 (Current error) of bit mask *bErrorRegister* should be set, otherwise reset.
3. At error codes (hexadecimal values) *3xxx* bit D2 (Voltage error) of bit mask *bErrorRegister* should be set, otherwise reset.
4. At error codes (hexadecimal values) *4xxx* bit D3 (Temperature error) of bit mask *bErrorRegister* should be set, otherwise reset.
5. At error codes (hexadecimal values) *81xx* bit D4 (Communication error) of bit mask *bErrorRegister* should be set, otherwise reset.

The relationship between *usErrorCode*, *bErrorRegister* and *abDiagnosticData* may also depend on the used profile.

6.4.1.2 Send CoE emergency confirmation packet

The stack will send this confirmation packet to the application on reception of a CoE emergency request.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	12
ulSta	uint32_t	0
ulCmd	uint32_t	0x1995

Table 79. ECAT_COE_SEND_EMERGENCY_CNF_T

6.5 Packets for Object Dictionary access

All packets for object dictionary access are described in [4]

6.6 Slave Information Interface (SII in virtual EEPROM)

Service	Command	Command Code
SII read service	ECAT_ESM_SII_READ_REQ	0x1914
	ECAT_ESM_SII_READ_CNF	0x1915
SII write service	ECAT_ESM_SII_WRITE_REQ	0x1912
	ECAT_ESM_SII_WRITE_CNF	0x1913
Register for SII write Indications service	ECAT_ESM_REGISTER_FOR_SIIWRITE_INDICATIONS_REQ	0x1B82
	ECAT_ESM_REGISTER_FOR_SIIWRITE_INDICATIONS_CNF	0x1B83
Unregister from SII write indications service	ECAT_ESM_UNREGISTER_FROM_SIIWRITE_INDICATIONS_REQ	0x1B84
	ECAT_ESM_UNREGISTER_FROM_SIIWRITE_INDICATIONS_CNF	0x1B85
SII write Indication service	ECAT_ESM_SII_WRITE_IND	0x1B80
	ECAT_ESM_SII_WRITE_REQ	0x1B81

Table 80. Overview over the SII packets of the EtherCAT Slave stack

6.6.1 SII read service

The service is used for reading information that has been stored in the Slave Information Interface (SII) of the device. The SII holds information about the slave, which the master needs for administrative purposes. For more details, also see [Slave Information Interface \(SII\)](#) in the stack structure chapter.

6.6.1.1 SII read request packet

This packet performs an SII read request. A data block of the size $u/Size$ ($= n$) is read from the location with the specified offset $u/Offset$ and is returned with the confirmation packet.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	8
ulSta	uint32_t	0
ulCmd	uint32_t	0x1914
tData	ECAT_ESM_SII_READ_REQ_DATA_T	
ulOffset	uint32_t	Offset value (byte address within the SII image)
ulSize	uint32_t	Size of data block to read

Table 81. ECAT_ESM_SII_READ_REQ_T

6.6.1.2 SII read confirmation packet

The stack will send this confirmation packet to the application on reception of an SII read request packet.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	None
ulSta	uint32_t	0
ulCmd	uint32_t	0x1915
tData	ECAT_ESM_SII_READ_CNF_DATA_T	
abData[ECAT_ESM_SII_READ_DATA_BYTESIZE]	uint8_t	Field for read data

Table 82. ECAT_ESM_SII_READ_CNF_T

6.6.2 SII write service

The service is used for sending information to be stored in the Slave Information Interface (SII) of the device. The SII holds information about the slave, which the master needs for administrative purposes. For more details, also see [Slave Information Interface \(SII\)](#) in the stack structure chapter.

6.6.2.1 SII write request packet

This packet performs an SII write request.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	4 + n
ulSta	uint32_t	0
ulCmd	uint32_t	0x1912
tData	ECAT_ESM_SII_WRITE_REQ_DATA_T	
ulOffset	uint32_t	Offset value (byte address within the SII image)

Variable	Type	Description
abData[ECAT_ESM_SII_WRITE_DATA_BYTESIZE]	uint8_t	Data to be written

Table 83. ECAT_ESM_SII_WRITE_REQ_T

6.6.2.2 SII write confirmation packet

The stack will send this confirmation packet to the application on reception of an SII write request packet.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	4 + n
ulSta	uint32_t	0
ulCmd	uint32_t	0x1913

Table 84. ECAT_ESM_SII_WRITE_CNF_T

6.6.3 Register for SII write Indications service

The application has to register on the EtherCAT Slave protocol stack in order to receive indications when the EtherCAT master writes to the SII.

6.6.3.1 Register for SII write Indications request packet

The application has to send this request to the EtherCAT Slave protocol stack in order to register for SII write indications

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	4
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B82
tData	ECAT_ESM_REGISTER_FOR_SIIWRITE_INDICATIONS_REQ_DATA_T	
ulIndicationFlags	uint32_t	Indication flags

Table 85. ECAT_ESM_REGISTER_FOR_SIIWRITE_INDICATIONS_REQ_T

ulIndicationFlags

Actually there is only one filter flag defined:

ECAT_ESM_FILTER_SIIWRITE_INDICATIONS_STATION_ALIAS: In the past, the application had to use several packets in order to set station alias address. Bit 0 of the variable *ulIndicationFlags* is set to 1 (define *ECAT_ESM_FILTER_SIIWRITE_INDICATIONS_STATION_ALIAS*) an application received only an SII write indication, if the station alias has been written from the master. Other write accesses will not lead to an SII write indication. If not set, every write access leads to an indication. The filter *ECAT_ESM_FILTER_SIIWRITE_INDICATIONS_STATION_ALIAS* was mainly intended helping to implement the remanent saving of the station alias address from application side.

Now the EtherCAT Slave stack executes the address handling concerning the station alias. Starting with version 4.5 (starting with version 4.6 for cifX cards). To use this filter function is no longer necessary for the application, except for netX52 devices.

This section relates to section [Set Station Alias service](#).

6.6.3.2 Register for SII write confirmation packet

The stack will send this confirmation packet to the application on reception of a [ECAT_ESM_REGISTER_FOR_SIIWRITE_INDICATIONS_REQ](#) packet.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B83

Table 86. ECAT_ESM_REGISTER_FOR_SIIWRITE_INDICATIONS_CNF_T

6.6.4 Unregister from SII write indications service

This service is used to unregister from getting indications, which occur when the EtherCAT Master writes to the SII.

6.6.4.1 Unregister from SII write indications request

This request has to be sent from the application to the stack in order to unregister from sii write indications.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B84

Table 87. ECAT_ESM_UNREGISTER_FROM_SIIWRITE_INDICATIONS_REQ_T

6.6.4.2 Unregister from SII write confirmation packet

The stack will send this confirmation packet to the application on reception of an *ECAT_ESM_UNREGISTER_FROM_SIIWRITE_INDICATIONS_REQ_T* packet.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B85

Table 88. ECAT_ESM_UNREGISTER_FROM_SIIWRITE_INDICATIONS_CNF_T

6.6.5 SII write Indication service

The service indicates that the master has written to the SII.

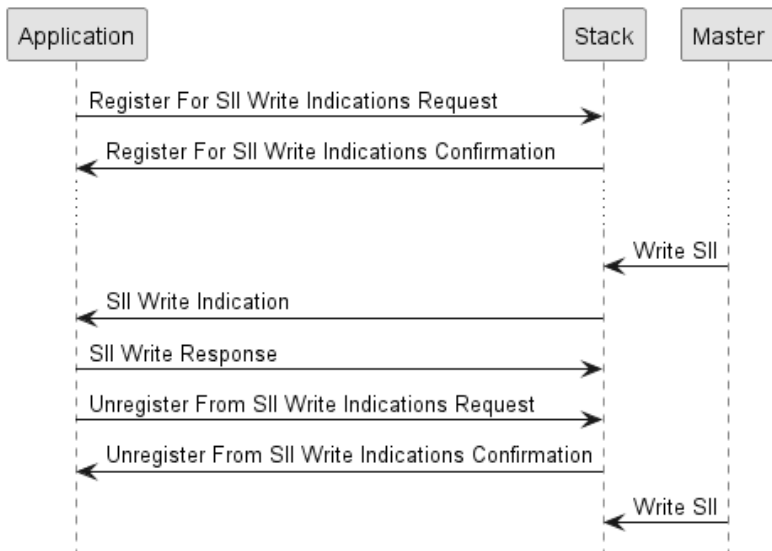


Figure 20. SII write Indication service

Permanent SII EEPROM storage

If the AP task requires implementing permanent SII EEPROM storage, it is possible to react on an SII write indication with a SII read request. This allows storing the SII image in any kind of permanent storage on the host side. The stored data can be written back on power up to the SII image with the SII write Request.

NOTE | It is necessary to register the application by using the Register for SII write Indications Request in order to receive an SII write Indication

6.6.5.1 SII write indication packet

The stack sends this indication to the application when the EtherCAT Master has written data to the SII.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	6
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B80
tData	ECAT_ESM_SII_WRITE_IND_DATA_T	
ulSiiWriteStartAddresses	uint32_t	Address to which was written in SII
abData[2]	uint8_t	Data which was written to SII

Table 89. ECAT_ESM_SII_WRITE_IND_T

6.6.5.2 SII write response packet

The application has to send this response to the protocol stack on reception of an SII write indication packet.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B81

Table 90. ECAT_ESM_SII_WRITE_RES_T

6.7 Ethernet over EtherCAT (EoE)

Service	Command	Command Code
Register for frame indications service	ECAT_EOE_REGISTER_FOR_FRAME_INDICATIONS_REQ	0x1B76
	ECAT_EOE_REGISTER_FOR_FRAME_INDICATIONS_CNF	0x1B77
Unregister from frame indications service	ECAT_EOE_UNREGISTER_FROM_FRAME_INDICATIONS_REQ	0x1B78
	ECAT_EOE_UNREGISTER_FROM_FRAME_INDICATIONS_CNF	0x1B79
Ethernet send frame service	ECAT_EOE_SEND_FRAME_REQ	0x1B72
	ECAT_EOE_SEND_FRAME_CNF	0x1B73
Ethernet frame received service	ECAT_EOE_FRAME_RECEIVED_IND	0x1B70
	ECAT_EOE_FRAME_RECEIVED_RES	0x1B71
Register for IP parameter indications service	ECAT_EOE_REGISTER_FOR_IP_PARAM_INDICATIONS_REQ	0x1B7A
	ECAT_EOE_REGISTER_FOR_IP_PARAM_INDICATIONS_CNF	0x1B7B
Unregister from IP parameter Indications service	ECAT_EOE_UNREGISTER_FROM_IP_PARAM_INDICATIONS_REQ	0x1B7C
	ECAT_EOE_UNREGISTER_FROM_IP_PARAM_INDICATIONS_CNF	0x1B7D
Set IP parameter service	ECAT_EOE_SET_IP_PARAM_IND	0x1B7E
	ECAT_EOE_SET_IP_PARAM_RES	0x1B7F
Get IP parameter service	ECAT_EOE_GET_IP_PARAM_IND	0x1B50
	ECAT_EOE_GET_IP_PARAM_RES	0x1B51

Table 91. Overview over the EoE Packets of the EtherCAT Slave Stack

EoE is a tunnel protocol which is tunneled via the EtherCAT mailbox for Ethernet frames. All EoE communication is passed through the master. There is no direct communication path. This causes the achievable bandwidth to be largely decreased compared to the actual bandwidth on the cable.

EoE requires the EtherCAT Slave stack to be at least in Pre-Operational state in order to be able to communicate via the EtherCAT mailbox.

It is also necessary that the EtherCAT Master supports EoE since all tunneled Ethernet frames are transported through the master. The master will typically assign one of the following values depending on the EoE section within the mailbox section of the EtherCAT Slave Information (ESI) file:

- MAC address
- IP address

Example of a mailbox section within the ESI enabling IP and MAC address assignment

```
<Mailbox>
  <EoE IP="1" MAC="1"/> <!-- EoE supported and IP and MAC assignment selected -->
  <CoE SdoInfo="1" CompleteAccess="0"/>
</Mailbox>
```

This will result into an IP parameter Written By Master indication if the application has registered for receiving this indication.

NOTE | The EoE service is only responsible for the tunneling of Ethernet frames. Transport layers like TCP or UDP have to be added by the user on application side. Only the targets for netX52 (Socket Api), netx51TCP (Packet Api) and netRAPID51 (Packet Api) includes the transport layers TCP or UDP via channel 1. For information on the interfaces see manuals socket interface [5] and packet interface [6]

6.7.1 Register for frame indications service

This service enables the application to receive Ethernet frame indications from the protocol stack.

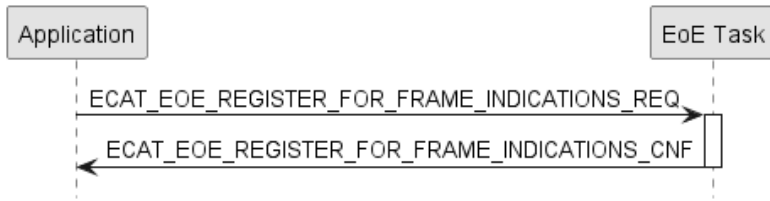


Figure 21. Sequence diagram for ECAT_EOE_REGISTER_FOR_FRAME_INDICATIONS_REQ/CNF packets

NOTE This service should not be used if the EthIntf is mapped to the second channel or if direct access via Drv_Edd within LOM is used or when a TCP stack is included in the firmware. Also, this service should not be used if the Lwip stack or the Socket API are included within the firmware. If you nevertheless use it, the LwIP stack or TCP functionality might not work correctly.

6.7.1.1 Register for frame indications request packet

The application has to send this request packet to the EtherCAT Slave protocol stack in order to register itself at the EtherCAT EoE stack for receiving indications (*ECAT_EOE_FRAME_RECEIVED_IND* packets) each time an EoE Ethernet frame is received by the EtherCAT EoE stack.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B76

Table 92. ECAT_EOE_REGISTER_FOR_FRAME_INDICATIONS_REQ_T

6.7.1.2 Register for frame indications confirmation packet

The stack will send this confirmation packet to the application after registering for receiving Ethernet frame indications.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B77

Table 93. ECAT_EOE_REGISTER_FOR_FRAME_INDICATIONS_CNF_T

6.7.2 Unregister from frame indications service

This service disables the application from receiving Ethernet frame indications.

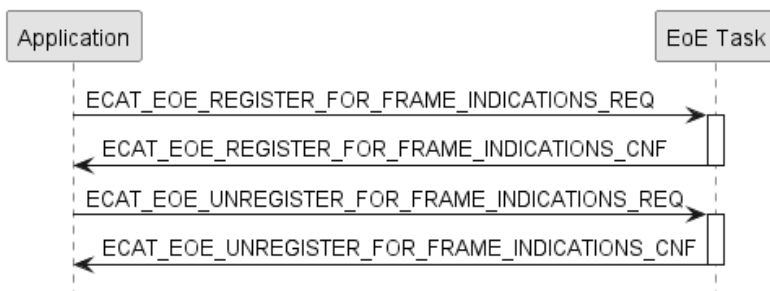


Figure 22. Sequence diagram for ECAT_EOE_UNREGISTER_FROM_FRAME_INDICATIONS_REQ/CNF packets

NOTE | This service should not be used if the EthIntf is mapped to the second channel or if direct access via *Drv_Edd* within LOM is used.

6.7.2.1 Unregister from frame indications request packet

The stack will send this confirmation packet to the application after unregistering from receiving Ethernet frame indications.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B79

Table 94. ECAT_EOE_UNREGISTER_FROM_FRAME_INDICATIONS_REQ_T

6.7.2.2 Unregister from frame indications confirmation packet

The application has to send this request packet to the EtherCAT Slave protocol stack in order to disable exception of Ethernet frame indications.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B78

Table 95. ECAT_EOE_UNREGISTER_FROM_FRAME_INDICATIONS_CNF_T

6.7.3 Ethernet send frame service

This service allows sending Ethernet frames via EoE.

NOTE | This service should not be used if the EthIntf is mapped to the second channel or if direct access via *Drv_Edd* within LOM is used.

The parameter *usFlags* is a bit mask which is used to specify whether some fields within the current packet is valid. Currently the following bits are defined:

Bit	Name	Description
D2-D15	Reserved	
D1	<i>ECAT_EOE_FRAME_FLAG_TIME_VALID</i>	The timestamp in the current packet is valid.
D0	<i>ECAT_EOE_FRAME_FLAG_TIME_REQUEST</i>	On requests, the master requests the actual transmission time of the frame when it is sent on the slave itself

Table 96. Meaning of bit mask *usFlags*

6.7.3.1 Ethernet send frame request packet

The *ECAT_EOE_SEND_FRAME_REQ* request allows your application to send Ethernet frames via EoE. Use the field *abData* to store the contents of the Ethernet frame to be sent.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	22+n
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B72

Variable	Type	Description
tData	ECAT_EOE_SEND_FRAME_REQ_DATA_T	
usFlags	uint16_t	bit mask specifying whether some fields within the current packet are valid see bit mask usFlags
usPortNo	uint16_t	determines the specific port to be used. port value range 1 to 15, value 0 means no specific port is used.
ulTimestampNs	uint32_t	timestamp based on the EtherCAT system time
abDstMacAddr[6]	uint8_t	destination MAC address of the frame to be sent through EoE from the slave
abSrcMacAddr[6]	uint8_t	source MAC address of frame received to be sent through EoE from the slave, refers to the origin of the Ethernet frame.
usEthType	uint16_t	Ethernet type of the EoE frame to be sent (in network byte order)
abData[ECAT_EOE_FRAME_DATA_SIZE]	uint8_t	field containing the data of the Ethernet frame (1504 bytes)

Table 97. ECAT_EOE_SEND_FRAME_REQ_T

6.7.3.2 Ethernet send frame confirmation packet

The stack will send this confirmation packet to the application after receiving an `ECAT_EOE_SEND_FRAME_REQ` request.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	8
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B73
tData	ECAT_EOE_SEND_FRAME_CNF_DATA_T	
usFlags	uint16_t	Flags, see table Meaning of bit mask usFlags
ulTimestampNs	uint32_t	EtherCAT system time of frame being received at destination, only valid if <code>ECAT_EOE_FRAME_FLAG_TIME_VALID</code> is set in usFlags.
usFrameLen	uint16_t	

Table 98. ECAT_EOE_SEND_FRAME_CNF_T

6.7.4 Ethernet frame received service

This indication will be sent to your application if both of the following conditions are fulfilled:

1. The application has registered for it by sending an `ECAT_EOE_REGISTER_FOR_FRAME_INDICATIONS_REQ` request to the stack.
2. A new Ethernet frame is received via EoE.

The contents of the Ethernet frame can be retrieved by accessing the field `abData`.

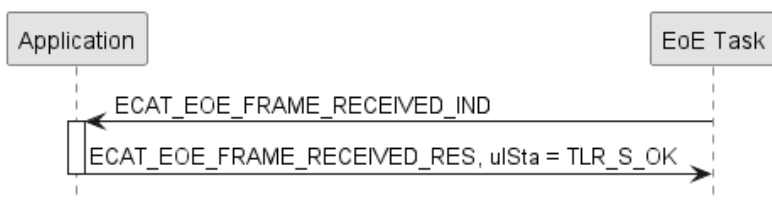


Figure 23. Sequence diagram EoE frame reception

NOTE It is necessary to register the application by using the Register for frame indications service in order to receive an Ethernet frame received indication.

NOTE This service should not be used if the `EthIntf` is mapped to the second channel or if direct access via `Drv_Edd` within LOM is used.

6.7.4.1 Ethernet frame received indication packet

The stack will send this indication packet to the application.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	22+n
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B70
tData	ECAT_EOE_FRAME_RECEIVED_IND_DATA_T	
usFlags	uint16_t	bit mask specifying whether some fields within the current packet are valid Meaning of bit mask usFlags
usPortNo	uint16_t	determines the specific port to be used. port value range 1 to 15, value 0 means no specific port is used.
ulTimestampNs	uint32_t	timestamp based on the EtherCAT system time
abDstMacAddr[6]	uint8_t	destination MAC address of the frame to be sent through EoE from the slave
abSrcMacAddr[6]	uint8_t	source MAC address of frame received to be sent through EoE from the slave, refers to the origin of the Ethernet frame.
usEthType	uint16_t	Ethernet type of the EoE frame to be sent (in network byte order)
abData[ECAT_EOE_FRAME_DATA_SIZE]	uint8_t	field containing the data of the Ethernet frame (1504 bytes)

Table 99. ECAT_EOE_FRAME_RECEIVED_IND_T

6.7.4.2 Ethernet frame received response packet

The application has to send this response packet to the protocol stack after receiving an *ECAT_EOE_FRAME_RECEIVED_IND* indication packet.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	8
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B71
tData	ECAT_EOE_FRAME_RECEIVED_RES_DATA_T	
usFlags	uint16_t	Flags, see table Meaning of bit mask usFlags
ulTimestampNs	uint32_t	EtherCAT system time of frame being received at destination, only valid if <i>ECAT_EOE_FRAME_FLAG_TIME_VALID</i> is set in usFlags.
usFrameLen	uint16_t	

Table 100. ECAT_EOE_FRAME_RECEIVED_RES_T

6.7.5 Register for IP parameter indications service

This service is used for registering an application for receiving the following indications:

- Set IP parameter service
- Get IP parameter service

NOTE | Only one application can use this service.

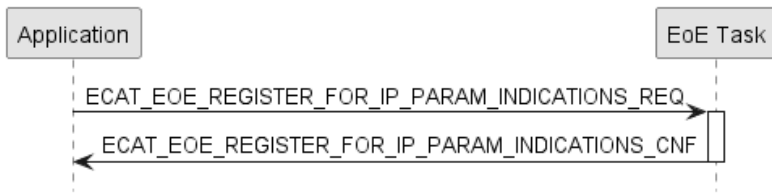


Figure 24. Sequence diagram for ECAT_EOE_REGISTER_FOR_IP_PARAM_INDICATIONS_REQ/CNF

6.7.5.1 Register for IP parameter indications request

The application can register at the notify queue for receiving indications (*ECAT_EOE_SET_IP_PARAM_IND* and *ECAT_EOE_GET_IP_PARAM_IND* packets) each time the master requests to change IP or MAC address parameters.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B7A

Table 101. ECAT_EOE_REGISTER_FOR_IP_PARAM_INDICATIONS_REQ_T

6.7.5.2 Register for IP parameter indications confirmation

The stack will send this confirmation packet to the application after registering for IP parameter indications

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B7B

Table 102. ECAT_EOE_REGISTER_FOR_IP_PARAM_INDICATIONS_CNF_T

6.7.6 Unregister from IP parameter Indications service

This service is used for registering an application for receiving the following indications:

- Set IP parameter service
- Get IP parameter service

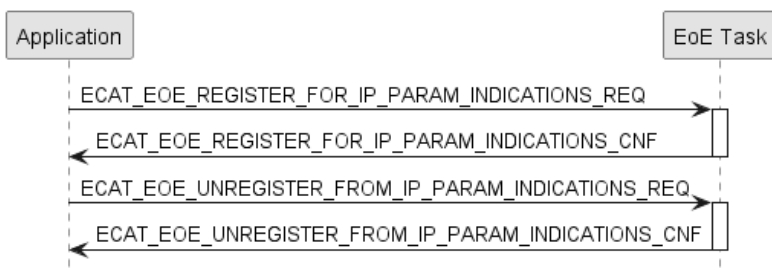


Figure 25. Sequence diagram for ECAT_EOE_UNREGISTER_FOR_IP_PARAM_INDICATIONS_REQ/CNF

6.7.6.1 Unregister from IP parameter indications request packet

The application can unregister at the queue from the reception of indications (*ECAT_EOE_SET_IP_PARAM_IND* packets) each time the master requests to change IP or MAC address parameters.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	

Variable	Type	Description
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B7C

Table 103. ECAT_EOE_UNREGISTER_FROM_IP_PARAM_INDICATIONS_REQ_T

6.7.6.2 Unregister from IP parameter indications confirmation packet

The stack will send this confirmation packet to the application after unregistering from receiving IP parameter indications.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B7D

Table 104. ECAT_EOE_UNREGISTER_FROM_IP_PARAM_INDICATIONS_CNF_T

6.7.7 Set IP parameter service

This service is used for indicating that the EtherCAT master intends to set new IP/MAC parameters. In order to receive Set IP parameter Indications, the following requirements have to be fulfilled:

- It is necessary to register the application by using the [Register for IP parameter indications service](#) in order to receive an IP parameter Written By Master indication.
- The EtherCAT Slave stack is at least in Pre-Operational state.
- The master currently intends to set new IP/MAC parameters.

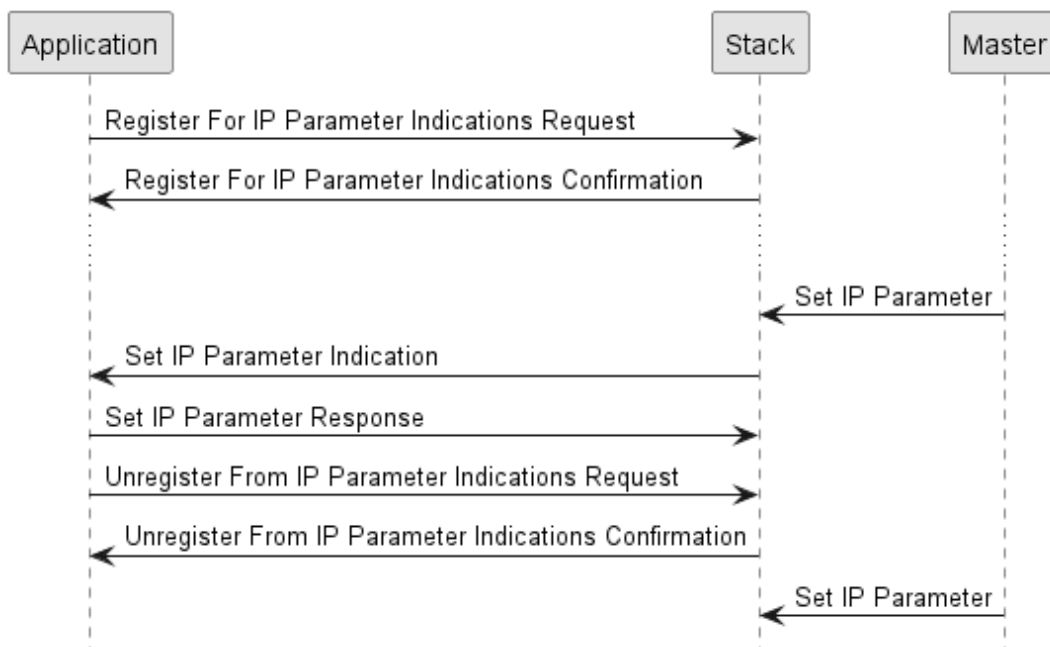


Figure 26. Set IP parameter service

The parameter *ulFlags* is a bit mask which is used to specify which fields within the packets are valid. Currently the following bits are defined:

Value	Name	Description
0x00000001	ECAT_EOE_SET_IP_PARAM_MAC_ADDRESS_INCLUDED	If set, field <i>abMacAddr</i> provides a valid MAC address.
0x00000002	ECAT_EOE_SET_IP_PARAM_IP_ADDRESS_INCLUDED	If set, field <i>abIpAddr</i> provides a valid IP address.
0x00000004	ECAT_EOE_SET_IP_PARAM_SUBNET_MASK_INCLUDED	If set, field <i>abSubnetMask</i> provides a valid subnet mask.
0x00000008	ECAT_EOE_SET_IP_PARAM_DEFAULT_GATEWAY_INCLUDED	If set, field <i>abDefaultGateway</i> provides a valid default gateway
0x00000010	ECAT_EOE_SET_IP_PARAM_DNS_SERVER_IP_ADDR_INCLUDED	If set, field <i>abDnsServerIpAddress</i> provides a valid DNS Server IP Address
0x00000020	ECAT_EOE_SET_IP_PARAM_DNS_NAME_INCLUDED	If set, field <i>abDnsName</i> provides a valid DNS name

 Table 105. Bitmask for parameter *ulFlag* of *ECAT_EOE_SET_IP_PARAM_IND_DATA_T*

6.7.7.1 Set IP parameter indication packet

These are the IP parameters written by master. This indication will be sent to your application if both of the following conditions are fulfilled:

1. The application has registered for it by sending a *ECAT_EOE_REGISTER_FOR_IP_PARAM_INDICATIONS_REQ* packet to the stack
2. The EtherCAT Master intends to set new IP/MAC parameters (and has sent an according request to the EtherCAT Slave) The values are stored in IP network byte order.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
<i>ulDest</i>	<i>uint32_t</i>	0x20
<i>ulLen</i>	<i>uint32_t</i>	0
<i>ulSta</i>	<i>uint32_t</i>	0
<i>ulCmd</i>	<i>uint32_t</i>	0x1B7E
tData	ECAT_EOE_SET_IP_PARAM_IND_DATA_T	
<i>ulFlags</i>	<i>uint32_t</i>	The single bits determine which of the subsequent fields are valid, see bit mask <i>ulFlags</i>
<i>abMacAddr</i> [6]	<i>uint8_t</i>	contains the MAC address to be set - only valid if flag is set: <i>ECAT_EOE_SET_IP_PARAM_MAC_ADDRESS_INCLUDED</i>
<i>abIpAddr</i> [4]	<i>uint8_t</i>	contains the IP address to be set - only valid if flag is set: <i>ECAT_EOE_SET_IP_PARAM_IP_ADDRESS_INCLUDED</i>
<i>abSubnetMask</i> [4]	<i>uint8_t</i>	contains the subnet mask to be set - only valid if flag is set: <i>ECAT_EOE_SET_IP_PARAM_SUBNET_MASK_INCLUDED</i>
<i>abDefaultGateway</i> [4]	<i>uint8_t</i>	contains the default gateway to be set - only valid if flag is set: <i>ECAT_EOE_SET_IP_PARAM_DEFAULT_GATEWAY_INCLUDED</i>
<i>abDnsServerIpAddress</i> [4]	<i>uint8_t</i>	contains the default gateway to be set - only valid if flag is set: <i>ECAT_EOE_SET_IP_PARAM_DNS_SERVER_IP_ADDR_INCLUDED</i> s
<i>abDnsName</i> [32]	<i>char</i>	contains the DNS name to be set - only valid if flag is set: <i>ECAT_EOE_SET_IP_PARAM_DNS_NAME_INCLUDED</i>

 Table 106. *ECAT_EOE_SET_IP_PARAM_IND_T*

6.7.7.2 Set IP parameter response packet

The application has to send this response packet to the protocol stack on reception of an IP parameter indication. The response packet does not have any parameters.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B7F

Table 107. ECAT_EOE_SET_IP_PARAM_RES_T

6.7.8 Get IP parameter service

This service is used for indicating that the master wants to retrieve the current IP/MAC parameters. In order to receive Get IP parameter Indications, the following requirements have to be fulfilled:

- It is necessary to register the application by using the [Register for IP parameter indications service](#) in order to receive an IP parameter Written By Master indication.
- The EtherCAT Slave stack is at least in Pre-Operational state.

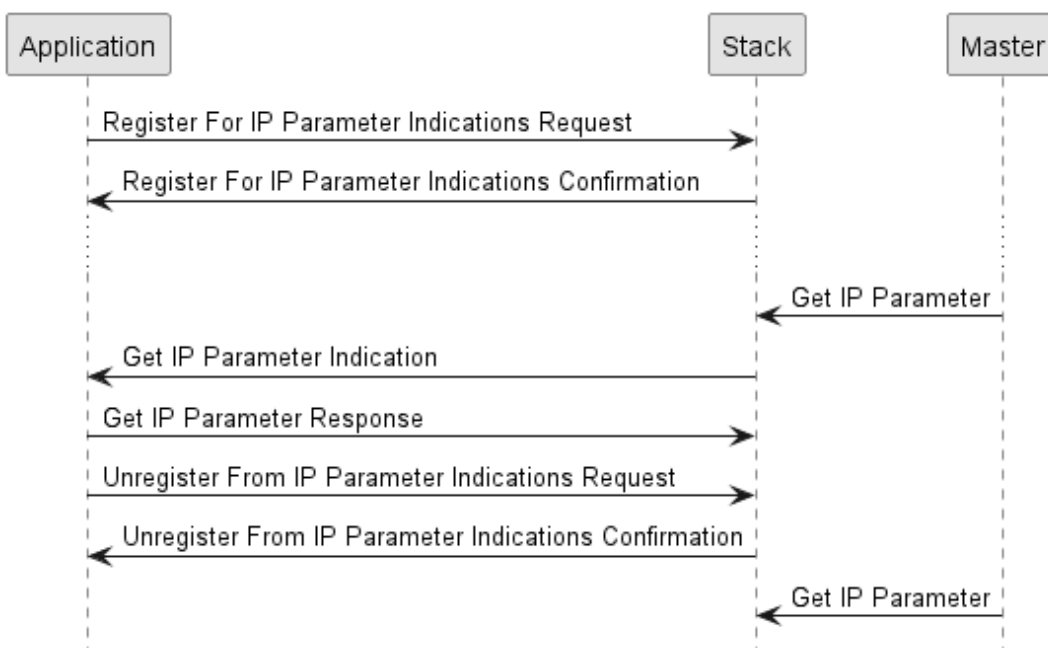


Figure 27. Get IP parameter service

The parameter *ulFlags* is a bit mask which is used to specify which fields within the packets are valid. Currently the following bits are defined:

Value	Name	Description
0x00000001	ECAT_EOE_GET_IP_PARAM_MAC_ADDRESS_INCLUDED	If set, field <i>abMacAddr</i> provides a valid MAC address.
0x00000002	ECAT_EOE_GET_IP_PARAM_IP_ADDRESS_INCLUDED	If set, field <i>abIpAddr</i> provides a valid IP address.
0x00000004	ECAT_EOE_GET_IP_PARAM_SUBNET_MASK_INCLUDED	If set, field <i>abSubnetMask</i> provides a valid subnet mask.
0x00000008	ECAT_EOE_GET_IP_PARAM_DEFAULT_GATEWAY_INCLUDED	If set, field <i>abDefaultGateway</i> provides a valid default gateway
0x00000010	ECAT_EOE_GET_IP_PARAM_DNS_SERVER_IP_ADDR_INCLUDED	If set, field <i>abDnsServerIpAddress</i> provides a valid DNS Server IP Address
0x00000020	ECAT_EOE_GET_IP_PARAM_DNS_NAME_INCLUDED	If set, field <i>abDnsName</i> provides a valid DNS name

Table 108. Bitmask for parameter ulFlag of ECAT_EOE_GET_IP_PARAM_IND_DATA_T

6.7.8.1 Get IP parameter indication packet

This packet indicates that the master wants to retrieve the current IP/MAC parameters. For receiving the indication, the application has to register via the request. The indication packet does not have any parameters:

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B50

Table 109. ECAT_EOE_GET_IP_PARAM_IND_T

6.7.8.2 Get IP parameter response packet

The application has to send this response packet to the protocol stack on reception of the parameter read by master indication. This response has to be sent from the application to the stack.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	58
ulSta	uint32_t	0
ulCmd	uint32_t	0x1B51
tData	ECAT_EOE_GET_IP_PARAM_RES_DATA_T	
ulFlags	uint32_t	The single bits determine which of the subsequent fields are valid, see bit mask ulFlags
abMacAddr[6]	uint8_t	contains the MAC address to be set - only valid if following flag is set: ECAT_EOE_SET_IP_PARAM_MAC_ADDRESS_INCLUDED
abIpAddr[4]	uint8_t	contains the IP address to be set - only valid if following flag is set: ECAT_EOE_SET_IP_PARAM_IP_ADDRESS_INCLUDED
abSubnetMask[4]	uint8_t	contains the subnet mask to be set - only valid if following flag is set: ECAT_EOE_SET_IP_PARAM_SUBNET_MASK_INCLUDED
abDefaultGateway[4]	uint8_t	contains the default gateway to be set - only valid if following flag is set: ECAT_EOE_SET_IP_PARAM_DEFAULT_GATEWAY_INCLUDED
abDnsServerIpAddress[4]	uint8_t	contains the default gateway to be set - only valid if following flag is set: ECAT_EOE_SET_IP_PARAM_DNS_SERVER_IP_ADDR_INCLUDED s
abDnsName[32]	char	contains the DNS name to be set - only valid if following flag is set: ECAT_EOE_SET_IP_PARAM_DNS_NAME_INCLUDED

Table 110. ECAT_EOE_GET_IP_PARAM_RES_T

6.8 File Access over EtherCAT (FoE)

Service	Command	Command Code
Set FoE options service	ECAT_FOE_SET_OPTIONS_REQ	0x1BD6
	ECAT_FOE_SET_OPTIONS_CNF	0x1BD7
FoE register file service	ECAT_FOE_REGISTER_FILE_INDICATIONS_REQ	0x9500
	ECAT_FOE_REGISTER_FILE_INDICATIONS_CNF	0x9501
FoE unregister file service	ECAT_FOE_UNREGISTER_FILE_INDICATIONS_REQ	0x9502
	ECAT_FOE_UNREGISTER_FILE_INDICATIONS_CNF	0x9503
FoE write file service	ECAT_FOE_WRITE_FILE_IND	0x9510
	ECAT_FOE_WRITE_FILE_RES	0x9511

Service	Command	Command Code
FoE read file service	ECAT_FOE_READ_FILE_IND	0x9512
	ECAT_FOE_READ_FILE_RES	0x9513
FoE file written Service	ECAT_FOE_FILE_WRITTEN_IND	0x9520
	ECAT_FOE_FILE_WRITTEN_RES	0x9521
FoE File Write Aborted Service	ECAT_FOE_FILE_WRITE_ABORTED_IND	0x9530
	ECAT_FOE_FILE_WRITE_ABORTED_RES	0x9531

Table 111. Overview over the FoE Packets of the EtherCAT Slave Stack

FoE and similar file operations only in states without active data exchange

FoE and other file operations on the local file system (FAT) may only be performed when the EtherCAT Slave stack is in states without active data exchange. This means:

NOTE | Only use file transfer (FoE), if the EtherCAT Slave device is either in BOOT state or PreOP state.

The file system of the EtherCAT Slave stack is intended for storing firmware and configuration files only. Do not store any other information there.

6.8.1 Set FoE options service

The service is used to define restrictions in file download via FoE. For instance, the firmware download can be rejected in case of not matching protocol class or communication class. Options request does not work on virtual files (see [FoE register file service](#)).

Value	Name	Description
0x00000001	ECAT_FOE_SET_OPTIONS_CHECK_PROTOCOL_CLASS	If set, downloads with mismatching protocol class will be rejected. Example: protocol class != EtherCAT
0x00000002	ECAT_FOE_SET_OPTIONS_CHECK_COMMUNICATION_CLASS	If set, downloads with mismatching communication class will be rejected. Example: comm class != Slave
0x00000004	ECAT_FOE_SET_OPTIONS_REJECT_NON_NXF_FILE_DOWNLOADS	If set, other file downloads than *.nxf file downloads will be rejected.
0x00000008	ECAT_FOE_SET_OPTIONS_CHECK_VARIANT	If set, downloads with mismatching variant will be rejected. Example: tDeviceInfo.usReserved != usExpectedBuildDeviceVariant
0x00000010	ECAT_FOE_SET_OPTIONS_CHECK_DEVICE_CLASS	If set, downloads with mismatching device class will be rejected. Example: device class != netX 500

Table 112. Filter Flags for ECAT_FOE_SET_OPTIONS_REQ_DATA_T

6.8.1.1 Set FoE options request packet

The service is used to define restrictions in filedownload.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	6
ulSta	uint32_t	0
ulCmd	uint32_t	0x1BD6
tData	ECAT_FOE_SET_OPTIONS_REQ_DATA_T	
ulOptions	uint32_t	Options for restricting file transfer, see ulOptions Flags
usExpectedBuildDeviceVariant	uint16_t	Expected device variant for use of customer devices

Table 113. ECAT_FOE_SET_OPTIONS_REQ_T

6.8.1.2 Set FoE options confirmation packet

It confirms that the settings for file download have been changed. The confirmation packet mirrors the data from request.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1BD7
tData	ECAT_FOE_SET_OPTIONS_CNF_DATA_T	
ulOptions	uint32_t	Options for restricting file transfer, see ulOptions Flags
usExpectedBuildDeviceVariant	uint16_t	Expected device variant for use of customer devices

Table 114. ECAT_FOE_SET_OPTIONS_CNF_T

6.8.2 FoE register file service

The application has to send the request packet to the EtherCAT Slave protocol stack in order to register for indications which occur when a file operation (up- or download) is initiated from the EtherCAT Master side. Depending on the value *blndicationType*, the application gets notifications for different events.

blndicationType is a value allowing to set the registration type of the registered file as follows:

Value	Name	Description
1	ECAT_FOE_INDICATION_TYPE_FILE_WRITTEN	If set, the stack sends an indication to the application if the file with the registered name was successfully written to the file system.
2	ECAT_FOE_INDICATION_TYPE_ANY_FILE_WRITTEN	If set, the stack sends an indication to the application for every file that is written successfully to the filesystem
3	ECAT_FOE_INDICATION_TYPE_VIRTUAL_FILE	The packet allows handling read and write requests to registered files, which are not stored on the volume (e.g. SYSVOLUME) but are provided by the registered application. If set, the stack sends indications to the application if the file with the registered name will be read or written. (Note: Options requests does not work on virtual files)
4	ECAT_FOE_INDICATION_TYPE_ANY_VIRTUAL_FILE	This flag is only available for LOM, not for LFW! The packet allows the read and write handling requests to any files, which are not stored on the volume (e.g. SYSVOLUME) but are provided by the registered application. If set, the stack sends indications to the application if the application reads or writes a file. (Note: Options requests do not work on virtual files, used for rcX File Handler)
5	ECAT_FOE_INDICATION_TYPE_ANY_FILE_WRITE_ABORTED	If set, the stack sends an indication to the application for every file on which the write process is aborted

Table 115. Filter Flags for blndicationType of ECAT_FOE_REGISTER_FILE_INDICATIONS_

Indications initiated by FoE Register File Indications

If the master side initiates a file operation (like up- or download) and the application is registered, the stack will receive the following indication packets that will have to be answered with the related response.

Packet	Relates to bIndication Type	Explanation
ECAT_FOE_WRITE_FILE_IND	3,4	Contains file name on first indication and only data on the following indications Example: First indication with file name: "ABCDEF" tHead.ulLen = 6 abData = { 0x41, 0x42, 0x43, 0x44, 0x45, 0x46 } Following indications with data tHead.ulLen = 10 abData = { 0x41, 0x42, 0x43, 0x44, 0x00, 0x44, 0x44, 0x44, 0x55, 0x66 }
ECAT_FOE_WRITE_FILE_RES	3,4	ulLen = 0, no data part
ECAT_FOE_READ_FILE_IND	3,4	Contains abFilename, ulPassword, and ulMaximumByteSizeOfFragment on first indication and no data part on the following indications (ulLen = 0)
ECAT_FOE_READ_FILE_RES	3, 4	After file could be accessed with filename, send abData[ulLen]
ECAT_FOE_FILE_WRITTEN_IND	1, 2	Contains file name on indication
ECAT_FOE_FILE_WRITTEN_RES	1, 2	ulLen = 0, no data part
ECAT_FOE_FILE_WRITE_ABORTED_IND	5	Contains file name on indication
ECAT_FOE_FILE_WRITE_ABORTED_RES	5	ulLen = 0, no data part

Table 116. Overview over the indications of FoE register file indications:

The indication packets for request and confirmation are put together in a packet union for easy handling. There is no constraint to use it, the packets can also be sent separately.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
tReq	ECAT_FOE_REGISTER_FILE_INDICATIONS_REQ_T	
tCnf	ECAT_FOE_REGISTER_FILE_INDICATIONS_CNF_T	

Table 117. ECAT_FOE_REGISTER_FILE_INDICATIONS_PCK_T

6.8.2.1 FoE register file indications request packet

The application has to send the request packet to the EtherCAT Slave protocol stack in order to register for indications

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	1 + n
ulSta	uint32_t	0
ulCmd	uint32_t	0x9500
tData	ECAT_FOE_REGISTER_FILE_INDICATIONS_REQ_DATA_T	
bIndicationType	uint8_t	controls what type of indication is to be registered, see flags bIndicationType
abFilename[ECAT_FOE_MAX_FILE_NAME_LENGTH]	char	contains the NUL-terminated file name to be registered for indications, max length 256

Table 118. ECAT_FOE_REGISTER_FILE_INDICATIONS_REQ_T

6.8.2.2 FoE register file indications confirmation packet

The confirmation packet confirms the registration. The data part of the confirmation packet contains the same data as the registration request packet.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	1 + n
ulSta	uint32_t	0
ulCmd	uint32_t	0x9501
tData	ECAT_FOE_REGISTER_FILE_INDICATIONS_CNF_DATA_T	
bIndicationType	uint8_t	controls what type of indication is to be registered, see flag bIndicationType
abFilename[ECAT_FOE_MAX_FILE_NAME_LENGTH]	char	contains the NUL-terminated file name to be registered for indications, max length 256

Table 119. ECAT_FOE_REGISTER_FILE_INDICATIONS_CNF_T

6.8.3 FoE unregister file service

The application has to send this request packet to the EtherCAT Slave protocol stack in order to unregister from formerly registered indications on file operations (up- or download from the master side). Depending on the value *bIndicationType*, the application does not get notifications for different events anymore. The bitmask for *bIndicationType* is the same as used for registration.

The indication packets for request and confirmation are put together in a packet union for easy handling. There is no constraint to use it, the packets can also be sent separately.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
tReq	ECAT_FOE_UNREGISTER_FILE_INDICATIONS_REQ_T	
tCnf	ECAT_FOE_UNREGISTER_FILE_INDICATIONS_CNF_T	

Table 120. ECAT_FOE_UNREGISTER_FILE_INDICATIONS_PCK_T

6.8.3.1 FoE register file indications request packet

The application has to send the request packet to the EtherCAT Slave protocol stack in order to unregister for indications

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	1 + n
ulSta	uint32_t	0
ulCmd	uint32_t	0x9502
tData	ECAT_FOE_UNREGISTER_FILE_INDICATIONS_REQ_DATA_T	
bIndicationType	uint8_t	controls what type of indication is to be registered, see bIndicationType
abFilename[ECAT_FOE_MAX_FILE_NAME_LENGTH]	char	contains the NUL-terminated file name to be registered for indications, max length 256

Table 121. ECAT_FOE_UNREGISTER_FILE_INDICATIONS_REQ_T

6.8.3.2 FoE unregister file indications confirmation packet

The confirmation packet confirms the unregistration.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20

Variable	Type	Description
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x9503

Table 122. ECAT_FOE_UNREGISTER_FILE_INDICATIONS_CNF_T

6.8.4 FoE write file service

In order to receive FoE write file indications, the application must have registered itself for receiving these using the service FoE register file indications with *bIndicationType* either set to *ECAT_FOE_INDICATION_TYPE_VIRTUAL_FILE* (=3) or to *ECAT_FOE_INDICATION_TYPE_ANY_VIRTUAL_FILE* (=4).

6.8.4.1 FoE write file indication packet

This indication packet informs the application about an incoming request to write or download a file to the file system via FoE. It initiates a series of packets to transfer the file data from the EtherCAT Master to the EtherCAT Slave device. For each new packet being received by the EtherCAT Slave stack, one more indication packet is sent to the application. The meaning of the data depends on the sequence of reception:

- The first segment contains filename and password in *abData*. The password has to be checked for correctness.
- The last segment is signaled when *ulExt.Seq* is set to LAST, it can have zero bytes of data.
- The segments in-between are signaled when *ulExt.Seq* is set to MIDDLE.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	1024
ulSta	uint32_t	0
ulCmd	uint32_t	0x9510
tData	ECAT_FOE_WRITE_FILE_IND_DATA_T	
abData[1024]	uint8_t	may be larger depending on foreign queue size first packet, following packets and last differ, see main description

Table 123. ECAT_FOE_WRITE_FILE_IND_T

6.8.4.2 FoE write file response packet

Every time the application receives an FoE write file indication from the EtherCAT Master, it should send an *ECAT_FOE_WRITE_FILE_RES* packet to the stack as response. /n Positive response or negative response are explained after packet description

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	1024
ulSta	uint32_t	0
ulCmd	uint32_t	0x9511
tData	ECAT_FOE_WRITE_FILE_RES_DATA_T	
abText[1024]	uint8_t	only valid when packet status != 0

Table 124. ECAT_FOE_WRITE_FILE_RES_T

Positive response

If the data have been received correctly, *ulSta* should be set to 0. *ulLen* should also be set to 0. In this case, *abText* is not needed.

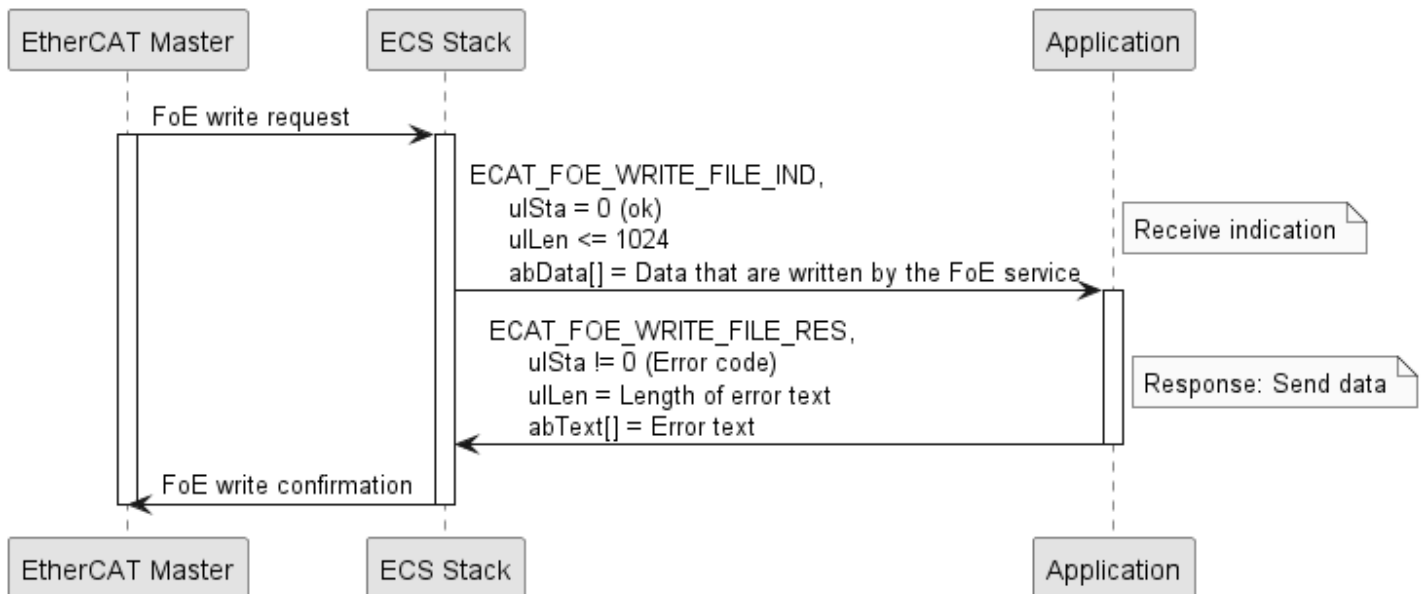


Figure 28. ECAT_FOE_WRITE_FILE_RES – FoE write file response - Positive response

Negative response

If an error occurred on reception of the FoE Write File Indication, the application should send an *ECAT_FOE_WRITE_FILE_RES* packet with *ulSta* being set to a non-zero value indicating an applicable error code. In this case, the variable *abText* should contain an informative text on the current problem (up to 1024 characters long). *ulLen* should be set to the length of *abText* (i.e. number of characters including spaces then).

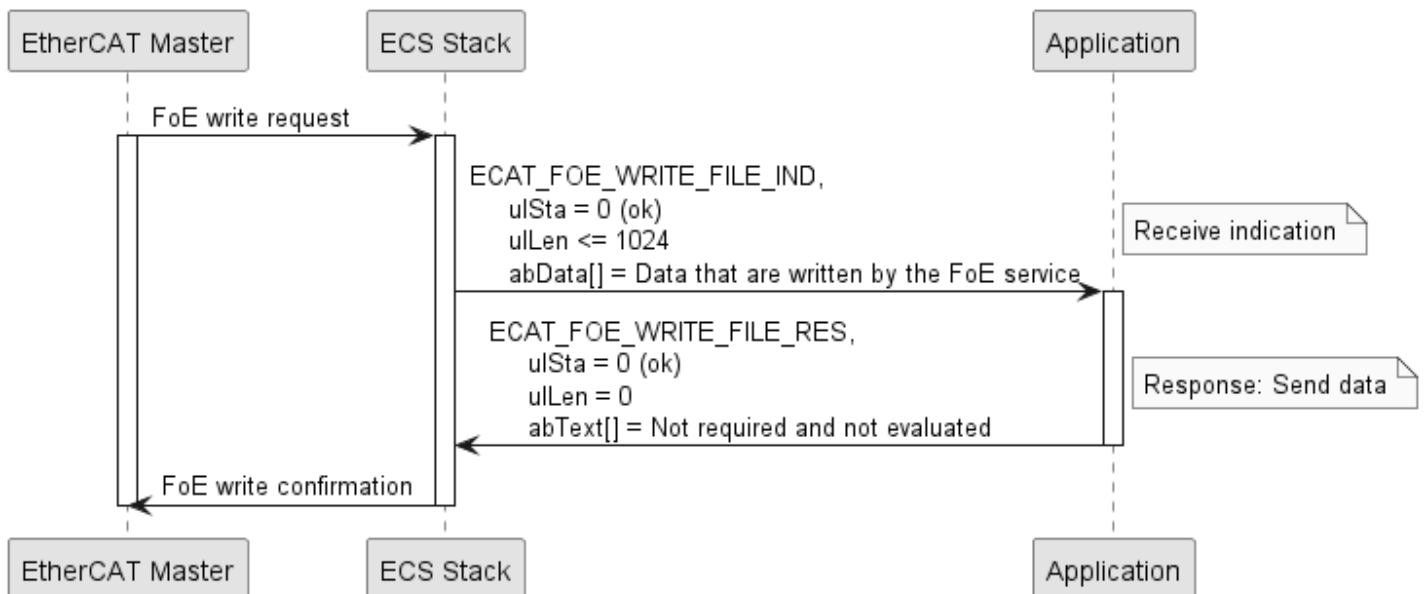


Figure 29. ECAT_FOE_WRITE_FILE_RES – FoE write file response - Negative response

6.8.5 FoE read file service

In order to receive FoE Read File Indications, the application must have registered itself for receiving these using the service FoE register file indications with *bIndicationType* either set to *ECAT_FOE_INDICATION_TYPE_VIRTUAL_FILE* (=3) or to *ECAT_FOE_INDICATION_TYPE_ANY_VIRTUAL_FILE* (=4).

6.8.5.1 FoE read file indication packet

This indication packet informs the application about an incoming request to read or upload a file from the file system of the device via FoE. It initiates a series of packets to transfer the file data from the EtherCAT Slave device to the EtherCAT Master.

The indication supplies the following information relevant for the response:

- Does a file with the requested filename really exist on the filesystem of the device.
- Is the password correct?
- Can the file be accessed?
- Is the EtherCAT Slave in the correct mode of operation (i.e. bootstrap mode)?
- Does the hardware support the downloaded firmware?

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	8+n
ulSta	uint32_t	0
ulCmd	uint32_t	0x9512
tData	ECAT_FOE_READ_FILE_IND_DATA_T	
ulMaximumByteSizeOfFragment	uint32_t	each packet fragment that is not marked LAST has to use this amount of data in its response handling on MIDDLE segments
ulPassword	uint32_t	This optional parameter contains the password of the file to be read which the application has to check for correctness, only valid on first fragment
abFilename[ECAT_FOE_MAX_FILE_NAME_LENGTH]	uint8_t	This parameter contains the name of the file to be transferred which the application has to check for existence, only valid on first fragment

Table 125. ECAT_FOE_READ_FILE_IND_T

6.8.5.2 FoE read file indication packet

This indication packet informs the application about an incoming request to read or upload a file from the file system of the device via FoE. It initiates a series of packets to transfer the file data from the EtherCAT Slave device to the EtherCAT Master.

The indication supplies the following information relevant for the response:

- Does a file with the requested filename really exist on the filesystem of the device.
- Is the password correct?
- Can the file be accessed?
- Is the EtherCAT Slave in the correct mode of operation (i.e. bootstrap mode)?
- Does the hardware support the downloaded firmware?

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0...1600 for Seq FIRST or MIDDLE, 0...1599 for Seq LAST, First and Middle equal ulMaximumByteSizeOfFragment
ulSta	uint32_t	0
ulCmd	uint32_t	0x9513
ulExt	uint32_t	ulExt.Seq = FIRST, MIDDLE or LAST
tData	ECAT_FOE_READ_FILE_RES_DATA_T	
abData[1600]	uint8_t	

Table 126. ECAT_FOE_READ_FILE_RES_T

Negative response If not all of the conditions to be tested after receiving the indication are fulfilled, the application has to send a negative response with ulSta set to an appropriate error code (). The sequence is shown in the following figure:

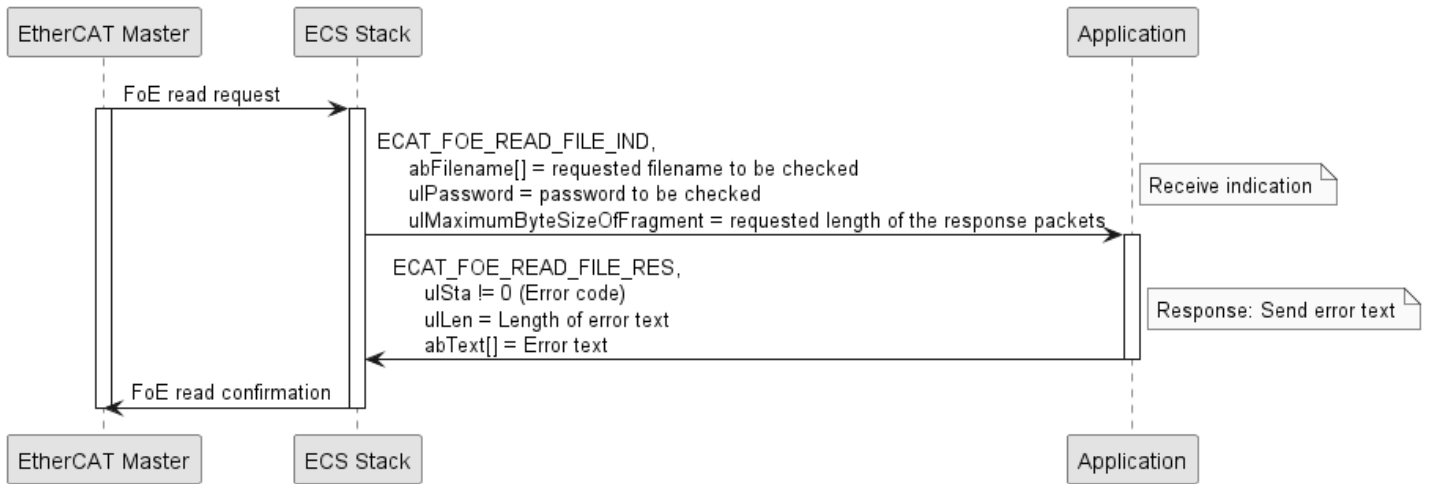


Figure 30. ECAT_FOE_READ_FILE_RES – FoE read file response - Negative response

The following error codes are available:

Code	Meaning	Description
0x8000	Not defined	See error text.
0x8001	Not found	The requested file could not be found on the server.
0x8002	Access denied	Read or right access not allowed.
0x8003	Disk full	Disk to store the file is full or memory allocation exceeded.
0x8004	Illegal operation	Illegal FoE operation, e.g. invalid service identifier
0x8005	Packet number wrong	FoE packet number invalid
0x8006	Already exists	The requested file already exists
0x8007	No user	No user
0x8008	Bootstrap only	FoE only supported in bootstrap mode
0x8009	Not bootstrap	File may not be accessed in bootstrap state
0x800A	No rights	Password invalid
0x800B	Program error	Generic programming error
0x800C	Checksum error	A checksum included in the file is invalid.
0x800D	Firmware does not fit for hardware	The hardware does not support the downloaded firmware.
0x8010	File header does not exist	Missing file header or error in file header
0x8011	Flash problem	Flash memory cannot be accessed

Table 127. FoE Read File Res Error Codes

Positive response

If all conditions are fulfilled, the file transfer can begin. Files will be transferred in segments. The following rules apply:

- For each new segment of the file to be transferred, the application is expected to send one more response packet to the EtherCAT Slave stack.
- For the first segment, *ulExt.Seq* is set to FIRST. The segment transfers as many bytes as has been specified within the variable *ulMaximumByteSizeOfFragment* of the indication packet.
- For the center segments, *ulExt.Seq* is set to MIDDLE. Set *ulLen* to 4. The segment transfers as many bytes as has been specified within the variable *ulMaximumByteSizeOfFragment* of the indication packet.
- For the last segment, *ulExt.Seq* is set to LAST. The number of bytes to be transferred is less than for a center segment (*ulLen* < *ulMaximumByteSizeOfFragment*). It can have zero bytes of data. When the data of the last segment exactly fits into the mailbox, this segment is transferred as center segment (*ulExt.Seq* is set to MIDDLE). Subsequently, the last segment is transferred with zero bytes of data.

The sequence of data transfer is as follows:

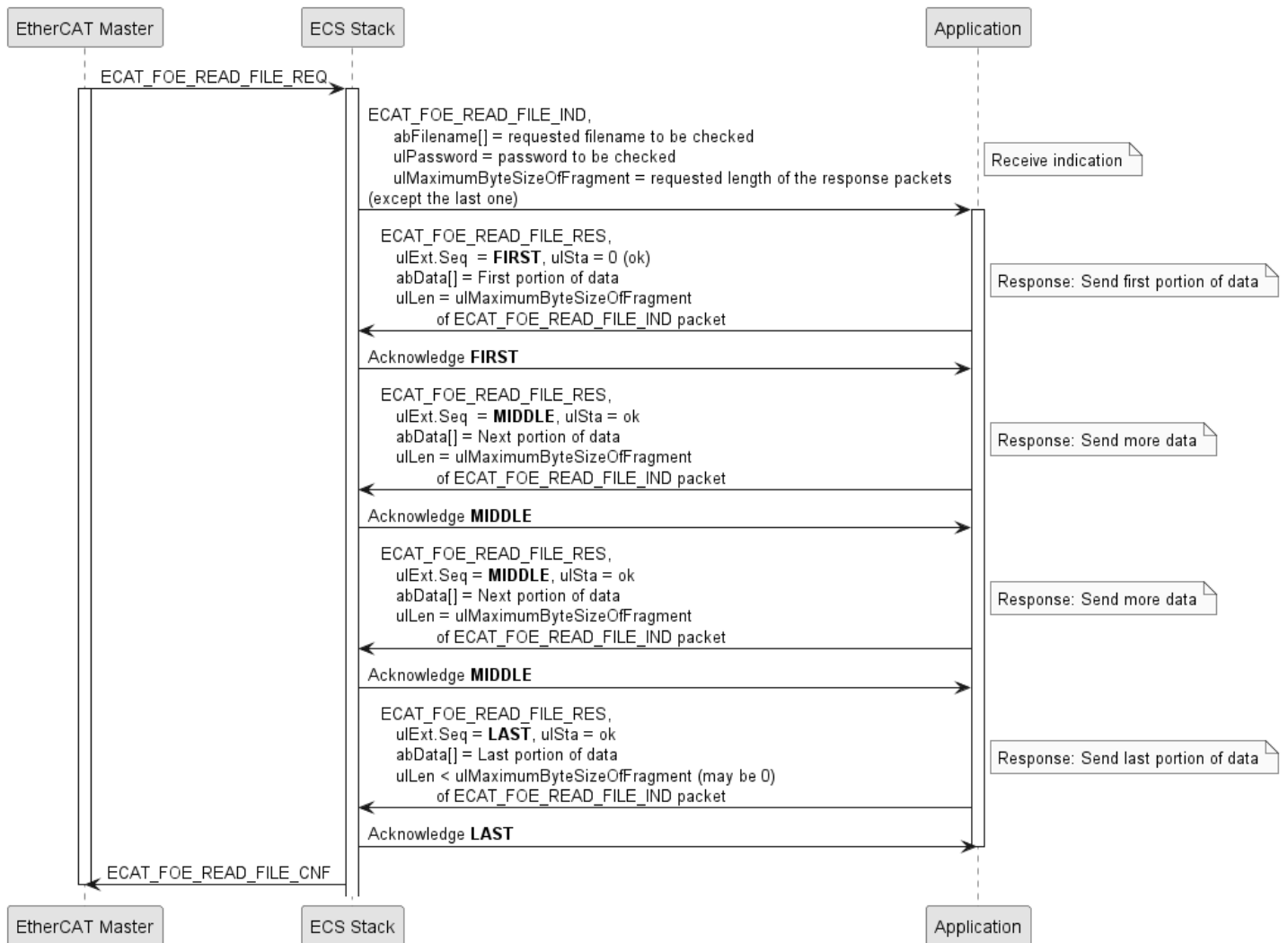


Figure 31. ECAT_FOE_READ_FILE_RES – FoE read file response - Positive response

6.8.6 FoE file written Service

In order to receive FoE file written Indications, the application must have registered itself for receiving these using the service [FoE register file service](#) with *bIndicationType* either set to

- `ECAT_FOE_INDICATION_TYPE_FILE_WRITTEN (=1)` or to
- `ECAT_FOE_INDICATION_TYPE_ANY_FILE_WRITTEN (=2)`.

The indication packets for request and confirmation are put together in a packet union for easy handling. There is no constraint to use it, the packets can also be sent separately.

Variable	Type
	Description
tHead	HIL_PACKET_HEADER_T
tInd	ECAT_FOE_FILE_WRITTEN_IND_T
tRes	ECAT_FOE_FILE_WRITTEN_RES_T

Table 128. ECAT_FOE_FILE_WRITTEN_PCK_T

6.8.6.1 FoE file written indication packet

If the FoE write file service has successfully transferred a complete file to the file system of the device, the EtherCAT Slave stack will send this indication in order to inform the application about completion of the transfer. The name of file having been written is provided in variable *abFilename[]*.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	n
ulSta	uint32_t	0
ulCmd	uint32_t	0x9520
tData	ECAT_FOE_FILE_WRITTEN_IND_DATA_T	
abFilename[ECAT_FOE_MAX_FILE_NAME_LENGTH]	uint8_t	Name of file having been written

Table 129. ECAT_FOE_FILE_WRITTEN_IND_T

6.8.6.2 FoE file written response packet

The application should acknowledge the reception of an FoE File Written Indication by sending this response packet. The FoE File Written Response packet does not have any data part.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x9521

Table 130. ECAT_FOE_FILE_WRITTEN_RES_T

6.8.7 FoE File Write Aborted Service

In order to receive FoE File Written Indications, the application must have registered itself for receiving these using the service [FoE register file service](#) with *bIndicationType* equal to `ECAT_FOE_INDICATION_TYPE_ANY_FILE_WRITE_ABORTED (=5)`.

6.8.7.1 FoE File Write Aborted Indication packet

If the FoE Write File Service has aborted the transfer of a file to the file system of the device, the EtherCAT Slave stack will send this indication in order to inform the application about the abort of the transfer. The name of the file whose transfer has been aborted is provided in variable *abFilename[]*.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	n
ulSta	uint32_t	0
ulCmd	uint32_t	0x9530
tData	ECAT_FOE_FILE_WRITE_ABORTED_IND_DATA_T	
abFilename[ECAT_FOE_MAX_FILE_NAME_LENGTH]	uint8_t	Name of file whose transfer has been aborted

Table 131. ECAT_FOE_FILE_WRITE_ABORTED_IND_T

6.8.7.2 FoE File Write Aborted response packet

The application should acknowledge the reception of an FoE File Write Aborted Indication by sending this response packet. The FoE File Write Aborted Response packet does not have any data part.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	

Variable	Type	Description
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x9531

Table 132. ECAT_FOE_FILE_WRITE_ABORTED_RES_T

6.9 ADS over EtherCAT (AoE)

The EtherCAT Slave protocol stack supports ADS over EtherCAT (AoE). ADS (Automation Device Specification) is a protocol defined within document “EtherCAT Protocol Enhancements. ETG.1020” [11] which can be optionally used to provide multiple object dictionaries when implementing a modular device according to ETG.5001. Therefore, the EtherCAT Slave protocol stack provides the possibility to work with additional object dictionaries, which can be uniquely identified by a port number in the range 0...65534.

NOTE | Within AoE, the special port number 65535 addresses the original object dictionary of ODV3.

The application has to register these additional object dictionaries at the AoE component of the EtherCAT Slave protocol stack. To do so, use the AoE register port Request (*ECS_AOE_REGISTER_PORT_REQ*). If you register an additional object dictionary using this request, then the stack will send the necessary indications to the application. As the application will have to process these indications, you will have to adapt your application accordingly. The indications to be processed include:

- ODV3_READ_OBJECT_IND/RES
- ODV3_WRITE_OBJECT_IND/RES
- ODV3_GET_OBJECT_INFO_IND/RES
- ODV3_GET_OBJECT_LIST_IND/RES
- ODV3_GET_SUBOBJECT_INFO_IND/RES
- ODV3_GET_OBJECT_ACCESS_INFO_REQ

There are two additional indications, which are only sent to the application in case that an additional object dictionary is provided via the AoE component of the EtherCAT Slave protocol stack

- ODV3_READ_ALL_BY_INDEX_IND/RES
- ODV3_WRITE_ALL_BY_INDEX_IND/RES

The AoE port number to which an indication belongs is stored within the lowest 16 bits of variable *ulld* in the indication packet. This allows a simple identification during processing the indications. If the stack detects an unregistered AoE port number, the protocol stack will issue an appropriate error message.

To distinguish whether the received object is an AoE object or was sent from another object dictionary instance (e.g. from CoE object dictionary), can be done by setting the *ulSrc* parameter in the registration packet (*ECS_AOE_REGISTER_PORT_REQ*). The value used there will appear in the *ulDest* field of the received packets, the *ulSrcID* field of the received packets holds the port number.

If an object dictionary is no longer used, it should be unregistered with the corresponding AoE unregister port Request (*ECS_AOE_UNREGISTER_PORT_REQ*). Unregistering causes the indications no longer to be sent. In this case, handling of indications is no longer necessary.

The following table gives an overview on the available AoE packets:

Service	Command	Command Code
AoE register port service	<i>ECS_AOE_REGISTER_PORT_REQ</i>	0x8D00
	<i>ECS_AOE_REGISTER_PORT_CNF</i>	0x8D01
AoE unregister port service	<i>ECS_AOE_UNREGISTER_PORT_REQ</i>	0x8D02
	<i>ECS_AOE_UNREGISTER_PORT_CNF</i>	0x8D03

Table 133. Overview over the AoE packets of the EtherCAT Slave stack

AoE also provides another important advantage compared to CoE, namely non-blocking processing. This means, contrary to CoE, you do not have to wait for an order to be finished before you can make a new order as orders can be processed in parallel.

6.9.1 AoE register port service

6.9.1.1 AoE register port request packet

The application can send this packet to the EtherCAT Slave protocol stack in order to register a port for AoE.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	6
ulSta	uint32_t	0
ulCmd	uint32_t	0x8D00
tData	ECS_AOE_REGISTER_PORT_REQ_DATA_T	
usPort	uint16_t	Port number to be registered, values 0..65534, (65535 reserved for ODV)
ulPortFlags	uint32_t	Bit mask allowing to set services, see bitmask <code>ulPortFlags</code> , value 0..3

Table 134. ECS_AOE_REGISTER_PORT_REQ_T

The following bitmask applies for the parameter `ulPortFlags`.

Value	Name	Description
0x00000001	MSK_ECS_AOE_PORT_FLAGS_SDO	
0x00000002	MSK_ECS_AOE_PORT_FLAGS_SDOINFO	

Table 135. Definitions for parameter `ulPortFlags` of ECS_AOE_REGISTER_PORT_REQ_DATA_T

6.9.1.2 AoE register port confirmation packet

By sending this packet to the application, the protocol stack confirms the registration of the specified port for AoE. The confirmation packet does not have any parameters.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x8D01

Table 136. ECS_AOE_REGISTER_PORT_CNF_T

6.9.2 AoE unregister port service

6.9.2.1 AoE unregister port request packet

This packet can be used to unregister a port from AoE.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	2
ulSta	uint32_t	0
ulCmd	uint32_t	0x8D02
tData	ECS_AOE_UNREGISTER_PORT_REQ_DATA_T	
usPort	uint16_t	

Table 137. ECS_AOE_UNREGISTER_PORT_REQ_T

6.9.2.2 AoE unregister port confirmation

6.9.2.3 AoE unregister port confirmation packet

The confirmation packet does not have any parameters. It confirms the unregistration of the specified port at AoE.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x8D03

Table 138. ECS_AOE_UNREGISTER_PORT_CNF_T

6.10 Vendor-specific protocol over EtherCAT (VoE)

VoE (Vendor-specific protocol over EtherCAT) is one of the EtherCAT mailbox protocols. As such, it is an acyclic service. The following Table 130: Overview over the VoE Packets of the EtherCAT Slave stack shows the available packets and command codes:

Service	Command	Command Code
Mailbox register type service	ECAT_MAILBOX_ADDTYPE_REQ	0x1902
	ECAT_MAILBOX_ADDTYPE_CNF	0x1903
Mailbox unregister type service	ECAT_MAILBOX_REMTYPE_REQ	0x190C
	ECAT_MAILBOX_REMTYPE_CNF	0x190D
Mailbox service	ECAT_MAILBOX_IND	0x1900
	ECAT_MAILBOX_RES	0x1901
Mailbox send service	ECAT_MAILBOX_SEND_REQ	0x1906
	ECAT_MAILBOX_SEND_CNF	0x1907

Table 139. Overview over the VoE Packets of the EtherCAT Slave stack

6.10.1 Mailbox register type service

6.10.1.1 Mailbox register type request packet

The application should send this packet to the protocol stack in order to register a task for a specific mailbox type.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	4
ulSta	uint32_t	0
ulCmd	uint32_t	0x1902
tData	ECAT_MBX_ADD_TYPE_REQ_DATA_T	
ulType	uint32_t	Mailbox type number. The type number for VoE is 0x000F, as defined in document ETG1000.4

Table 140. ECAT_MBX_ADD_TYPE_REQ_T

6.10.1.2 Mailbox register type confirmation packet

By sending this packet to the application, the protocol stack confirms the registration of a task for a specific mailbox type.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20

Variable	Type	Description
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x1903

Table 141. ECAT_MBX_ADD_TYPE_CNF_T

6.10.2 Mailbox unregister type service

6.10.2.1 Mailbox unregister type request

6.10.2.2 Mailbox unregister type request packet

This packet is used to unregister a task from a specific mailbox type.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	4
ulSta	uint32_t	0
ulCmd	uint32_t	0x190C
tData	ECAT_MBX_REM_TYPE_REQ_DATA_T	
ulType	uint32_t	Mailbox type number. The type number for VoE is 0x000F, as defined in document ETG1000.4

Table 142. ECAT_MBX_REM_TYPE_REQ_T

6.10.2.3 Mailbox unregister type confirmation packet

By sending this packet to the application, the protocol stack confirms the unregistration of a task from a specific mailbox type.

Variable	Type	Description
tHead	HIL_PACKET_HEADER_T	
ulDest	uint32_t	0x20
ulLen	uint32_t	0
ulSta	uint32_t	0
ulCmd	uint32_t	0x190D

Table 143. ECAT_MBX_REM_TYPE_CNF_T

6.10.3 Mailbox service

Mailbox indication / response

6.10.3.1 Mailbox indication packet

Every time the mailbox receives a VoE telegram, the stack sends the indication *ECAT_PACKET_MAILBOX_IND_T* to the application.

Packet description

Variable	Type	Value/Range	Description
ulDest	UINT32	0	Destination queue handle set to 0: destination is operating system rcX 32 (0x20): destination is the protocol stack
ulLen	UINT32	6 + length of bData	Packet data length in bytes
ulSta	UINT32		
ulCmd	UINT32	0x1900	ECAT_MAILBOX_IND command

Variable	Type	Value/Range	Description
tData - Structure ECAT_PACKET_MAILBOX_IND_T (= ECAT_PACKET_MAILBOX_REQ_T)			
usLength	UINT16		Mailbox type number (0x000F denotes mailbox type VoE)
usAddress	UINT16		For master use 0
uChannelandPriority	UINT8		Lower 6 bits: Channel Upper 2 bits: Priority
uType	UINT8		Mailbox type VoE = 0x0F, upper 4 bits always have to be set to 0
bData[ECAT_MAILBOX_DATA_SIZE]	UINT8[]		Data area

Table 144. ECAT_PACKET_MAILBOX_IND_T

6.10.3.2 Mailbox response packet

In LOM firmwares, the application has to answer to the indication *ECAT_PACKET_MAILBOX_IND_T* by sending this response packet to the protocol stack.

In LFW firmwares, answering to this response is not necessary.

Packet description

Variable	Type	Value/Range	Description
ulDest	UINT32	0	Destination queue handle
ulLen	UINT32	0	Packet data length in bytes
ulSta	UINT32		
ulCmd	UINT32	0x1901	ECAT_MAILBOX_RES command

Table 145. ECAT_PACKET_MAILBOX_RES_T - Mailbox response

6.10.4 Mailbox send service

Mailbox request / confirmation

6.10.4.1 Mailbox send request packet

To send VoE telegrams from the application to the network, the command code *ECAT_MAILBOX_SEND_REQ* has to be used.

Packet description

Variable	Type	Value/Range	Description
ulDest	UINT32	0	Destination queue handle set to 0: destination is operating system rcX 32 (0x20): destination is the protocol stack
ulLen	UINT32	6 + length of bData	Packet data length in bytes
ulSta	UINT32		See section [_status_and_error_codes]
ulCmd	UINT32	0x1906	ECAT_MAILBOX_SEND_REQ command
tData - Structure ECS_AOE_UNREGISTER_PORT_REQ_DATA_T			
usLength	UINT16		Mailbox type number (0x000F denotes mailbox type VoE)
usAddress	UINT16		For master use 0
usChannel	UINT8		Lower 6 bits: Channel Upper 2 bits: Priority
uType	UINT8		Mailbox type VoE = 0x0F, upper 4 bits always have to be set to 0
bData[ECAT_MAILBOX_DATA_SIZE]	UINT8[]		Data area

Table 146. ECAT_PACKET_MAILBOX_REQ_T - Mailbox send request

6.10.4.2 Mailbox send confirmation packet

The mailbox answers to a *ECAT_MAILBOX_SEND_REQ* packet with the command *ECAT_MAILBOX_SEND_CNF* and status code 0 if it has been sent properly.

Packet description

Variable	Type	Value/Range	Description
ulDest	UINT32	0	Destination queue handle
ulLen	UINT32	0	Packet data length in bytes
ulSta	UINT32		
ulCmd	UINT32	0x1907	ECAT_MAILBOX_SEND_CNF command

Table 147. ECAT_PACKET_MAILBOX_CNF_T – Mailbox send confirmation

Chapter 7 Special topics

This chapter provides information for users of linkable object modules (LOM).

7.1 For programmers

Observe the following topics:

- **Config.c**
 - The *config.c* file contains among others the hardware resource declarations and the static task list.
- **Hardware resources**
 - Besides the standard rcX resources and the user application resources, the following hardware resources should be declared. The EtherCAT Slave stack uses these resources.
- **Hardware timer**
 - The timer interval determines the minimum cycle time of the device.
- **Ethernet PHYs**
 - The Ethernet Physical Interface (PHY) is the connection between the xC Units and the Ethernet Network. They must be declared depending on the used xC code.
- **Static task list** The static task list should contain the timer task and the user application tasks.

7.2 Getting the receiver task handle of the process queue

To get the handle of the process queue of the tasks of the EtherCAT slave protocol stack the macro *TLR_QUE_IDENTIFY()* needs to be used. It is described in detail within section 10.1.9.3 of the Hilscher Task Layer Reference Model Manual. 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 tasks, which you have to use as current value for the first parameter (*pszIdn*) are

ASCII queue name	Description
"ECAT_ESM_QUE"	ECAT_ESM task queue name+ ECAT_ESM task handles all ESM states and AL control Events
"ECAT_MBX_QUE"	ECAT_MBX task queue name ECAT_MBX task handles mailboxes
"ECAT_MBXS_QUE"	ECAT_MBXS queue name ECAT_MBXS task handles send mailbox
"ECAT_COE_QUE"	ECAT_COE task queue name sending of CoE message will go through this queue
"ECAT_SDO_QUE"	ECAT_SDO task queue name ECAT_SDO task handles all SDO communications of the CoE Component part
"ECAT_EOE_QUE"	ECAT_EOE task queue name ECAT_EOE task handles all Ethernet over EtherCAT communications
"ECAT_FOE_QUE"	ECAT_FOE task queue name ECAT_FOE task handles all File Access over EtherCAT communications
"ECAT_SOEIDN_QUE"	ECAT_SOEIDN task queue name ECAT_SOEIDN task handles all Servo Profile over EtherCAT communications
"QUE_ECAT_DPM"	ECAT_DPM task queue name ECAT_DPM task handles dual port memory access

Table 148. Names of queues in EtherCAT Slave stack

The returned handle has to be used as value *ulDest* in all initiator packets the AP task intends to send to the respective task. This handle is the same handle that has to be used in conjunction with the macros like *_TLR_QUE_SENDFIFO/LIFO()* for sending a packet to the respective task.

Chapter 8 Status and error codes

Stack specific and component specific status and error codes are not listed here. They can be found in the following header files:

- EcsV4_Results.h (stack error codes)
- EcsV4_Public.h (AI status codes)
- TLR_Results.h
- TLR_global_error.h
- OdV3_Results.h or objdict_error.h (Objectdictionary related errorcodes)

8.1 Error LED

The meaning of each LED signal is specified in [docref12[12]]. The codes can be found in the related header files. As a quick reference, we list the meaning of the blinking of the error LED here:

Value	Name	Description
0	ECAT_ERRORLED_OFF	<i>No error</i> i.e. EtherCAT communication is in working condition.
1	ECAT_ERRORLED_SOLID_ON	<i>Application controller failure</i> , for instance a <i>PDI Watchdog timeout</i> has occurred (Application controller is not responding any more).
2	ECAT_ERRORLED_FLICKERING	<i>Booting error</i>
4	ECAT_ERRORLED_BLINKING	<i>Invalid Configuration: General Configuration Error</i> Example: State change commanded by master is impossible due to register or object settings. It is recommended to check and correct settings and hardware options.
5	ECAT_ERRORLED_SINGLE_FLASH	<i>Local error / Unsolicited State Change: Slave device application has changed the EtherCAT state autonomously:</i> Parameter Change in the AL status register is set to 0x01: change/error Example: Synchronization Error, device enters Safe-Operational automatically.
6	ECAT_ERRORLED_DOUBLE_FLASH	<i>Watchdog error</i> for instance, a Process Data Watchdog Timeout, EtherCAT Watchdog Timeout or Sync Manager Watchdog Timeout occurred.
7	ECAT_ERRORLED_TRIPLE_FLASH	Reserved for future use
8	ECAT_ERRORLED_QUADRUPLE_FLASH	Reserved for future use
9	ECAT_ERRORLED_QUINTUPLE_FLASH	Reserved for future use

Table 149. EtherCAT error LED codes

8.2 SDO abort codes

Return codes are generally structured into the following elements:

- Error Class
- Error Code
- Additional Code

Error class

The element Error Class (1 byte) generally classifies the kind of error, see table:

Class	Name	Description
1	vfd-state	Status error in virtual field device
2	application-reference	Error in application program
3	definition	
4	resource	Resource error
5	service	Error in service execution
6	access	Access error
7	od	Error in object dictionary
8	other	Other error

Error code The element Error Code (1 byte) accomplishes the more precise differentiation of the error cause within an Error Class. For Error Class = 8 (Other error) only Error Code = 0 (Other error code) is defined, for more detailing the Additional Code is available.

Additional code

The additional code contains the detailed error description.

8.2.1 SDO abort codes

The following codes can also be found in `Ecat_CoE_Structs.h` (LOM)

SDO abort code	Error Class	Error code	Additional code	Description
0x00000000	0	0	0	No error
0x05030000	5	3	0	Toggle bit not changed – Error in toggle bit at segmented transfer
0x05040000	5	4	0	SDO Protocol Timeout (at service execution)
0x05040001	5	4	1	Unknown command specifier (for SDO Service)
0x05040005	5	4	5	Out of memory - Memory overflow occurred at SDO Service execution
0x06010000	6	1	0	Unsupported access to an index
0x06010001	6	1	1	Write –only entry (Index may only be written but not read)
0x06010002	6	1	2	Read –only entry (Index may only be read but not written- parameter lock active)
0x06010003	6	1	3	Subindex cannot be written, subindex 0 must be 0 for write access
0x06010004	6	1	4	SDO Complete access not supported for objects of variable length such as ENUM object types
0x06010005	6	1	5	Object length exceeds mailbox size
0x06010006	6	1	6	Download blocked because object mapped to RxPDO
0x06020000	6	2	0	Object not existing – wrong index.
0x06040041	6	4	41	Object cannot be PDO-mapped – The index may not be mapped into a PDO
0x06040042	6	4	42	The number of mapped objects exceeds the capacity of the PDO
0x06040043	6	4	43	Parameter is incompatible (The data format of the parameter is incompatible for the index)
0x06040047	6	4	47	Internal device incompatibility (Device-internal error)
0x06060000	6	6	0	Hardware error (Device-internal error)
0x06070010	6	7	10	Parameter length error – data format for index has wrong size
0x06070012	6	7	12	Parameter length too long – Data format too large for index
0x06070013	6	7	13	Parameter length too short – Data format too small for index
0x06090011	6	9	11	Subindex not existing (has not been implemented)
0x06090030	6	9	30	Value exceeded a limit (value is invalid)
0x06090031	6	9	31	Value is too large
0x06090032	6	9	32	Value is too small
0x06090036	6	9	36	The maximum value is less than the minimum value
0x08000000	8	0	0	General error
0x08000020	8	0	20	Data cannot be read or stored – error in data access

SDO abort code	Error Class	Error code	Additional code	Description
0x08000021	8	0	21	Data cannot be read or stored because of local control – error in data access
0x08000022	8	0	22	Data cannot be read or stored in this state – error in data access
0x08000023	8	0	23	There is no object dictionary present.

8.3 Correspondence of SDO abort codes and status / error codes

The following table explains the correspondence between the SDO abort code on one hand and the status/error code of the EtherCAT Slave protocol stack on the other hand:

SDO abort code	Status/ Error code	Description
0x00000000	0x00000000 TL_S_OK	Status ok
0x05030000	0xC0B10001 ERR_ECSV4_COE_SDOABORT_TOGGLE_BIT_NOT_CHANGED	Toggle bit was not changed.
0x05040000	0xC0B10002 ERR_ECSV4_COE_SDOABORT_SDO_PROTOCOL_TIMEOUT	SDO protocol timeout.
0x05040001	0xC0B10003 ERR_ECSV4_COE_SDOABORT_CLIENT_SERVER_COMMAND_SPECIFIER_NOT_VALID	Client/Server command specifier not valid or unknown.
0x05040005	0xC0B10004 ERR_ECSV4_COE_SDOABORT_OUT_OF_MEMORY	Out of memory.
0x06010000	0xC0B10005 ERR_ECSV4_COE_SDOABORT_UNSUPPORTED_ACCESS_TO_AN_OBJECT	Unsupported access to an object.
0x06010001	0xC0B10006 ERR_ECSV4_COE_SDOABORT_ATTEMPT_TO_READ_A_WRITE_ONLY_OBJECT	Attempt to read a write only object.
0x06010002	0xC0B10007 ERR_ECSV4_COE_SDOABORT_ATTEMPT_TO_WRITE_TO_A_READ_ONLY_OBJECT	Attempt to write to a read only object.
0x06020000	0xC0B10008 ERR_ECSV4_COE_SDOABORT_OBJECT_DOES_NOT_EXIST	The object does not exist in the object dictionary.
0x06040041	0xC0B10009 ERR_ECSV4_COE_SDOABORT_OBJECT_CAN_NOT_BE_MAPPED_INTO_THE_PDO	The object cannot be mapped into the PDO.
0x06040042	0xC0B1000A ERR_ECSV4_COE_SDOABORT_NUMBER_AND_LENGTH_OF_OBJECTS_WOULD_EXCEED_PDO_LENGTH	The number and length of the objects to be mapped would exceed the PDO length.
0x06040043	0xC0B1000B ERR_ECSV4_COE_SDOABORT_GENERAL_PARAMETER_INCOMPATIBILITY_REASON	General parameter incompatibility reason.
0x06040047	0xC0B1000C ERR_ECSV4_COE_SDOABORT_GENERAL_INTERNAL_INCOMPATIBILITY_IN_DEVICE	General internal incompatibility in the device.
0x06060000	0xC0B1000D ERR_ECSV4_COE_SDOABORT_ACCESS_FAILED_DUE_TO_A_HARDWARE_ERROR	Access failed due to a hardware error.
0x06070010	0xC0B1000E ERR_ECSV4_COE_SDOABORT_DATA_TYPE_DOES_NOT_MATCH_LEN_OF_SRV_PARAM_DOES_NOT_MATCH	Data type does not match, length of service parameter does not match.

SDO abort code	Status/ Error code	Description
0x06070012	0xC0B1000F ERR_ECSV4_COE_SDOABORT_DATA_TYPE_DOES_NOT_MATCH_LEN_OF_SRV_PARAM_TOO_HIGH	Data type does not match, length of service parameter too high.
0x06070013	0xC0B10010 ERR_ECSV4_COE_SDOABORT_DATA_TYPE_DOES_NOT_MATCH_LEN_OF_SRV_PARAM_TOO_LOW	Data type does not match, length of service parameter too low.
0x06090011	0xC0B10011 ERR_ECSV4_COE_SDOABORT_SUBINDEX_DOES_NOT_EXIST	Subindex does not exist.

8.4 CoE emergency codes

The CoE emergency codes are defined by can open specification and adapted by Ethercat. To understand them in the Ethercat context and to make it easier to find the meaning, this section is added here.

Error Code (Hexadecimal Value)	Meaning of code
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	Current, device input side
22xx	Current inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains Voltage
32xx	Voltage inside the device
33xx	Output Voltage
40xx	Temperature
41xx	Ambient Temperature
42xx	Device Temperature
50xx	Device Hardware
60xx	Device Software
61xx	Internal Software
62xx	User Software
63xx	Data Set
70xx	Additional Modules
80xx	Monitoring
81xx	Communication
82xx	Protocol Error
8210	PDO not processed due to length error
8220	PDO length exceeded
90xx	External Error
A0xx	EtherCAT State Machine Transition Error
F0xx	Additional Functions
FFxx	Device specific

Table 150. CoE emergency codes

Appendix A: Appendix

A.1 List of figures

Figure 1. Usecase loadable Firmware

Figure 2. Usecase loadable Firmware

Figure 3. Set Configuration / Channel Init

Figure 4. Firmware structure

Figure 5. State diagram of EtherCAT State Machine (ESM)

Figure 6. Sequence diagram of state change with indication to application/host

Figure 7. Sequence diagram of EtherCAT state change controlled by application/host

Figure 8. Sequence diagram of state change controlled by application/host with additional AL status changed indications

Figure 9. Mapping scheme for a PDO

Figure 10. Sequence within the application

Figure 11. Initialization sequence with placing of registrations and object dictionary creation

Figure 12. Initialization sequence for Explicit Device Identification

Figure 13. SDO download with Complete Access (successful)

Figure 14. Dynamic PDO assignment: One application registered for write indications (successful)

Figure 15. Dynamic PDO assignment: One application registered for write indications (not successful)

Figure 16. Dynamic PDO assignment with Complete Access: One application registered for write indications (successful)

Figure 17. Set ready service request

Figure 18. Relation between Set Configuration packet and ESI file

Figure 19. Send CoE emergency service

Figure 20. SII write Indication service

Figure 21. Sequence diagram for ECAT_EOE_REGISTER_FOR_FRAME_INDICATIONS_REQ/CNF packets

Figure 22. Sequence diagram for ECAT_EOE_UNREGISTER_FROM_FRAME_INDICATIONS_REQ/CNF packets

Figure 23. Sequence diagram EoE frame reception

Figure 24. Sequence diagram for ECAT_EOE_REGISTER_FOR_IP_PARAM_INDICATIONS_REQ/CNF

Figure 25. Sequence diagram for ECAT_EOE_UNREGISTER_FOR_IP_PARAM_INDICATIONS_REQ/CNF

Figure 26. Set IP parameter service

Figure 27. Get IP parameter service

Figure 28. ECAT_FOE_WRITE_FILE_RES – FoE write file response - Positive response

Figure 29. ECAT_FOE_WRITE_FILE_RES – FoE write file response - Negative response

Figure 30. ECAT_FOE_READ_FILE_RES – FoE read file response - Negative response

Figure 31. ECAT_FOE_READ_FILE_RES – FoE read file response - Positive response

A.2 List of tables

- Table 1. Component configuration parameters
- Table 2. Extended configuration parameters
- Table 3. Input and output data netX 100, 500
- Table 4. Input and output data netX 100, 500
- Table 5. Slave Information Interface structure as defined in IEC 61158, part 6-12
- Table 6. Available standard categories
- Table 7. Slave Information Interface Categories
- Table 8. Abstract of the CoE Communication Area (0x1000 - 0x1FFF)
- Table 9. Minimal object dictionary
- Table 10. Default object dictionary
- Table 11. Extended status block
- Table 12. Set device identification value
- Table 13. abParameter
- Table 14. Request packet *RCX_SET_FW_PARAMETER_REQ_T*
- Table 15. Confirmation packet *RCX_SET_FW_PARAMETER_CNF_T*
- Table 16. EtherCAT Slave stack components
- Table 17. Summary of all queue names, which may be used by an AP task
- Table 18. Overview over the general packets of the EtherCAT Slave stack
- Table 19. Bitmask ulReadyBits of *ECAT_ESM_SETREADY_REQ_DATA_T*
- Table 20. *ECAT_ESM_SETREADY_REQ_T*
- Table 21. *ECAT_ESM_SETREADY_CNF_T*
- Table 22. *ECAT_ESM_INIT_COMPLETE_IND_T*
- Table 23. *ECAT_ESM_INIT_COMPLETE_RES_T*
- Table 24. Configuration packets overview
- Table 25. *ECAT_SET_CONFIG_REQ_T*
- Table 26. Basic configuration data *ECAT_SET_CONFIG_REQ_DATA_BASIC_T*
- Table 27. Flags for ulSystemFlags
- Table 28. Values for the parameters ulVendorId, ulProductCode and ulRevisionNumber
- Table 29. Parameter ulComponentInitialization
- Table 30. *ECAT_SET_CONFIG_REQ_DATA_COMPONENTS_T*
- Table 31. *ECAT_SET_CONFIG_COE_T*
- Table 32. Flag for bCoeDetails
- Table 33. *ECAT_SET_CONFIG_EOE_T*
- Table 34. *ECAT_SET_CONFIG_FOE_T*
- Table 35. *ECAT_SET_CONFIG_SOE_T*
- Table 36. *ECAT_SET_CONFIG_SYNCMODES_T*
- Table 37. Flag for bSyncSource
- Table 38. Example for SM2 synchronous mode
- Table 39. *ECAT_SET_CONFIG_SYNCPDI_T*
- Table 40. Definitions for parameter bSyncPdiConfig of *ECAT_SET_CONFIG_SYNCPDI_T*
- Table 41. *ECAT_SET_CONFIG_UID_T*
- Table 42. *ECAT_SET_CONFIG_BOOTMBX_T*
- Table 43. *ECAT_SET_CONFIG_DEVICEINFO_T*
- Table 44. Device info configuration parameters
- Table 45. *ECAT_SET_CONFIG_SMLENGTH_T*
- Table 46. *ECAT_SET_CONFIG_CNF_T*

Table 47.	ECAT_SET_CONFIG_EXT_STRUCTURE_TYPE_E
Table 48.	ECAT_SET_CONFIG_EXT_DATA_TYPE_SMS_T
Table 49.	ECAT_SET_CONFIG_EXT_DATA_TYPE_BOOTMBX_T
Table 50.	ECAT_SET_CONFIG_EXT_STRUCTUREDATA_T
Table 51.	ECAT_SET_CONFIG_EXT_REQ_DATA_T
Table 52.	ECAT_SET_CONFIG_EXT_REQ_T
Table 53.	ECAT_SET_CONFIG_EXT_CNF_T
Table 54.	ECAT_DPM_SET_IO_SIZE_REQ_T
Table 55.	ECAT_DPM_SET_IO_SIZE_CNF_T
Table 56.	ECAT_DPM_SET_STATION_ALIAS_REQ_T
Table 57.	ECAT_DPM_SET_STATION_ALIAS_CNF_T
Table 58.	ECAT_DPM_GET_STATION_ALIAS_REQ_T
Table 59.	ECAT_DPM_GET_STATION_ALIAS_CNF_T
Table 60.	Overview over the EtherCAT state machine related packets of the EtherCAT Slave stack
Table 61.	ECAT_ESM_REGISTER_FOR_ALCONTROL_INDICATIONS_REQ_T
Table 62.	ECAT_ESM_REGISTER_FOR_ALCONTROL_INDICATIONS_CNF_T
Table 63.	ECAT_ESM_UNREGISTER_FROM_ALCONTROL_INDICATIONS_REQ_T
Table 64.	ECAT_ESM_UNREGISTER_FROM_ALCONTROL_INDICATIONS_CNF_T
Table 65.	ECAT_ESM_ALCONTROL_CHANGED_IND_T
Table 66.	State definitions for AlControl uState
Table 67.	Coding of EtherCAT state
Table 68.	ECAT_ESM_ALCONTROL_CHANGED_RES_T
Table 69.	ECAT_ESM_ALSTATUS_CHANGED_IND_T
Table 70.	State definitions for AlStatus uState
Table 71.	ECAT_ESM_ALSTATUS_CHANGED_RES_T
Table 72.	ECAT_ESM_SET_ALSTATUS_REQ_T
Table 73.	ECAT_ESM_SET_ALSTATUS_CNF_T
Table 74.	ECAT_ESM_GET_ALSTATUS_REQ_T
Table 75.	ECAT_ESM_GET_ALSTATUS_CNF_T
Table 76.	Overview over the CoE packets of the EtherCAT Slave stack
Table 77.	ECAT_COE_SEND_EMERGENCY_REQ_T
Table 78.	Bit Mask bErrorRegister
Table 79.	ECAT_COE_SEND_EMERGENCY_CNF_T
Table 80.	Overview over the SII packets of the EtherCAT Slave stack
Table 81.	ECAT_ESM_SII_READ_REQ_T
Table 82.	ECAT_ESM_SII_READ_CNF_T
Table 83.	ECAT_ESM_SII_WRITE_REQ_T
Table 84.	ECAT_ESM_SII_WRITE_CNF_T
Table 85.	ECAT_ESM_REGISTER_FOR_SIIWRITE_INDICATIONS_REQ_T
Table 86.	ECAT_ESM_REGISTER_FOR_SIIWRITE_INDICATIONS_CNF_T
Table 87.	ECAT_ESM_UNREGISTER_FROM_SIIWRITE_INDICATIONS_REQ_T
Table 88.	ECAT_ESM_UNREGISTER_FROM_SIIWRITE_INDICATIONS_CNF_T
Table 89.	ECAT_ESM_SII_WRITE_IND_T
Table 90.	ECAT_ESM_SII_WRITE_RES_T
Table 91.	Overview over the EoE Packets of the EtherCAT Slave Stack
Table 92.	ECAT_EOE_REGISTER_FOR_FRAME_INDICATIONS_REQ_T
Table 93.	ECAT_EOE_REGISTER_FOR_FRAME_INDICATIONS_CNF_T

Table 94.	ECAT_EOE_UNREGISTER_FROM_FRAME_INDICATIONS_REQ_T
Table 95.	ECAT_EOE_UNREGISTER_FROM_FRAME_INDICATIONS_CNF_T
Table 96.	Meaning of bit mask usFlags
Table 97.	ECAT_EOE_SEND_FRAME_REQ_T
Table 98.	ECAT_EOE_SEND_FRAME_CNF_T
Table 99.	ECAT_EOE_FRAME_RECEIVED_IND_T
Table 100.	ECAT_EOE_FRAME_RECEIVED_RES_T
Table 101.	ECAT_EOE_REGISTER_FOR_IP_PARAM_INDICATIONS_REQ_T
Table 102.	ECAT_EOE_REGISTER_FOR_IP_PARAM_INDICATIONS_CNF_T
Table 103.	ECAT_EOE_UNREGISTER_FROM_IP_PARAM_INDICATIONS_REQ_T
Table 104.	ECAT_EOE_UNREGISTER_FROM_IP_PARAM_INDICATIONS_CNF_T
Table 105.	Bitmask for parameter ulFlag of ECAT_EOE_SET_IP_PARAM_IND_DATA_T
Table 106.	ECAT_EOE_SET_IP_PARAM_IND_T
Table 107.	ECAT_EOE_SET_IP_PARAM_RES_T
Table 108.	Bitmask for parameter ulFlag of ECAT_EOE_GET_IP_PARAM_IND_DATA_T
Table 109.	ECAT_EOE_GET_IP_PARAM_IND_T
Table 110.	ECAT_EOE_GET_IP_PARAM_RES_T
Table 111.	Overview over the FoE Packets of the EtherCAT Slave Stack
Table 112.	Filter Flags for ECAT_FOE_SET_OPTIONS_REQ_DATA_T
Table 113.	ECAT_FOE_SET_OPTIONS_REQ_T
Table 114.	ECAT_FOE_SET_OPTIONS_CNF_T
Table 115.	Filter Flags for blndicationType of ECAT_FOE_REGISTER_FILE_INDICATIONS_
Table 116.	Overview over the indications of FoE register file indications:
Table 117.	ECAT_FOE_REGISTER_FILE_INDICATIONS_PCK_T
Table 118.	ECAT_FOE_REGISTER_FILE_INDICATIONS_REQ_T
Table 119.	ECAT_FOE_REGISTER_FILE_INDICATIONS_CNF_T
Table 120.	ECAT_FOE_UNREGISTER_FILE_INDICATIONS_PCK_T
Table 121.	ECAT_FOE_UNREGISTER_FILE_INDICATIONS_REQ_T
Table 122.	ECAT_FOE_UNREGISTER_FILE_INDICATIONS_CNF_T
Table 123.	ECAT_FOE_WRITE_FILE_IND_T
Table 124.	ECAT_FOE_WRITE_FILE_RES_T
Table 125.	ECAT_FOE_READ_FILE_IND_T
Table 126.	ECAT_FOE_READ_FILE_RES_T
Table 127.	FoE Read File Res Error Codes
Table 128.	ECAT_FOE_FILE_WRITTEN_PCK_T
Table 129.	ECAT_FOE_FILE_WRITTEN_IND_T
Table 130.	ECAT_FOE_FILE_WRITTEN_RES_T
Table 131.	ECAT_FOE_FILE_WRITE_ABORTED_IND_T
Table 132.	ECAT_FOE_FILE_WRITE_ABORTED_RES_T
Table 133.	Overview over the AoE packets of the EtherCAT Slave stack
Table 134.	ECS_AOE_REGISTER_PORT_REQ_T
Table 135.	Definitions for parameter ulPortFlags of ECS_AOE_REGISTER_PORT_REQ_DATA_T
Table 136.	ECS_AOE_REGISTER_PORT_CNF_T
Table 137.	ECS_AOE_UNREGISTER_PORT_REQ_T
Table 138.	ECS_AOE_UNREGISTER_PORT_CNF_T
Table 139.	Overview over the VoE Packets of the EtherCAT Slave stack
Table 140.	ECAT_MBX_ADD_TYPE_REQ_T



Table 141. ECAT_MBX_ADD_TYPE_CNF_T

Table 142. ECAT_MBX_REM_TYPE_REQ_T

Table 143. ECAT_MBX_REM_TYPE_CNF_T

Table 144. ECAT_PACKET_MAILBOX_IND_T

Table 145. ECAT_PACKET_MAILBOX_RES_T - Mailbox response

Table 146. ECAT_PACKET_MAILBOX_REQ_T - Mailbox send request

Table 147. ECAT_PACKET_MAILBOX_CNF_T - Mailbox send confirmation

Table 148. Names of queues in EtherCAT Slave stack

Table 149. EtherCAT error LED codes

Table 150. CoE emergency codes



A.3 List of snippets

A.4 Legal Notes

Copyright

© Hilscher Gesellschaft für Systemautomation mbH

All rights reserved.

The images, photographs and texts in the accompanying materials (in the form of a user's manual, operator's manual, Statement of Work document and all other document types, support texts, documentation, etc.) are protected by German and international copyright and by international trade and protective provisions. Without the prior written consent, you do not have permission to duplicate them either in full or in part using technical or mechanical methods (print, photocopy or any other method), to edit them using electronic systems or to transfer them. You are not permitted to make changes to copyright notices, markings, trademarks or ownership declarations. Illustrations are provided without taking the patent situation into account. Any company names and product designations provided in this document may be brands or trademarks by the corresponding owner and may be protected under trademark, brand or patent law. Any form of further use shall require the express consent from the relevant owner of the rights.

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 uninterrupted 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.

A.5 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.
69800 Saint Priest
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, Bangalore
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Support

Phone: +91 8108884011
E-Mail: info@hilscher.in

Austria

Hilscher Austria GmbH
4020 Linz
Phone: +43 732 931 675-0
E-Mail: sales.at@hilscher.com

Support

Phone: +43 732 931 675-0
E-Mail: at.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

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

Republic of Korea

Hilscher Korea Inc.
13494, Seongnam, Gyeonggi
Phone: +82 (0) 31-739-8361
E-Mail: info@hilscher.kr

Support

Phone: +82 (0) 31-739-8363
E-Mail: kr.support@hilscher.com

Switzerland

Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch

Support

Phone: +41 (0) 32 623 6633
E-Mail: ch.support@hilscher.com

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