Protocol API

Ethernet POWERLINK Controlled Node

V3.3.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC160504API05EN | Revision 5 | English | 2017-07 | Released | Public

Table of Contents

1 Introduction

1.1 Abstract

This manual describes the application interface of the Ethernet POWERLINK Controlled Node stack, with the aim to support and lead you during the integration process of the given stack into your own application.

1.2 List of revisions

Rev	Date	Name	Revisions
2	2016-08-10	MA, RG	Firmware version V3.2.0 New services for diagnosis described in section Diagnosis: *Write Static Error Bit Field service* *Write Status Entry service* *New Error Entry Indication service* *New Status Entry Indication service*
3	2016-12-08	HH	Firmware version V3.2.0 Section Extended Status revoved.
4	2017-03-14	HH	Firmware version V3.3.0 Defines for *Status codes / Error* codes has been changed (EPLCN_E_ replaced by ERR_EPLCN_).
5	2017-07-06	MA, HH	Firmware version V3.3.0 Section *Object Dictionary entries (Communication Profile Area)*: Index 1C0E and 1C13 added. Section *Configure Stack service*: Parameter ulThresholdSoCJitter added. Table 46: E_DLL_JITTER_TH added.

Table 1: List of revisions

1.3 Functional overview

This stack has been written to meet the requirements outlined in the Ethernet POWERLINK (EPL) specification. The user of this stack is provided with a fully functional general-purpose Software package with the following main features:

- Implementation of the EPL- state machine
- Implementation of the CANopen-style object dictionary according to the EPL specification

1.4 System requirements

This software package has the following environmental system requirements:

- netX-Chip as CPU hardware platform
- Operating system for task scheduling required

1.5 Intended audience

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

- Knowledge of the programming language C
- Knowledge of the use of the real time operating system rcX
- Knowledge of the Ethernet Powerlink V 2.0 Specification

1.6 Specifications

The data below applies to the POWERLINK Controlled Node firmware and stack version V3.3.0.

1.6.1 Technical data

State machine

Implementation of the EPL-state machine

Object dictionary

Implementation of the CANopen-style object dictionary according to the EPL specification

Technical data

Maximum number of cyclic input data	1490 bytes (800 bytes on netX52)
Maximum number of cyclic output data	1490 bytes (800 bytes on netX52)
Acyclic data transfer	SDO Upload/Download
Functions:	SDO over ASnd and UDP
Baud rate	100 MBit/s, half-duplex
Data transport layer	Ethernet II, IEEE 802.3
Ethernet Powerlink version	V 2

Table 2: Technical data

Firmware/stack available for netX

netX50	no
netX 51, netX52	yes
netX 100, netX 500	yes

Table 3: Availability of firmware/stack

Configuration

Configuration by packet to transfer warmstart parameters

Diagnostic

Firmware supports common diagnostic in the dual-port-memory for loadable firmware

Limitations

No slave to slave communication (Loadable firmware)

Stack supports and handles only one network interface

1.7 Terms, abbreviations and definitions

Term	Description
AP	Application
API	Application Programming Interface
ARP	Address Resolution Protocol
ASnd	Asynchronous Send
CN	Controlled Node
CRC	Cyaclic Redundancy Check
DLL	Data Link Layer
DPM	Dual-Port Memory
EPL	Ethernet POWERLINK
EPSG	Ethernet POWERLINK Standardisation Group
HAL	Hardware Abstraction Layer
IEEE	Institute of Electrical and Electronic Engineers
IF	Interface
IP	Internet Protocol
LSB	Least significant byte
MN	Managing Node
MSB	Most significant byte
NMT	Network Management
OD	Object Dictionary
ODV3	Object Dictionary Version 3
PDO	Process Data Object
PReq	Poll Request
PRes	Poll Response
RAM	Random Access Memory
RxPDO	Receive PDO
SoA	Start of Asynchronous
SoC	Start of Cyclic/Start of Cycle
SDO	Service Data Object
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
TxPDO	Transmit PDO
XDD	XML Device Description
XML	Extensible Markup Language

Table 4: Terms, abbreviations and definitions

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

1.8 References to documents

This document is based on the following specifications:

1. Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX based products. Revision 12, English, 2008-2012, Document ID DOC060302DPM12EN
2. Ethernet Powerlink Communication Profile Specification; EPSG DS 301 V1.2.0; 2013
3. Hilscher Gesellschaft für Systemautomation mbH: Object Dictionary V3 API 03 Protocol API, Document ID DOC110106API03EN
4. Ethernet Powerlink XML Device Description; EPSG DS 311 V1.0.0; 2007

Table 5: References to documents

1.9 Legal notes

Copyright

Important notes

Utmost care was/is given in the preparation of the documentation at hand consisting of a user's manual, operating manual and any other document type and accompanying texts. However, errors cannot be ruled out. Therefore, we cannot assume any guarantee or legal responsibility for erroneous information or liability of any kind. You are hereby made aware that descriptions found in the user's manual, the accompanying texts and the documentation neither represent a guarantee nor any indication on proper use as stipulated in the agreement or a promised attribute. It cannot be ruled out that the user's manual, the accompanying texts and the documentation do not completely match the described attributes, standards or any other data for the delivered product. A warranty or guarantee with respect to the correctness or accuracy of the information is not assumed.

We reserve the right to modify our products and the specifications for such as well as the corresponding documentation in the form of a user's manual, operating manual and/or any other document types and accompanying texts at any time and without notice without being required to notify of said modification. Changes shall be taken into account in future manuals and do not represent an obligation of any kind, in particular there shall be no right to have delivered documents revised. The manual delivered with the product shall apply.

Under no circumstances shall Hilscher Gesellschaft für Systemautomation mbH be liable for direct, indirect, ancillary or subsequent damage, or for any loss of income, which may arise after use of the information contained herein.

Liability disclaimer

The hardware and/or software was created and tested by Hilscher Gesellschaft für Systemautomation mbH with utmost care and is made available as is. No warranty can be assumed for the performance or flawlessness of the hardware and/or software under all application conditions and scenarios and the work results achieved by the user when using the hardware and/or software. Liability for any damage that may have occurred as a result of using the hardware and/or software or the corresponding documents shall be limited to an event involving willful intent or a grossly negligent violation of a fundamental contractual obligation. However, the right to assert damages due to a violation of a fundamental contractual obligation shall be limited to contract-typical foreseeable damage.

It is hereby expressly agreed upon in particular that any use or utilization of the hardware and/or software in connection with

- Flight control systems in aviation and aerospace;
- Nuclear fusion processes in nuclear power plants;
- Medical devices used for life support and
- Vehicle control systems used in passenger transport

shall be excluded. Use of the hardware and/or software in any of the following areas is strictly prohibited:

- For military purposes or in weaponry;
- For designing, engineering, maintaining or operating nuclear systems;
- In flight safety systems, aviation and flight telecommunications systems;
- In life-support systems;
- In systems in which any malfunction in the hardware and/or software may result in physical injuries or fatalities.

You are hereby made aware that the hardware and/or software was not created for use in hazardous environments, which require fail-safe control mechanisms. Use of the hardware and/or software in this kind of environment shall be at your own risk; any liability for damage or loss due to impermissible use shall be excluded.

Warranty

Hilscher Gesellschaft für Systemautomation mbH hereby guarantees that the software shall run without errors in accordance with the requirements listed in the specifications and that there were no defects on the date of acceptance. The warranty period shall be 12 months commencing as of the date of acceptance or purchase (with express declaration or implied, by customer's conclusive behavior, e.g. putting into operation permanently).

The warranty obligation for equipment (hardware) we produce is 36 months, calculated as of the date of delivery ex works. The aforementioned provisions shall not apply if longer warranty periods are mandatory by law pursuant to Section 438 (1.2) BGB, Section 479 (1) BGB and Section 634a (1) BGB [Bürgerliches Gesetzbuch; German Civil Code] If, despite of all due care taken, the delivered product should have a defect, which already existed at the time of the transfer of risk, it shall be at our discretion to either repair the product or to deliver a replacement product, subject to timely notification of defect.

The warranty obligation shall not apply if the notification of defect is not asserted promptly, if the purchaser or third party has tampered with the products, if the defect is the result of natural wear, was caused by unfavorable operating conditions or is due to violations against our operating regulations or against rules of good electrical engineering practice, or if our request to return the defective object is not promptly complied with.

Costs of support, maintenance, customization and product care

Please be advised that any subsequent improvement shall only be free of charge if a defect is found. Any form of technical support, maintenance and customization is not a warranty service, but instead shall be charged extra.

Additional guarantees

Although the hardware and software was developed and tested in-depth with greatest care, Hilscher Gesellschaft für Systemautomation mbH shall not assume any guarantee for the suitability thereof for any purpose that was not confirmed in writing. No guarantee can be granted whereby the hardware and software satisfies your requirements, or the use of the hardware and/or software is uninterruptable or the hardware and/or software is fault-free.

It cannot be guaranteed that patents and/or ownership privileges have not been infringed upon or violated or that the products are free from third-party influence. No additional guarantees or promises shall be made as to whether the product is market current, free from deficiency in title, or can be integrated or is usable for specific purposes, unless such guarantees or promises are required under existing law and cannot be restricted.

Confidentiality

The customer hereby expressly acknowledges that this document contains trade secrets, information protected by copyright and other patent and ownership privileges as well as any related rights of Hilscher Gesellschaft für Systemautomation mbH. The customer agrees to treat as confidential all of the information made available to customer by Hilscher Gesellschaft für Systemautomation mbH and rights, which were disclosed by Hilscher Gesellschaft für Systemautomation mbH and that were made accessible as well as the terms and conditions of this agreement itself.

The parties hereby agree to one another that the information that each party receives from the other party respectively is and shall remain the intellectual property of said other party, unless provided for otherwise in a contractual agreement.

The customer must not allow any third party to become knowledgeable of this expertise and shall only provide knowledge thereof to authorized users as appropriate and necessary. Companies associated with the customer shall not be deemed third parties. The customer must obligate authorized users to confidentiality. The customer should only use the confidential information in connection with the performances specified in this agreement.

The customer must not use this confidential information to his own advantage or for his own purposes or rather to the advantage or for the purpose of a third party, nor must it be used for commercial purposes and this confidential information must only be used to the extent provided for in this agreement or otherwise to the extent as expressly authorized by the disclosing party in written form. The customer has the right, subject to the obligation to confidentiality, to disclose the terms and conditions of this agreement directly to his legal and financial consultants as would be required for the customer's normal business operation.

Export provisions

The delivered product (including technical data) is subject to the legal export and/or import laws as well as any associated regulations of various countries, especially such laws applicable in Germany and in the United States. The products / hardware / software must not be exported into such countries for which export is prohibited under US American export control laws and its supplementary provisions. You hereby agree to strictly follow the regulations and to yourself be responsible for observing them. You are hereby made aware that you may be required to obtain governmental approval to export, reexport or import the product.

1.10 Third party software licenses

lwIP IP stack

This software package uses the lwIP software for IP stack functionality. The following licensing conditions apply for this component:

2 Getting started / Configuration

This section explains some essential information you should know when starting to work with the Ethernet POWERLINK Controlled Node Protocol API.

2.1 Overview about essential functionality

You can find the most commonly used functionality of the Ethernet POWERLINK Controlled Node Protocol API within the following sections of this document:

Table 6: Overview about essential functionality

2.2 Configuration of POWERLINK Controlled Node

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

- Configuration via SYCON.net
- Configuration via packets
  - Static mapping configuration with default PDO objects created by the CN stack.
  - Static mapping configuration with PDO objects defined and created by the user.
  - Dynamic mapping configuration with PDO objects defined and created by the user.

All configuration variants via packets provide two running modes of the CN state machine:

- Automatic running mode: After the CN is configured and started; the complete state machine is triggered automatically.
- Application triggered mode: In this mode the application is responsible to trigger the state machine in the initialization phase and to confirm the EnableReadyToOperate command received from the bus.

2.2.1 Using the configuration tool SYCON.net

This configuration method is described in the tool documentation.

2.2.2 Using configuration via packets

The following sections will outline the different variants for configuring the CN via packets.

These parameters of *Configure Stack service* (page 44) are evaluated for all variants:

- Automatic or application triggered start of CN
- Automatic or application triggered running mode of CN state machine
- PReq exchange triggered from bus or from application
- Enable or disable PDO mapping version check
- Enable signaling transmit data validity using the DPM ApplicationReady flag
- Identity parameters (vendor and product identification data)
- POWERLINK communication parameters:
  - Cycle length
  - Node address
  - Error thresholds
  - Status entries
  - PDO size

2.2.2.1 Static mapping configuration with default PDO objects created by the CN stack

This configuration method is selected by keeping cleared the bit 1 in the field `ulStackCfgFlags` and the bit 6 in the field `ulFeatureFlags` of the *Configure Stack service* on page 44. For details, see chapter *Static mapping configuration with default PDO objects* on page 54.

When using this configuration way, the CN will configure default PDO objects based on the given data size. The stack will be the own responsible of these objects and their handling.

The mapping between the cyclic data stream on the bus and the created PDO objects is static.

In the cyclic communication, the application will just have to handle with transmit and receive data streams.

Acyclic access to the PDO objects will be handled in the CN stack.

2.2.2.2 Static mapping configuration with PDO objects defined and created by the user

This configuration method is selected by setting the bit 1 in the field `ulStackCfgFlags` and keeping the bit 6 cleared in the field `ulFeatureFlags` of the *Configure Stack service* on page 44. For details, see chapter *Static mapping configuration with user defined PDO objects* on page 57.

When using this configuration way, the application is responsible to create and handle the access to the PDO objects.

The application has also to define and implement the mapping configuration between data stream in the bus and its own PDO objects. This mapping configuration is static.

In the cyclic communication, the application will exchange with the DPM the same data stream as in the bus and it is up to the application to map the data to the corresponding PDO objects depending on the configuration.

Acyclic access to the PDO objects has to be handled in the application.

Use the Object Dictionary V3 (ODV3) packets to create and handle the PDO objects.

Note: It is mandatory to create the necessary PDO objects and implement all corresponding access handlings.

2.2.2.3 Dynamic mapping configuration with PDO objects defined and created by the user

This configuration method is selected by setting the bit 1 in the field `ulStackCfgFlags` and the bit 6 in the field `ulFeatureFlags` of *Configure Stack service* on page 44. For details, see chapter *Dynamic mapping configuration* on page 59.

When using this configuration way, the application is responsible to create and handle the access to the PDO objects.

The application has also to define and implement a default mapping configuration between data stream in the bus and its own PDO objects. This mapping is dynamic and may be changed by the bus.

In the cyclic communication, the application will exchange with the DPM the same data stream as in the bus and it is up to the application to map the data to the corresponding PDO objects depending on the configuration.

Acyclic access to the PDO objects has to be handled in the application.

Use the Object Dictionary V3 (ODV3) packets to create and handle the PDO objects. These are described in reference [3].

Note: It is mandatory to create the necessary PDO objects and implement all corresponding access handlings.

3 Overview

3.1 Task structure of the Ethernet POWERLINK Controlled Node stack

The figure below displays the internal structure of the tasks which together represent the Ethernet POWERLINK Controlled Node stack:

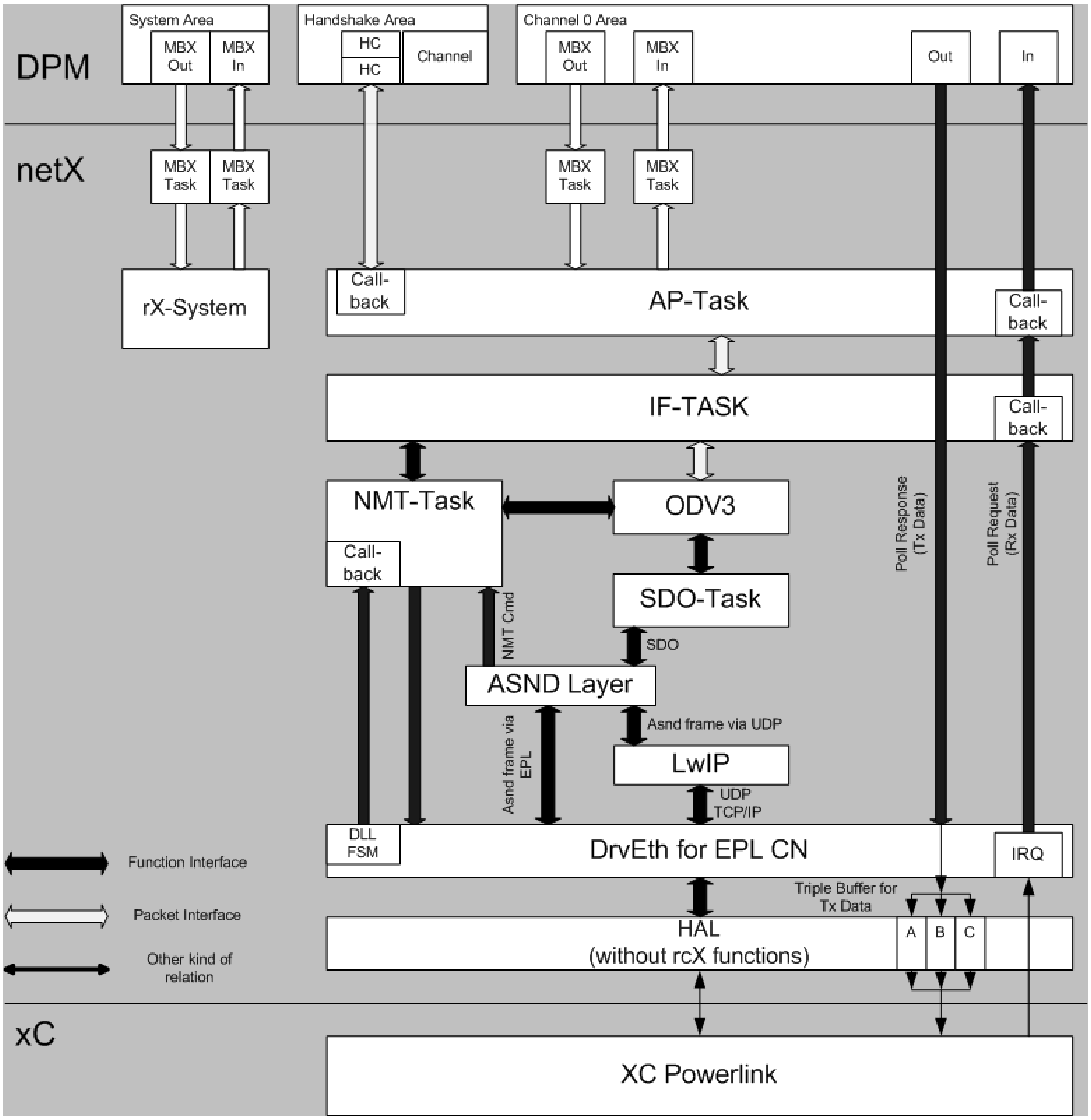

Figure 1: Internal structure of Ethernet POWERLINK Controlled Node stack

The dual-port memory is used to exchange the information such as status, configuration messages, service and process data between netX EPL firmware and host application.

The user application only accesses the task located in the highest layer namely the AP task which constitutes the application interface of the Ethernet POWERLINK Controlled Node stack.

The POWERLINK Controlled Node Stack consists of the following components running on netX core:

- EplCn-AP-Task (DPM handling)
- EplCn-IF-Task (Interface to stack)
- EplCn-NMT-Task (Stack configuration and NMT state machine)
- ODV3 (Object Dictionary module)
- EplCn-SDO-Task (SDO server)
- EplCn-ASnd-Layer (ASnd frames)
- LwIP (TCP/IP stack)
- DrvEth for EPL CN (Interface between stack and HAL. Includes the DLL state machine)

3.1.1 EplCn-AP task

This task is actually present in all stack implementations for netX controllers (using DPM) and has basically the same functionality in all variants. This task handles the interface to DPM, routes the received packets to the IF task, handles the indicators (LED) in accordance with the indications received from the stack and updates the IO data.

3.1.2 EplCn-IF task

This task is responsible for the translation between packets and function interfaces. This task makes the first validations of the received packets before calling the stack API functions or if necessary routes them again to other components of the stack, e.g. ODV3.

3.1.3 EplCn-NMT task

This is the main task of the stack within the high level state machine (NMT state machine). It is responsible for the following:

- Configuration of stack
  - Setting default and user values to the different stack parameters
  - Creation of the default communication OD
- Handling of write and read accesses to the created default OD
- NMT state machine
- Error handling
  - Communication errors e.g. loss of frames, are detected by the DLL state machine and signaled here. The NMT component is responsible for the corresponding handling
  - Vendor or profile specific errors are detected by the application and have also to be signaled to this task.
- Status entries handling
  - If the application wants to send a status entry over the bus, this has to be set in the NMT component. The task will handle the inclusion of this information in the EPL communication
- Handling of the received NMT command from the bus.

3.1.4 ODV3

This is a generic component which is responsible for the management of the Object Dictionary (OD). It provides functionalities to create and handle objects within their index and subindex structures.

3.1.5 EplCn-SDO task

This task handles the access to the local OD from the bus (SDO server) and provides functionalities in order to access the OD of other EPL nodes (SDO client).

3.1.6 EplCn-ASND layer

This is an abstraction layer for the ASnd frames. POWERLINK protocol supports asynchronous frames via normal EPL as well as via UDP. The requested service within these frames may to be handled in different components e.g. NMT commands have to be handled in the NMT task.

The ASnd abstraction layer provides register functionality for specific services as well as functionalities to send own ASnd frames. The ASnd abstraction layer receives the asynchronous frames (both EPL and UDP) and sends the content to the corresponding component.

Note: The application can only register for the manufacturer specific services (range 0xA0.0xFE). Other services are internally handled or not supported.

3.1.7 LwIP

This component receives all IP and ARP frames from the bus and handles them. It provides common IP-based communication functionalities e.g. listening to a specific UDP port or sending of own IP frames.

Note: The local UDP port 3819 is used for POWERLINK specific UDP/IP frames

3.1.8 DrvEth for EPL CN

This component is the lower level of the POWERLINK Controlled Node stack. It is the interface between the high level stack (chip independent) and the HAL (chip specific).

This task is responsible for the following:

- Configuration of the HAL.
- Receiving and handling of frames (IRQ). Route the frames to the correspondent components e. g. PReq frames are sent directly to the AP-Task.
- Handling of the lower level stack state machine (DLL). This state machine handles the communication in the POWERLINK cycle. It tracks the order of the frames received in a cycle and detects errors in the communication. These errors are signaled to the high level of the stack (NMT) for the corresponding handling. Since this state machine has no direct effect on the application, it is not explained in more details in this manual. For more information please refer to reference 2 (Ethernet Powerlink Communication Profile Specification; EPSG DS 301 V1.2.0; 2013, chapter 4.2.4.5 “CN Cycle State Machine”).

3.2 State machine (NMT)

This section explains the general states of the POWERLINK Controlled Node. The current state of the stack is part of the information contained in the EPL frames. The application is also informed about changes of state by the corresponding indications (see section *Current NMT State indication* on page 77)

The NMT state machine can be split in two super states:

- Initialization phase (NMT_GS_INITIALISATION).
- Communicating phase (NMT_GS_COMMUNICATING).

An overview of the stack states is provided in the next subsections. For more information about the NMT state machine please refer to reference 2, chapter 7 “Network Management (NMT)”.

3.2.1 Initialization phase

This is the initialization phase of the stack and corresponds to the super state NMT_GS_INITIALISATION. In this super state, the node is not communicating with the bus yet.

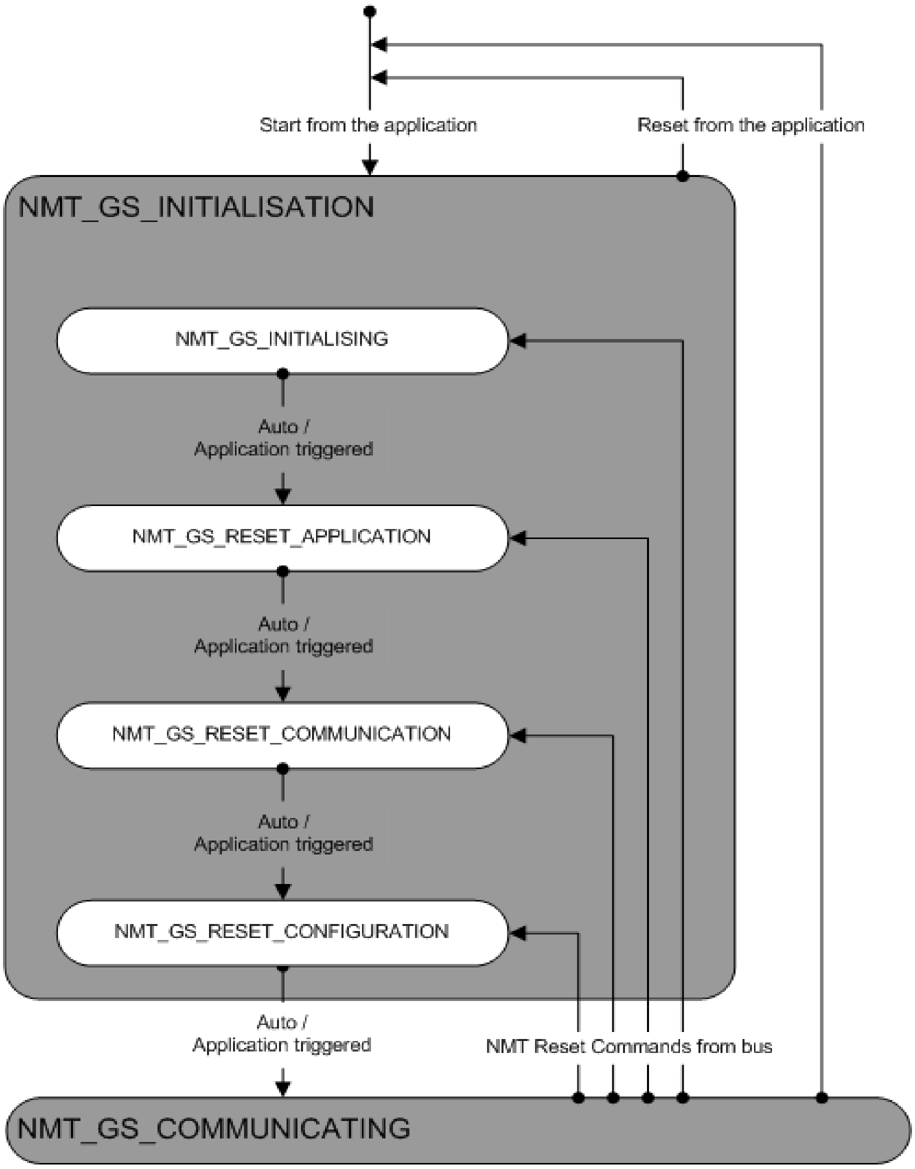

Figure 2: State diagram of the Ethernet POWERLINK Controlled Node - Initialization

The stack enters this phase after receiving the BusOn request from the application. If a reset command is received from the bus while in communicating phase, the stack automatically changes its state to the requested state in the initialization phase.

The stack leaves this phase to change to communicating when the initialization is complete or to start again if the application requests a reset.

The transition between the states of the initialization super state occurs automatically or triggered by application depending on the received configuration (see section *Stack Configuration Flags* on page 41).

In the initialization phase, the POWERLINK Controlled Node stack may be in one of the following states:

- NMT_GS_INITIALISING
- NMT_GS_RESET_APPLICATION
- NMT_GS_RESET_COMMUNICATION
- NMT_GS_RESET_CONFIGURATION

3.2.1.1 NMT_GS_INITIALISING

In this state the stack parameters are initialized with their default or user specific values (from the configuration).

3.2.1.2 NMT_GS_RESET_APPLICATION

In this state the parameters of the manufacturer-specific profile and of the standardized device profile are set to their Power On values (Set values in OD).

3.2.1.3 NMT_GS_RESET_COMMUNICATION

In this state the parameters of the communication profile are set to their Power On values (Set values in OD).

3.2.1.4 NMT_GS_RESET_CONFIGURATION

Until this point the whole configuration was stored in the OD. In the last state of the initialization phase, the active node configuration is generated based on the parameter set stored within the object dictionary.

Now the stack is configured and the communication over the bus can begin.

3.2.2 Communication phase

This corresponds to the super state (NMT_GS_COMMUNICATING). When entering this super state, the node starts the communication over the bus. At this point, the handling of received frames begins.

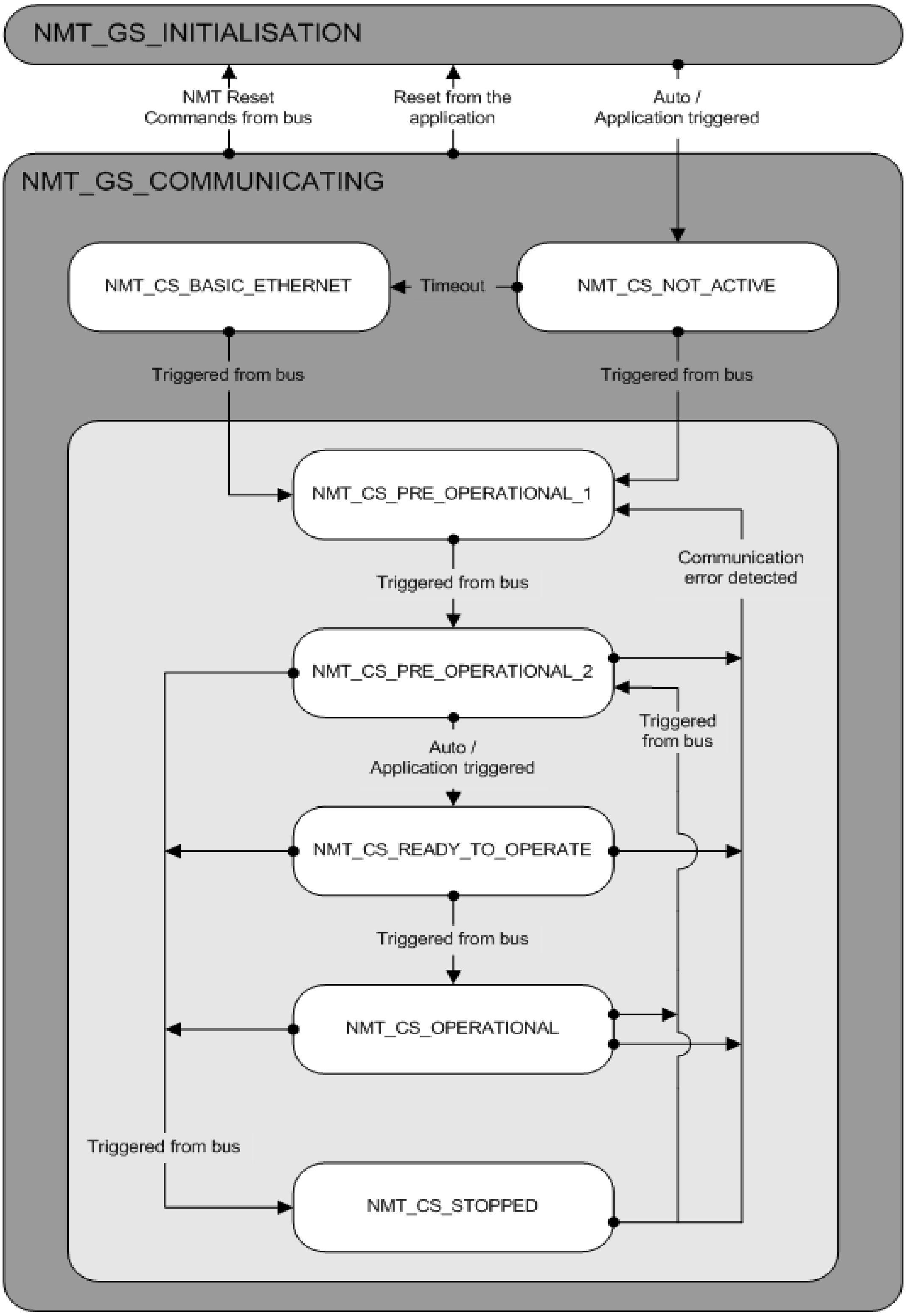

Figure 3: State diagram of the Ethernet POWERLINK Controlled Node - Communicating

The stack enters this phase after the initialization phase is completed.

The stack leaves this phase when a reset command from the bus or the application is received.

In the communicating phase, the POWERLINK Controlled Node stack may be in one of the following states:

- NMT_CS_NOT_ACTIVE
- NMT_CS_BASIC_ETHERNET
- NMT_CS_PRE_OPERATIONAL_1
- NMT_CS_PRE_OPERATIONAL_2
- NMT_CS_READY_TO_OPERATE
- NMT_CS_OPERATIONAL
- NMT_CS_STOPPED

3.2.2.1 NMT_CS_NOT_ACTIVE

This is the first state after the initialization phase. The node begins tracking the bus and analyzing the received frames. If an EPL frame has been received the stack switches to NMT_CS_PRE_OPERATIONAL_1. If no EPL frames have been received within a defined timeout (Basic Ethernet Timeout), the node switches automatically to NMT_CS_BASIC_ETHERNET.

3.2.2.2 NMT_CS_BASIC_ETHERNET

This state is entered, if after connecting to the bus, the node does not receive any EPL frames within a defined timeout.

In this state, the node works according to IEEE 802.3 and may perform Legacy Ethernet communication. No Ethernet POWERLINK cycle is available in this state.

The node changes its state to NMT_CS_PRE_OPERATIONAL_1 if a valid EPL frame is received.

Note: Only in this state the node is able to send frames autonomously. In all other states, the node needs permissions from the MN to send frames.

3.2.2.3 NMT_CS_PRE_OPERATIONAL_1

This is the first state within valid EPL network communication. Here the MN starts configuring the node. The node proceeds to the next state after detecting a SoC frame on the bus (Start of cycle).

3.2.2.4 NMT_CS_PRE_OPERATIONAL_2

This state is entered, if an Ethernet POWERLINK cycle is detected (SoC frames are received cyclically). In this state the MN continues configuration of the node.

From this moment, the node is part of the EPL cycle and if a communication error is detected e.g. loss of frames, the node returns to NMT_CS_PRE_OPERATIONAL_1. Also from this point, if the command NMT_StopNode is received from the bus, the node immediately goes to NMT_CS_STOPPED.

After the complete configuration has been received from the bus, the MN sends the command NMT_EnableReadyToOperate. The node has to confirm this by changing to the next state. This happens either automatically or triggered by the application depending on the configuration parameters (see section Stack Configuration Flags on page 41).

3.2.2.5 NMT_CS_READY_TO_OPERATE

This state is used to confirm the command NMT_EnableReadyToOperate to the MN. This signals that the node is configured and waiting for order to go to NMT_CS_OPERATIONAL.

This change happens when receiving the command NMT_StartNode.

3.2.2.6 NMT_CS_OPERATIONAL

In this state the node is configured and the cyclic IO data exchange starts.

3.2.2.7 NMT_CS_STOPPED

This state is entered if the command NMT_StopNode is received while in valid EPL cycle. In this state the node continues to be part of a valid EPL cycle, but no valid IO data exchange is performed.

The command NMT_EnterPreoperational2 triggers the return to the state NMT_CS_PRE_OPERATIONAL_2.

3.3 Object Dictionary

This section provides an overview of the Object Dictionary concept and handling. For a detailed description refer to reference 3 Hilscher Gesellschaft für Systemautomation mbH: Object Dictionary V3 API 03.

3.3.1 Definition of the Object Dictionary

The object dictionary is a special area for the storage of parameters, application data and the mapping information between process data and application data (PDO mapping). In order to use CANopen-based device and application profiles in Ethernet POWERLINK Controlled Node, the object dictionary functionality is similar to the one defined in the CANopen standard. Access to the object dictionary is possible via Service Data Objects (SDO) which provide mailbox-based access functionality.

- All CANopen-related data objects are contained in the object dictionary and can be accessed in a standardized manner. You can view the object dictionary as a container for device parameter data structures.

3.3.2 Indexing concept

Indexing is generally done via two values, namely the index and the sub-index. The index governs which function will be performed generally. Indices are ordered in ranges of functionality effective within the same area, see below. The sub-index, however, determines which detailed part of an array, record or structure will be active, i.e. which function will be performed.

3.3.3 General structure of the Object Dictionary

The object dictionary is structured in separate areas. Each area has its own range of permitted index values and its special purpose as defined in the table below:

Index range	Area name	Purpose
0x0000 – 0x0FFF	Data Type Area	Definition and description of data types.
0x1000 – 0x1FFF	Communication Profile Area	Definition of generally applicable variables (communication objects for all devices as defined by CANopen standard DS 301).
0x2000 – 0x5FFF	Manufacturer-specific Area	Definition of manufacturer-specific variables
0x6000 – 0x9FFF	Profile Area	Definition of variables related to a specific profile
0xA000 – 0xFFFF	Reserved Area	This area is reserved for future use

Table 7: General structure of Object Dictionary

3.3.4 Definition of objects

The object dictionary contains descriptions of objects. Each entry of the object dictionary represents the description of one single object. It contains the following object-related information:

- The index of the object
- The object code (classification of type, see explanation below)
- The name of the object
- The data type of the object
- The attributes of the object
- The information whether the object is mandatory or optional.

Index

The index is used for addressing and referencing purposes. It contains the position of the entry within the object dictionary. Complex objects may also be addressed by index and sub-index.

Object code

The following object codes providing different classes of objects may be defined within the object dictionary:

Object code	Object name
0002	DOMAIN
0005	DEFTYPE
0006	DEFSTRUCT
0007	VAR
0008	ARRAY
0009	RECORD

Table 8: Definition of Objects

- A domain can be seen as a large amount of data regardless of its structure, for instance a piece of executable code.
- DEFTYPE contains the definition of a simple data type such as Boolean, unsigned16 or float.
- DEFSTRUCT contains the type definition of a structured data object (i.e. it is composed of parts) such as for instance a record.
- VAR contains a value of a simple data type.
- ARRAY contains a multiple data field of values of the same type.
- RECORD contains a multiple data field of values of different types.

Name

The name component of the entry should give a clear and short textual description of the purpose or function of the object.

Access rights

The attribute indicates access rights (as bit flags) such as read or write access is allowed or prohibited.

Bit flag	Access right if set
0x0001	Read in NMT_CS_PRE_OPERATIONAL_1
0x0002	Read in NMT_CS_PRE_OPERATIONAL_2
0x0004	Read in NMT_CS_READY_TO_OPERATE
0x0008	Read in NMT_CS_OPERATIONAL
0x0010	Read in NMT_CS_STOPPED
0x0020	Read in NMT_CS_BASIC_ETHERNET
0x0040	Read in NMT_CS_NOT_ACTIVE
0x0080	Read during reset of the node
0x00FF	Read in all states
0x0100	Write in NMT_CS_PRE_OPERATIONAL_1
0x0200	Write in NMT_CS_PRE_OPERATIONAL_2
0x0400	Write in NMT_CS_READY_TO_OPERATE
0x0800	Write in NMT_CS_OPERATIONAL
0x1000	Write in NMT_CS_STOPPED
0x2000	Write in NMT_CS_BASIC_ETHERNET
0x4000	Write in NMT_CS_NOT_ACTIVE
0x8000	Write during reset of the node
0xFF00	Write in all states

Table 13: Available access rights for POWERLINK objects

Data types

The following data types are available:

Data type index	Name	Object
0001	BOOLEAN	DEFTYPE
0002	INTEGER8	DEFTYPE
0003	INTEGER16	DEFTYPE
0004	INTEGER32	DEFTYPE
0005	UNSIGNED8	DEFTYPE
0006	UNSIGNED16	DEFTYPE
0007	UNSIGNED32	DEFTYPE
0008	REAL32	DEFTYPE
0009	VISIBLE_STRING	DEFTYPE
000A	OCTET_STRING	DEFTYPE
000B	UNICODE_STRING	DEFTYPE
000C	TIME_OF_DAY	DEFTYPE
000D	TIME_DIFFERENCE	DEFTYPE
000E	Reserved	
000F	DOMAIN	DEFTYPE
0010	INTEGER24	DEFTYPE
0011	REAL64	DEFTYPE
0012	INTEGER40	DEFTYPE
0013	INTEGER48	DEFTYPE
0014	INTEGER56	DEFTYPE
0015	INTEGER64	DEFTYPE
0016	UNSIGNED24	DEFTYPE
0017	Reserved	
0018	UNSIGNED40	DEFTYPE
0019	UNSIGNED48	DEFTYPE
001A	UNSIGNED56	DEFTYPE
001B	UNSIGNED64	DEFTYPE
001C-0022	Reserved for future use	
0023	IDENTITY	DEFSTRUCT
0024-003F	Reserved	
0040-005F	Manufacturer Specific Complex Data Types	DEFSTRUCT
0060-007F	Device Profile 0 Specific Standard Data Types	DEFTYPE
0080-009F	Device Profile 0 Specific Complex Data Types	DEFSTRUCT
00A0-00BF	Device Profile 1 Specific Standard Data Types	DEFTYPE
00C0-00DF	Device Profile 1 Specific Complex Data Types	DEFSTRUCT

00E0-00FF	Device Profile 2 Specific Standard Data Types	DEFTYPE
0100-011F	Device Profile 2 Specific Complex Data Types	DEFSTRUCT
0120-013F	Device Profile 3 Specific Standard Data Types	DEFTYPE
0140-015F	Device Profile 3 Specific Complex Data Types	DEFSTRUCT
0160-017F	Device Profile 4 Specific Standard Data Types	DEFTYPE
0180-019F	Device Profile 4 Specific Complex Data Types	DEFSTRUCT
01A0-01BF	Device Profile 5 Specific Standard Data Types	DEFTYPE
01C0-01DF	Device Profile 5 Specific Complex Data Types	DEFSTRUCT
01E0-01FF	Device Profile 6 Specific Standard Data Types	DEFTYPE
0100-021F	Device Profile 6 Specific Complex Data Types	DEFSTRUCT
0220-023F	Device Profile 7 Specific Standard Data Types	DEFTYPE
0240-025F	Device Profile 7 Specific Complex Data Types	DEFSTRUCT
0260-0400	Reserved	Reserved
0401	MAC_ADDRESS	DEFTYPE
0402	IP_ADDRESS	DEFTYPE
0403-041F	Reserved	Reserved
0420	PDO_CommParam_Record_TYPE	DEFSTRUCT
0422	SDO_ParameterRecord_TYPE	DEFSTRUCT
0423	Reserved	Reserved
0424	DLL_ErrorCntRec_TYPE	DEFSTRUCT
0425	NWL_IpGroup_TYPE	DEFSTRUCT
0426	NWL_IpAddrTable_TYPE	DEFSTRUCT
0427-0428	Reserved	Reserved
0429	NMT_ParameterStorage_TYPE	DEFSTRUCT
042A	Reserved	Reserved
042B	NMT_InterfaceGroup_Xh_TYPE	DEFSTRUCT
042C	NMT_CycleTiming_TYPE	DEFSTRUCT
042E-0434	Reserved	Reserved
0435	CFM_VerifyConfiguration_TYPE	DEFSTRUCT
0436-0438	Reserved	Reserved
0439	NMT_EPLNodeID_TYPE	DEFSTRUCT
043A-0FFF	Reserved	Reserved

Table 9: Available Data Type Definitions – Part 2

Further description details of these data types can be found in the Ethernet Powerlink specification (reference 2, chapter 6.1.)^

3.3.5 Accessing the Object Dictionary

The Object Dictionary module (ODV3) provides its own packet API set. This is described in reference [3].

3.3.6 Object Dictionary entries (Communication Profile Area)

According to the Ethernet Powerlink standard, the Communication Profile Area located at the index range from 0x1000 to 0x1FFFh contains the communication specific parameters for the entire network. These entries are common for all devices.

The following table provides an overview of the relevant objects of the Communication Profile Area for a Powerlink Controlled Node, which can be supported by the Hilscher's stack. The table also shows whenever an object is directly allocated and handled by the stack. Some of those objects e.g. the identity object get their content from the configuration set sent by the application:

Object Dictionary entries for Communication Profile					
Data type index	**Object**	**Name**	**Type**	**M/O/C**	**Created and handled by stack**
1000	VAR	Device type	UNSIGNED32	M	Yes
1001	VAR	Error register	UNSIGNED8	M	Yes
1003	ARRAY	Error history	DOMAIN	O	Yes
1006	VAR	Communication cycle period	UNSIGNED32	M	Yes
1008	VAR	Manufacturer Device Name	VISIBLE_STRING	O	No
1009	VAR	Manufacturer Hardware Version	VISIBLE_STRING	O	No
100A	VAR	Manufacturer Software Version	VISIBLE_STRING	O	No
1010	RECORD	Store parameters	DEFSTRUCT	O	No
1011	RECORD	Restore default parameters	DEFSTRUCT	O	No
1016	ARRAY	Consumer heartbeat time	UNSIGNED32	O	No
1018	RECORD	Identity object	DEFSTRUCT	M	Yes
1020	RECORD	Verify configuration	DEFSTRUCT	M	Yes
1021	VAR	Store device description file	DOMAIN	O	No
1022	VAR	Store device description format	UNSIGNED16	C[1]	No
::::	::::	::::	::::	::::	::::
1030	RECORD	Interface group 0	DEFSTRUCT	M	Yes
1031	RECORD	Interface group 1	DEFSTRUCT	O[2]	No
::::	::::	::::	::::	::::	::::

[1] Object 0x1022 is mandatory to be created if the object 0x1021 is also created.

[2] Object 0x1031...0x1039 should not be implemented since the Hilscher EPL CN stack supports and handles only one network interface.

Object Dictionary entries for Communication Profile					
Data type index	**Object**	**Name**	**Type**	**M/O/C**	**Created and handled by stack**
1039	RECORD	Interface group 9	DEFSTRUCT	O	No
1101	RECORD	NMT telegrams counter	DEFSTRUCT	O	No
1102	RECORD	Error statistics	DEFSTRUCT	O	No
::::	::::	::::	::::	::::	::::
1200	RECORD	SDO Server container parameter 0	DEFSTRUCT	O	No
::::	::::	::::		::::	::::
127F	RECORD	SDO Server container parameter 127	DEFSTRUCT	O	No
1280	RECORD	SDO Client container parameter 0	DEFSTRUCT	O	No
::::	::::	::::		::::	::::
12FF	RECORD	SDO Client container parameter 127	DEFSTRUCT	O	No
1300	VAR	SDO sequence layer timeout	UNSIGNED32	M	Yes
1301	VAR	SDO command layer timeout	UNSIGNED32	O	No
1302	VAR	SDO sequence layer number of Ack	UNSIGNED32	O	No
1400	RECORD	PDO Rx communication parameters 0	DEFSTRUCT	C[1]	Yes[2]
1600	ARRAY	PDO Rx mapping parameters 0	UNSIGNED64	C[1]	Yes[2]
1800	RECORD	PDO Tx communication parameters 0	DEFSTRUCT	C[3]	Yes[2]
1A00	ARRAY	PDO Tx mapping parameters 0	UNSIGNED64	C[3]	Yes[2]
1C0A	RECORD	DLL CN collision	DEFSTRUCT	O	Yes
1C0B	RECORD	DLL CN loss SoC	DEFSTRUCT	M	Yes
1C0C	RECORD	DLL CN loss SoA	DEFSTRUCT	O	Yes
1C0D	RECORD	DLL CN loss PReq	DEFSTRUCT	O	Yes
1C0E	RECORD	DLL CN SoC Jitter	DEFSTRUCT	O	Yes
1C0F	RECORD	DLL CN CRC error	DEFSTRUCT	O	Yes
1C13	VAR	DLL Cn SoC Jitter Range	UNSIGNED32	O	Yes
1C14	VAR	DLL Cn loss of SoC tolerance	UNSIGNED32	M	Yes
1E40	RECORD	IP address table 0	DEFSTRUCT	C	Yes
1E41	RECORD	IP address table 1	DEFSTRUCT	C[4]	No

[1] Objects 0x1400 and 0x1600 are mandatory if the node supports Rx data

[2] Objects 0x1400, 0x1600, 0x1800 and 0x1A00 may be created by the stack, depending on the configuration received from the application

[3] Objects 0x1800 and 0x1A00 are mandatory if the node supports Tx data

[4] Object 0x1E41...0x1E49 should not be implemented since the Hilscher EPL CN stack supports and handles only one network interface.

Object Dictionary entries for Communication Profile					
Data type index	**Object**	**Name**	**Type**	**M/O/C**	**Created and handled by stack**
::::	::::	::::	::::	::::	::::
1E49	RECORD	IP address table 9	DEFSTRUCT	C	No
1E4A	RECORD	IP group	DEFSTRUCT	C	Yes
1F82	VAR	Feature flags	UNSIGNED32	M	Yes
1F83	VAR	EPL version	UNSIGNED8	M	Yes
1F8C	VAR	Current NMT state	UNSIGNED8	M	Yes
1F93	RECORD	EPL Node Id	DEFSTRUCT	M/C[1]	Yes
1F98	RECORD	Cycle timing	DEFSTRUCT	M	Yes
1F99	VAR	Basic Ethernet timeout	UNSIGNED32	M	Yes
1F9A	VAR	Host name	VISIBLE_STRING	C	Yes
1F9E	VAR	Reset command	UNSIGNED8	M	Yes

Table 10: Communication Profile - General Overview

For more detailed information about the complete communication, there are profile objects and their structures, please refer to Ethernet Powerlink Communication Profile Specification; EPSG DS 301 V1.2.0; 2013 (App. 1), reference [2].

[1] Subindex 3 of object 0x1F93 is only created and handled if the node supports NodeIdBySw (not supported yet)

3.4 Cyclic data communication / PDO

Within all CANopen-based communication systems (such as Ethernet Powerlink), cyclic communication is done by PDOs (Process Data Objects). These PDOs are used to transfer time-critical process data in real-time. PDO data are exchanged exclusively in the cyclic time slot of the Ethernet Powerlink data transmission cycle.

A PDO defines a storage area for data which is cyclically filled up with current data again. These data can be assigned to subobjects of objects stored in the Ethernet Powerlink object dictionary (i.e. addressing by index and subindex is applied). The process of assignment of cyclic data from the PDO to various objects within the object dictionary is denominated as PDO mapping.

In order to practically define a PDO mapping within the object dictionary, you need to know two objects, namely

- The PDO Communication Parameter Object.
- The PDO Mapping Object

Such objects are defined separately for receive and transmit PDO’s, see below.

Note: You can assign 240 receive PDOs (RxPDO) to an Ethernet POWERLINK Controlled Node, but only one transmit PDO (TxPDO).

3.4.1 Configuring a receive PDO

The Ethernet POWERLINK Controlled Node requires the objects 0x1400...0x14FF and 0x1600...0x16FF to configure a receive PDO. These objects contain the following information:

- Source of the data
- Mapping version
- Mapping configuration

Source of the data

The EPL node, from which the data are received, has to be configured in the objects 0x1400...0x14FF in subindex 1. Following values are allowed:

- 0: This received data in this PDO corresponds to the PReq sent from the MN (This is the common setting for object 0x1400).
- Own node ID: Own PRes (Transmit data) will be mapped into this receive PDO.
- Other: Pres from other node ID is mapped into this receive PDO.

Mapping version

This parameter allows checking the validity of the current IO data with the configured mapping.

The mapping version is configured in the object 0x1400...0x14FF in subindex 2.

This parameter is a byte where:

- Bits 4-7 contain the main version number
- Bits 0-3 contain the sub version number

Note: The mapping version is received into PReq and PRes frames. The main part is used to check the validity of the current IO data.

Mapping configuration

The mapping configuration defines how to map the received data into the application objects. This relation is defined in the objects 0x1600...0x16FF in subindices 1...255. Each of these subindices contains a mapping entry in the following format:

Object mapping interpretation of values (UNSIGNED64)					
No. of bits	**63-48**	**47-32**	**31-24**	**23-16**	**15-0**
Name	Length	Offset	Reserved	Sub index	Index
Data type	UNSIGNED16	UNSIGNED16		UNSIGNED8	UNSIGNED16

Table 11: Receive PDO mapping - Object mapping - Interpretation of values

- Index: index of object to be mapped from the receive data
- Subindex: subindex of object to be mapped from the receive data
- Offset: offset of the data into the IO data block (starting at the begin of the PDO area) to be mapped into the application object
- Length: size of mapped data (in Bits)

Note: Before configuring a PDO, this has to be disabled by setting the subindex 0 (“NumberOfEntries”) of object 0x16XX to 0. If the object 0x14XX has to be changed too, its subindex 0 has also to be set to 0 at first. When the configuration is finished, set both subindices again to the correct value.

3.4.2 Configuring transmit PDO

Since an Ethernet POWERLINK Controlled Node only supports one single transmit PDO, the configuration can be made with the objects 0x1800 and 0x1A00 only. These objects contain the following information:

- Mapping version
- Mapping configuration

Note: The parameter NodeId (object 0x1800 subindex 1) is not used in Controlled Nodes, because the transmit data (PRes) are sent as Broadcast frames. The value has to be set to 0.

Mapping version

This parameter allows checking the validity of the current IO data with the configured mapping.

The mapping version is configured in the object 0x1800 in subindex 2.

This parameter is a byte where:

- Bits 4-7 contain the main version number
- Bits 0-3 contain the sub version number

Note: The mapping version is transmitted into PRes frames. The node receiving our data will check the main part of the version to ensure validity of the IO data.

Mapping configuration

The mapping configuration defines how to map the data from the application objects into the transmit data stream. This relation is defined in the object 0x1A00 in subindices 1...255. Each of these subindices contains a mapping entry in the following format:

Object mapping interpretation of values (UNSIGNED64)					
No. of bits	**63-48**	**47-32**	**31-24**	**23-16**	**15-0**
Name	Length	Offset	Reserved	Sub index	Index
Data type	UNSIGNED16	UNSIGNED16		UNSIGNED8	UNSIGNED16

Table 12: Transmit PDO mapping - Object mapping - Interpretation of values

- Index: index of object to be mapped in the transmit data
- Subindex: subindex of object to be mapped in the transmit data
- Offset: offset into the IO data image (starting at the begin of the PDO area) where the data have to be mapped
- Length: size of mapped data (in Bits)

Note: Before configuring a PDO, this has to be disabled by setting the subindex 0 (“NumberOfEntries”) of object 0x1A00 to 0. If the object 0x1800 has to be changed too, its subindex 0 has also to be set to 0 at first. When the configuration is done, set both subindices again to the correct value.

3.5 Acyclic data communication / SDO

Acyclic data communication denominates the transfer of data between participants of the network which is not regularly or periodically repeated, but typically happening only once. Contrary to the situation of cyclic communication, this kind of communication usually deals with not extremely time-critical data or even data of low priority. Such communication is needed for purposes like commands which are to be executed only once, configuration, diagnosis or handling of emergency or error situations.

Within Ethernet Powerlink, acyclic data communication is done by Service Data Objects (SDOs). SDO communication is based on the client-server-model and is used to access the Object Dictionary of an EPL node.

3.5.1 SDO Server

The node receiving an SDO request is the server in the SDO communication.

Since Hilscher's stack uses a separate component for the Object Dictionary (ODV3), the SDO handling on application side in case of a SDO server is simplified to ODV3 handling. The SDO task of the stack receives the requests from the bus and routes them to the ODV3 component. This will send indications to the behavior of the requested object.

With this implementation, the application is abstracted from the SDO frame received from the bus and it has just to handle the ODV3 indications received for its objects.

Note: The application can receive the indications only for the objects created by itself and for which it has been registered. This means, all objects created by the stack will also be handled by the stack. For more information about creation of objects and registration for object access indications refer to reference 3 (document *Hilscher Gesellschaft für Systemautomation mbH:* Object Dictionary V3 API 03).

Lists of objects that are created automatically by the stack can be found in section *Object Dictionary entries (Communication Profile Area)*.

3.6 Diagnosis

The following diagnosis mechanisms are provided by the Ethernet POWERLINK Controlled Node protocol stack:

- Static Error Bit Field
- Status Entry
- Error Entry

More details about the diagnosis handling and the corresponding API can be found in section *Diagnosis* on page 81.

3.6.1 Static Error Bit Field

The Static Error Bit Field provides 64 bit flags with general diagnosis information. The first 8 bits contains the same information as the Error Register Object in the OD (Object 0x1001). The bits 8-15 are reserved and the rest of bits are vendor or profile specific.

3.6.2 Status Entry

The Status Entry is a vendor or profile specific entry to be written by the application.

3.6.3 Error Entry

Depending on its configuration, an Error Entry may contain a communication profile, a vendor specific or a device profile specific error. All errors detected by the stack itself are signaled using this mechanism (communication profile errors). All Error Entries are automatically logged in the Error History object in the OD (Object 0x1003). Additionally, the entries may be signaled in the Emergency Queue of the Status Response frame.

3.7 Commonly used values in packets

3.7.1 Values for identifying NMT states in packets

The following values are used for identifying NMT states in any field and relate to the current state or target state.

Value	Definition in API	Definition in EPSG
0x00	EPL_NMT_GS_OFF	NMT_GS_OFF
0x19	EPL_NMT_GS_INITIALISING	NMT_GS_INITIALISING
0x29	EPL_NMT_GS_RESET_APPLICATION	NMT_GS_RESET_APPLICATION
0x39	EPL_NMT_GS_RESET_COMMUNICATION	NMT_GS_RESET_COMMUNICATION
0x79	EPL_NMT_GS_RESET_CONFIGURATION	NMT_GS_RESET_CONFIGURATION
0x1C	EPL_NMT_CS_NOT_ACTIVE	NMT_CS_NOT_ACTIVE
0x1D	EPL_NMT_CS_PRE_OPERATIONAL_1	NMT_CS_PRE_OPERATIONAL_1
0x5D	EPL_NMT_CS_PRE_OPERATIONAL_2	NMT_CS_PRE_OPERATIONAL_2
0x6D	EPL_NMT_CS_READY_TO_OPERATE	NMT_CS_READY_TO_OPERATE
0xFD	EPL_NMT_CS_OPERATIONAL	NMT_CS_OPERATIONAL
0x4D	EPL_NMT_CS_STOPPED	NMT_CS_STOPPED
0x1E	EPL_NMT_CS_BASIC_ETHERNET	NMT_CS_BASIC_ETHERNET

Table 13: Meaning of bCurrentState and bTargetState

4 Application interface

This chapter defines the application interface of the Ethernet POWERLINK Controlled Node stack.

The following service categories are supported:

- 4.1 Configuring the Ethernet POWERLINK Controlled Node (page 40)
- 4.2 NMT State control (page 68)
- 4.3 Status indications (page 73)
- 4.4 Diagnosis (page 81)

4.1 Configuring the Ethernet POWERLINK Controlled Node

This chapter explains the service and the procedure to configure the Ethernet POWERLINK Controlled Node stack.

The sections *Parameters used in all configuration variants* (page 41), *Common services (All packet configuration variants)* (page 44), and *Common configuration sequence* (page 51) describe the common configuration, which is used in all configuration variants. The rest of section provides specific details for each configuration variant.

The basic configuration method is explained in section *Static mapping configuration with default PDO objects* on page 54. This is the method to be used for applications with a defined static PDO mapping configuration, where the application does not care about the creation of PDO objects and the configuration of PDO mapping.

Applications with static PDO mapping, which need a full control of the PDO layout and mapping, should use the configuration variant described in *Static mapping configuration with user defined PDO objects* (page 57).

Applications with dynamic PDO mapping have to use the configuration method explained in *Dynamic mapping configuration* (page 59). This configuration variant provides a full control about the PDO configuration to the application. A default PDO mapping has to be defined at startup. The MN may request a change of this configuration. In this case, the application receives the corresponding indications and has to signal the new PDO configuration to the stack.

4.1.1 Parameters used in all configuration variants

4.1.1.1 Stack Configuration Flags

The following table provides an overview of the Stack Configuration Flags used in the configuration packet (see *Configure Stack service* on page 44):

Bit	Description
D5-31	Reserved Reserved, set to 0
D4	MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_USE_APP_READY_FOR_PRES_RD_FLAG If set, the firmware uses the DPM Application Ready flag in the communication change of state register for the RD flag (Data valid) of the Poll Response frames. Otherwise, the firmware set the Poll Response data to valid only after the transmit data were exchanged the first time in the DPM.
D3	MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_DISABLE_PDO_MAP_VERS_CHECK If set, the firmware does not check the mapping version received within the Poll Request data from the bus. Otherwise, the check is done and if a mismatch is detected, the corresponding communication error is signaled.
D2	MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_USE_CUSTOM_PDO_OBJ If set, the application creates and handles the PDO objects (Objects 0x1400, 0x1600, 0x1800, 0x1A00 and the user PDO objects) in Object Dictionary itself. Otherwise, the firmware creates the PDO objects automatically. **Note:** For dynamic mapping devices, this flag has to be set.
D1	MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_DISABLE_HOST_TRIGGERED_PREQ_XCHG If set, the Poll Request data (Receive data) will be exchanged with the DPM as soon as they are received from the bus. Otherwise, the PReq data exchange in DPM will be triggered by the host application.
D0	MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_NMT_TRIGGERED_BY_APP If set, some state switches in the NMT state machine will be triggered by the host application. See section *State machine (NMT)* on page 21 for more details about the application controlled state changes. Otherwise, the NMT state machine runs automatically.

Table 14: Value for Stack Configuration Flags

4.1.1.2 Use of Custom Threshold Flags

The following table provides an overview of the values for the parameter `bUseCustomThreshold` in the configuration packet (see *Configure Stack service* on page 44):

Bit	Description
D5-31	Reserved Reserved, set to 0
D4	MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_CRC_ERROR_THRESHOLD If set, the value of ulThresholdCrcError in the configuration packet will be used. Otherwise, the firmware uses the default value described in the Ethernet POWERLINK protocol.
D3	MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_COLLISION_THRESHOLD If set, the value of ulThresholdCollision in the configuration packet will be used. Otherwise, the firmware uses the default value described in the Ethernet POWERLINK protocol.
D2	MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_LOSS_SOA_THRESHOLD If set, the value of ulThresholdLossSoA in the configuration packet will be used. Otherwise, the firmware uses the default value described in the Ethernet POWERLINK protocol.
D1	MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_LOSS_PREQ_THRESHOLD If set, the value of ulThresholdLossPReq in the configuration packet will be used. Otherwise, the firmware uses the default value described in the Ethernet POWERLINK protocol.
D0	MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_LOSS_SOC_THRESHOLD If set, the value of ulThresholdSoC in the configuration packet will be used. Otherwise, the firmware uses the default value described in the Ethernet POWERLINK protocol.

Table 15: Value for use of Custom Threshold flags

4.1.1.3 Feature flags

The Ethernet POWERLINK Feature Flags (Object 0x18F2) describes the optional protocol functionalities supported by the node.

The following table provides a complete overview of the Feature Flags for the EPL communication profile. Some of the flags are set automatically by the stack, since these functionalities are always supported. Others may be set by the application, within the configuration data.

The table below explains the meaning and significance of the single bits of the feature flag set:

Bit	Description	Info
D9-31	Reserved	Reserved
D8	EPL_FEATURE_FLAGS_CONFIGURATION_MANAGER TRUE: Device implements a configuration manager for the storage of configuration in non-volatile memory. (For more information about this feature see section 6.7 “Configuration Management” of reference 2. FALSE: Not supported.	Application must set this flag if implementing a configuration manager.
D7	EPL_FEATURE_FLAGS_NMT_SERVICE_VIA_UDP TRUE: Device supports NMT services (NMT commands) via UDP/IP (NMT services via POWERLINK frames are standard). FALSE: Not supported	Stack set this flag automatically to TRUE, since this is always supported.
D6	EPL_FEATURE_FLAGS_DYNAMIC_PDO_MAPPING TRUE: Device supports dynamic PDO mapping. FALSE: Not supported.	Application must set this flag if implementing a device with dynamic PDO mapping.
D5	EPL_FEATURE_FLAGS_EXTENDED_NMT_STATE_COMMANDS TRUE: Device supports NMT extended state commands. This is used by the MN to address a group of nodes with the same NMT state command. FALSE: Not supported	Stack set this flag automatically to TRUE, since this is always supported.
D3-4	Reserved	Reserved
D2	EPL_FEATURE_FLAGS_SDO_VIA_ASND TRUE: Device supports SDO communication via POWERLINK ASnd frames. FASLE: Not supported	Stack set this flag automatically to TRUE, since this is always supported.
D1	EPL_FEATURE_FLAGS_SDO_VIA_UDP TRUE: Device supports SDO communication via UDP/IP frames. FASLE: Not supported	Stack set this flag automatically to TRUE, since this is always supported.
D0	EPL_FEATURE_FLAGS_ISOCHRONOUS TRUE: Device supports cyclic IO communication and may be accessed isochronously via PReq. FALSE: Asynchronous device without cyclic IO communication. Only acyclic communication is allowed.	Stack set this flag automatically to TRUE, since this is always supported.

Table 16: Meaning of the Feature Flags

4.1.2 Common services (All packet configuration variants)

4.1.2.1 Configure Stack service

This service has to be used by the host application when configuring the stack using the packet interface. This packet is part of the basic packet set and is used in all configuration variants.

The following rules apply for the behavior of the Ethernet POWERLINK Controlled Node stack when receiving the `EPLCN_IF_SET_CONFIG_REQ` command:

- The configuration data is checked for consistency and integrity.
- In case of failure, no data is accepted and the packet is return with error code
- In case of success, the configuration data are stored internally (within the RAM).
- The new configuration data will be activated only after a channel init request (`RCX_CHANNEL_INIT_REQ` / command code 0x2F80).
- This packet does not perform any registration at the stack automatically. Details for registration of the application can be found in section *Registration and deregistration of Status Indications* on page 73.

Packet structure reference

```
/* System flags */
#define MSK_EPLCN_IF_CFG_SYSTEM_FLAGS_APP_CONTROLLED                          0x00000001

/* Stack configuration flags */
#define MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_NMT_TRIGGERED_BY_APP                 0x00000001
#define MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_DISABLE_HOST_TRIGGERED_PREQ_XCHG     0x00000002
#define MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_USE_CUSTOM_PDO_OBJ                   0x00000004
#define MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_DISABLE_PDO_MAP_VERS_CHECK           0x00000008
#define MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_USE_APP_READY_FOR_PRES_RD_FLAG       0x00000010

/* Custom threshold for loss of frames detection */
#define MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_LOSS_SOC_THRESHOLD                     0x00000001
#define MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_LOSS_PREQ_THRESHOLD                    0x00000002
#define MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_LOSS_SOA_THRESHOLD                     0x00000004
#define MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_COLLISION_THRESHOLD                    0x00000008
#define MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_CRC_ERROR_THRESHOLD                    0x00000010
#define MSK_EPLCN_IF_CFG_USE_CUSTOM_TH_SOC_JITTER_THRESHOLD                   0x00000020

typedef struct EPLCN_IF_SET_CONFIG_REQ_DATA_Ttag
{
  TLR_UINT32      ulSystemFlags;
  TLR_UINT32      ulWatchdogTime;
  TLR_UINT32      ulStackCfgFlags;
  TLR_UINT32      ulVendorId;
  TLR_UINT32      ulProductCode;
  TLR_UINT32      ulRevisionNumber;
  TLR_UINT32      ulSerialNumber;
  TLR_UINT32      ulCycleLength;
  TLR_UINT32      ulDeviceType;
  TLR_UINT32      ulFeatureFlags;
  TLR_UINT16      usPReqDataSize;
  TLR_UINT16      usPResDataSize;
  TLR_UINT8       bPReqMappingVersion;
  TLR_UINT8       bPResMappingVersion;
  TLR_UINT16      usMaxPReqDataSize;
  TLR_UINT16      usMaxPResDataSize;
  TLR_UINT8       bNodeId;
  TLR_UINT32      ulGatewayAddress;
  TLR_UINT8       abNodeName[32];
  TLR_UINT8       bNumberOfStatusEntries;
  TLR_UINT8       bUseCustomThreshold;
  TLR_UINT32      ulThresholdLossSoC;
  TLR_UINT32      ulThresholdLossPReq;
  TLR_UINT32      ulThresholdLossSoA;
  TLR_UINT32      ulThresholdCollision;
  TLR_UINT32      ulThresholdCrcError;
  TLR_UINT32      ulMinCycleLength;
  TLR_UINT32      ulThresholdSoCJitter;

  TLR_UINT32       aulReserved[8];
} EPLCN_IF_SET_CONFIG_REQ_DATA_T;

typedef struct EPLCN_IF_SET_CONFIG_REQ_Ttag
{
  TLR_PACKET_HEADER_T             tHead;
  EPLCN_IF_SET_CONFIG_REQ_DATA_T  tData;
} EPLCN_IF_SET_CONFIG_REQ_T;
```


Packet description

Structure EPLCN_IF_SET_CONFIG_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	149	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		See section Status codes / Error codes
ulCmd	UINT32	0xA230	EPLCN_IF_SET_CONFIG_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EPLCN_IF_SET_CONFIG_REQ_DATA_T			
ulSystemFlags	UINT32 (Bit field)	0, 1 Default: 0	System flags area The start of the device can be performed either application controlled or automatically: Automatic (0): Communication with the MN after device start is allowed without “BUS ON”. But the communication will be interrupted if the “BUS ON” flag changes state to 0 Application controlled (1): The channel firmware is forced to wat for the Application Ready flag in the communication change of state register to be set (see section 3.2.5.1 of reference 1). Communication with the MN is only allowed with the “BUS ON” flag. For more information concerning this topic see section 4.4.1 “Controlled or Automatic Start” of reference 1.
ulWatchdogTime	UINT32	0, 20..65535 Default: 1000	DPM Watchdog time (in milliseconds). 0 = Watchdog timer has been switched off
ulStackCfgFlags	UINT32 (Bit field)	0..2^{32}-1 Default: 0	Stack Configuration flags. See subsection *Stack Configuration Flags* on page 41 for details about this parameter.
ulVendorId	UINT32	0..2^{32}-1	Vendor identification. This is an identification number for the manufacturer of an Ethernet POWERLINK device. Vendor IDs are managed by the EPSG (see http://www.ethernet-powerlink.org) This parameter corresponds to object 0x1018 subindex 1 of the Object Dictionary
ulProductCode	UINT32	0..2^{32}-1	Product Code of the device. This is a manufacturer specific ID for the device. This parameter corresponds to object 0x1018 subindex 2 of the Object Dictionary
ulRevisionNumber	UINT32	0..2^{32}-1	Revision number of the device. This parameter corresponds to object 0x1018 subindex 3 of the Object Dictionary

Structure `EPLCN_IF_SET_CONFIG_REQ_T`			Type: Request
ulSerialNumber	UINT32	0..2^{32}-1	Serial number of the device. 0 = Firmware set the serial number stored in the security memory or flash device label (If available) This parameter corresponds to object 0x1018 subindex 4 of the Object Dictionary
ulCycleLength	UINT32	Default: 1000	Default communication cycle time interval in microseconds. Possible value range is limited by the value of NMTCycleTimeMin and NMTCycleTimeMax in the decryption file (XDD) of the device. See section *Device description file (XDD)* on page 103 for more information. 0 = Stack ignores this parameter and set the default value. This value may be changed by the MN. The default value is activated again if the device returns to NMT_GS_RESET_COMMUNICATION state This parameter corresponds to object 0x1006 of the Object Dictionary.
ulDeviceType	UINT32	0..2^{32}-1 Default: 0	Device Type of the device. This parameter describes the type of device and its functionality. LSB = Device Profile Number (0 = no standardized device) MSB = Additional Information This parameter corresponds to object 0x1000 of the Object Dictionary. See reference 2, chapter 7.2.1.1.1 for more information about this parameter.
ulFeatureFlags	UINT32	0..2^{32}-1	Feature flags of the device. See subsection *Feature flags* on page *43* for details about this parameter. This Feature Flags appears also in the object 0x1F82 of the Object Dictionary
usPReqDataSize	UINT16	0..1490	Poll Request data size (Receive data). For static mapping device, this parameter defines the fix size of the receive data. For dynamic mapping device, this parameter defines the default size of the receive data. In this case, the value may be changed by the application or the MN depending on new IO configuration.
usPResDataSize	UINT16	0..1490	Poll Response data size (Transmit data). For static mapping device, this parameter defines the fix size of the transmit data. For dynamic mapping device, this parameter defines the default size of the transmit data. In this case, the value may be changed by the application or the MN depending on new IO configuration.
bPReqMappingVersion	UINT8	0..255	Mapping version of the Poll Request mapping configuration. For static mapping device, this parameter defines the fix mapping configuration version of the receive data. For dynamic mapping device, this parameter defines the version of the default mapping configuration for the receive data. This may be changed, if a new mapping is configured.
bPResMappingVersion	UINT8	0..255	Mapping version of the Poll Response mapping configuration. For static mapping device, this parameter defines the fix mapping configuration version of the transmit data. For dynamic mapping device, this parameter defines the version of the default mapping configuration for the transmit data. This may be changed, if a new mapping is configured.

Structure `EPLCN_IF_SET_CONFIG_REQ_T`			Type: Request
usMaxPReqDataSize	UINT16	0..1490	Maximum Poll Request data size supported by the device. This parameter defines the upper limit for the value of usPReqDataSize
usMaxPResDataSize	UINT16	0..1490	Maximum Poll Response data size supported by the device. This parameter defines the upper limit for the value of usPResDataSize
bNodeId	UINT8	1..239	Node ID of the device. This parameter defines the node address of the device. The IP address of the device derives from this value as following: 192.168.100.x, with x = bNodeId
ulGatewayAddress	UINT32	192.168.100.1..192.168.100.254 Default: 192.168.100.254	Default value of the gateway address for the IP communication. 0 = Stack ignores this parameter and set the default value. This value may be changed by the MN. The default value is activated again if the device returns to NMT_GS_RESET_COMMUNICATION state.
abNodeName[32]	UINT8[]		DNS host name of the device for the IP communication. This value may be changed by the MN. The default value is activated again if the device returns to NMT_GS_RESET_COMMUNICATION state. This parameter corresponds to object 0x1F9A of the Object Dictionary.
bNumberOfStatusEntries	UINT8	0..13 Default: 0	Number of Status Entries supported by the device.
bUseCustomThreshold	UINT8 (Bit field)	0..31 Default: 0	Bit flags to define whenever the application defines the threshold level on its own. If the flag of a threshold is set, the corresponding threshold value in the configuration will be set. If the flag of a threshold is not set, the corresponding threshold value in the configuration will be ignored and the stack will set the default value instead.
ulThresholdLossSoC	UINT32	$0..2^{32}$-1 Default: 15	Threshold for Loss of SoC. 0 = Threshold is deactivated This value may be changed by the MN. The default value is activated again if the device returns to NMT_GS_RESET_COMMUNICATION state. More information about the thresholds can be found in section *Error detection* on page 81. This parameter corresponds to object 0x1C0B subindex 3 of the Object Dictionary
ulThresholdLossPReq	UINT32	$0..2^{32}$-1 Default: 15	Threshold for Loss of PReq. 0 = Threshold is deactivated This value may be changed by the MN. The default value is activated again if the device returns to NMT_GS_RESET_COMMUNICATION state More information about the thresholds can be found in section *Error detection* on page 81. This parameter corresponds to object 0x1C0D subindex 3 of the Object Dictionary

Structure EPLCN_IF_SET_CONFIG_REQ_T			Type: Request
ulThresholdLossSoA	UINT32	0..2^{32}-1 Default: 15	Threshold for Loss of SoA. 0 = Threshold is deactivated This value may be changed by the MN. The default value is activated again if the device returns to NMT_GS_RESET_COMMUNICATION state. More information about the thresholds can be found in section *Error detection* on page 81. This parameter corresponds to object 0x1C0C subindex 3 of the Object Dictionary.
ulThresholdCollision	UINT32	0..2^{32}-1 Default: 15	Threshold for Collision errors. 0 = Threshold is deactivated This value may be changed by the MN. The default value is activated again if the device returns to NMT_GS_RESET_COMMUNICATION state. More information about the thresholds can be found in section *Error detection* on page 81. This parameter corresponds to object 0x1C0A subindex 3 of the Object Dictionary.
ulThresholdCrcError	UINT32	0..2^{32}-1 Default: 15	Threshold for CRC errors. 0 = Threshold is deactivated This value may be changed by the MN. The default value is activated again if the device returns to NMT_GS_RESET_COMMUNICATION state. More information about the thresholds can be found in section *Error detection* on page 81. This parameter corresponds to object 0x1C0F subindex 3 of the Object Dictionary.
ulMinCycleLength	UINT32	Default: 0	Minimum cycle length supported by the device in microseconds. This parameter corresponds to the XDD file entry NMTCycleTimeMin. The value is used to check the value range of the parameter ulCycleLength. This parameter allows the user to redefine its own minimum cycle length. The new minimum cycle has to equal or greater than the hardware specific minium cycle time: netX100/500: 200us netX51/52: 200us If the value 0 is configured, the parameter will be ignored and the stack will be configured with the hardware specific values.
ulThresholdSoCJitter	UINT32	0..2^{32}-1 Default: 15	Threshold for SoC Jitter errors. 0 = Threshold is deactivated This value may be changed by the MN. The default value is activated again if the device returns to NMT_GS_RESET_COMMUNICATION state. More information about the thresholds can be found in section *Error detection* on page 81. This parameter corresponds to object 0x1C0E subindex 3 of the Object Dictionary.
aulReserved[8]	UINT32[]		Reserved.

Table 17: EPLCN_IF_SET_CONFIG_REQ – Configure Stack Request

Packet structure reference

```
typedef struct EPLCN_IF_SET_CONFIG_CNF_Ttag
{
  TLR_PACKET_HEADER_T             tHead;
} EPLCN_IF_SET_CONFIG_CNF_T;
```

Packet description

Structure EPLCN_IF_SET_CONFIG_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA231	EPLCN_IF_SET_CONFIG_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 18: EPLCN_IF_SET_CONFIG_CNF – Configure Stack Confirmation

4.1.3 Common configuration sequence

This chapter provides an overview of the common configuration sequence, which is used by all configuration variants. This sequence depends on the start modus of the stack, Auto-Start or Start-by-Application (see *Configure Stack service* on page 44). In the Auto-Start modus, the stack starts automatically after a Channel Init is performed. Since the stack resets the objects in OD after each Channel Init, this modus can only be used for the simplest configuration variant (Static mapping configuration with default PDO objects and without creation of any other user objects). For all other configuration modes, the Start-by-Application mode should be used.

4.1.3.1 Configuration sequence with Auto-start mode

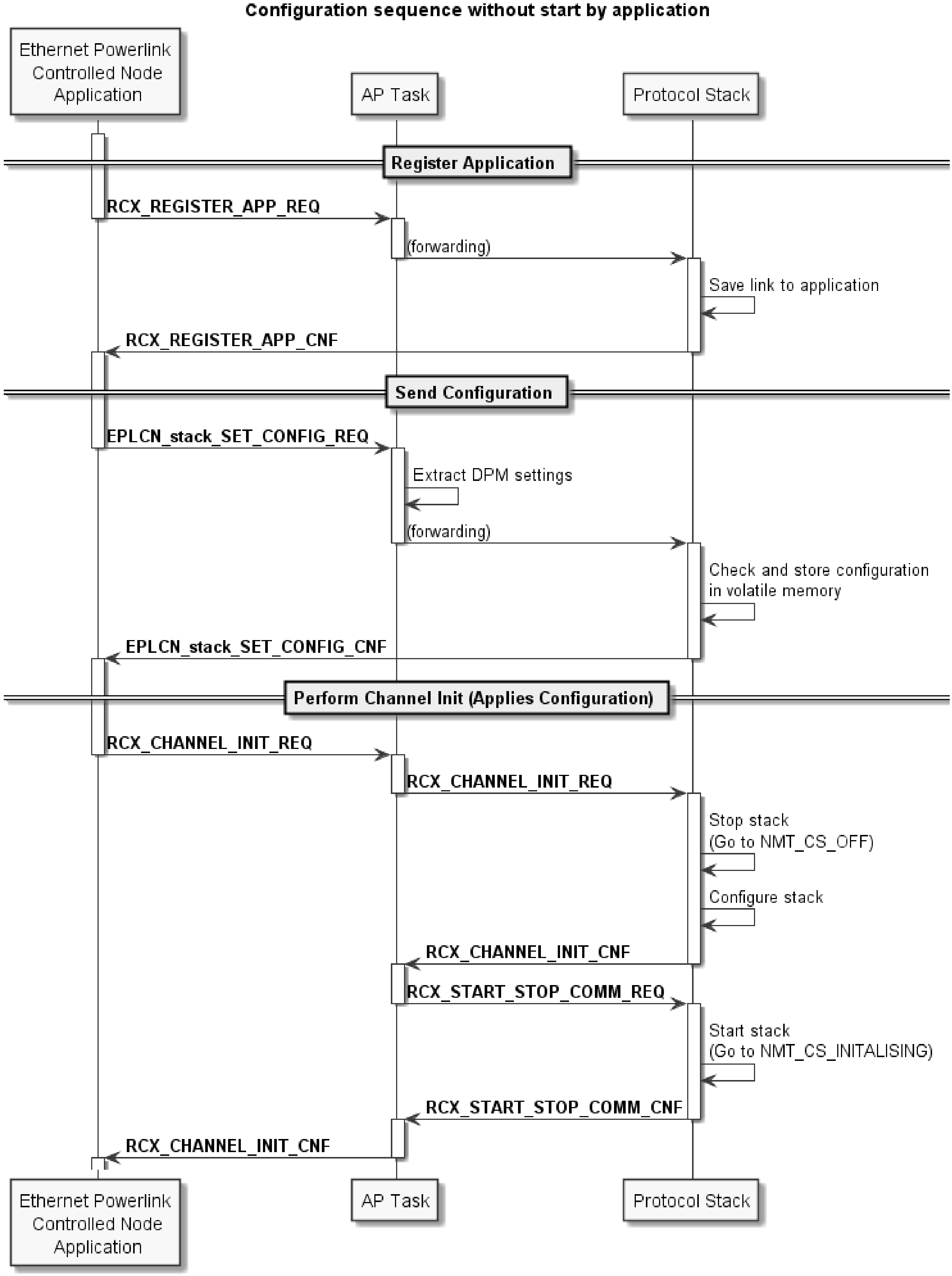

Figure 4: Configuration sequence with Auto-start mode

4.1.3.2 Configuration sequence with Start-by-Application mode

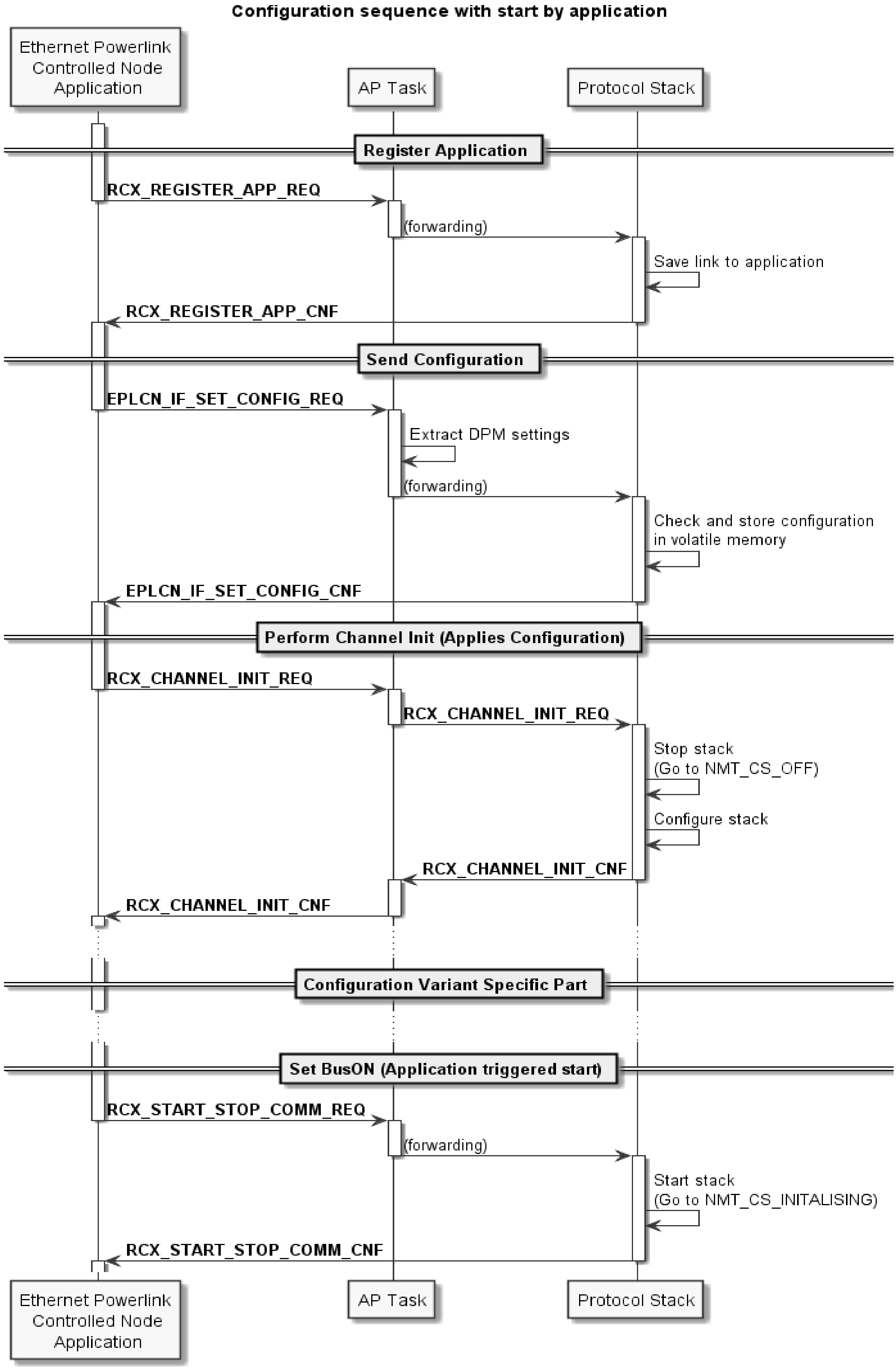

Figure 5: Configuration Sequence with Start-by-Application Mode

4.1.4 Static mapping configuration with default PDO objects

For this configuration variant, only the `EPLCN_IF_SET_CONFIG_REQ` service is needed. Additionally, after performing the Channel Init, the application may create its own objects using the ODV3 interface.

The following sections describe the flags configuration needed for this variant and show the variant's specific configuration sequence. The last section provides an overview about the mechanism, how the stack creates the default PDO objects and defines the mapping configuration.

4.1.4.1 Stack Configuration Flags

In the following table, only the parameters with a fixed value for this configuration variant are displayed. Other parameters follow the common rules.

Bit	Description	Value
D2	MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_USE_CUSTOM_PDO_OBJ	FALSE

Table 19: Stack Configuration Flags for static mapping configuration with default PDO objects

4.1.4.2 Feature Flags

In the following table, only the parameters with a fixed value for this configuration variant are displayed. Other parameters follow the common rules.

Bit	Description	Value
D6	EPL_FEATURE_FLAGS_DYNAMIC_PDO_MAPPING	FALSE

Table 20: Feature Flags for static mapping configuration with default PDO objects

4.1.4.3 Configuration sequence

If no user objects have to be created, the application may use the configuration sequence described in section *Configuration sequence with Auto-start mode* on page 52. Otherwise, the application has to use the sequence from section *Configuration sequence with Start-by-Application mode* on page 53 within following sequence for the configuration variant specific part:

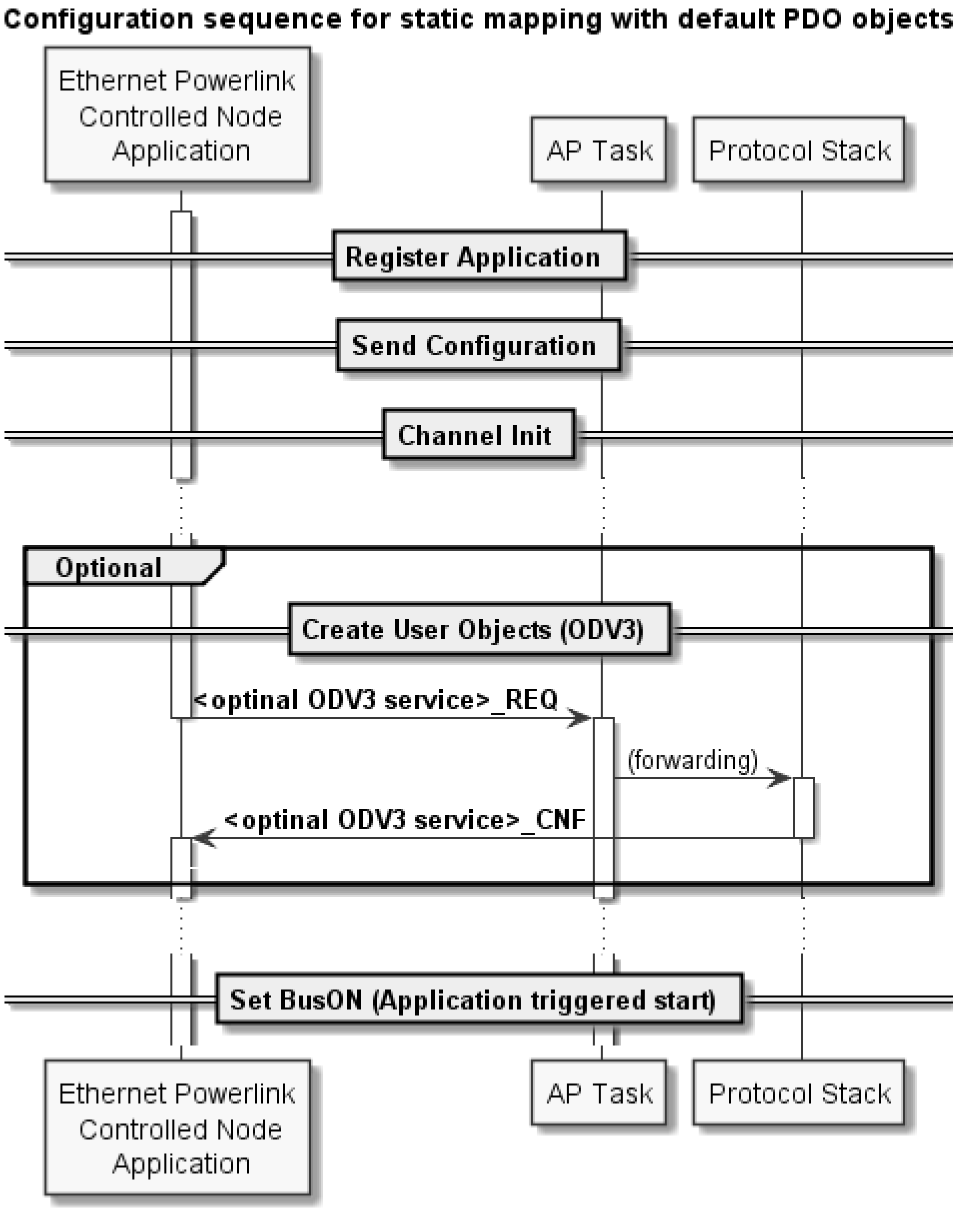

Figure 6: Sequence of static mapping configuration with default PDO objects

4.1.4.4 PDO objects configuration

After the configuration is received, the stack creates default PDO objects and the corresponding mapping objects according the following rules:

- Receive data (Poll Request):
  - The object 0x2000 is created for the cyclic data. The data total bytes are grouped in the minimum subindices of type UNSIGNED64, UNSIGNED32 and UNSIGNED8.
  - The object 0x1600 is created for the mapping configuration. Each subindex of this object maps to the same subindex of the object 0x2000.
  - The object 0x1400 is created for the mapping version information. The mapping version is set to 0.
- Transmit data (Poll Response):
  - The object 0x2100 is created for the cyclic data. The data total bytes are grouped in the minimum subindices of type UNSIGNED64, UNSIGNED32 and UNSIGNED8.
  - The object 0x1A00 is created for the mapping configuration. Each subindex of this object maps to the same subindex of the object 0x2100.
  - The object 0x1800 is created for the mapping version information. The mapping version is set to 0.

Example

Configuring stack with 13 Bytes PReq and 9 Bytes PRes:

Poll request configuration		
Object	**Subindex**	**Description**
0x2000	1	Data type UNSIGNED64. Here the bytes 0...7 of the PReq data are included.
	2	Data type UNSIGNED32. Here the bytes 8...11 of the PReq data are included.
	3	Data type UNSIGNED8. Here is the byte 12 of the PReq data.
0x1600	1	Maps 8 bytes of the PReq data starting by offset 0 to the subindex 1 of 0x2000
	2	Maps 4 bytes of the PReq data starting by offset 8 to the subindex 2 of 0x2000
	3	Maps 1 byte of the PReq data starting by offset 12 to the subindex 3 of 0x2000
0x1400	1	Set to 0.
	2	Set to 0.
Poll response configuration		
Object	**Subindex**	**Description**
0x2100	1	Data type UNSIGNED64. Here the bytes 0..7 of the PRes data are included
	2	Data type UNSIGNED8. Here is the byte 8 of the PRes data.
0x1A00	1	Maps 8 bytes of the PRes data starting by offset 0 to the subindex 1 of 0x21000
	2	Maps 1 byte of the PRes data starting by offset 8 to the subindex 2 of 0x2100
0x1800	1	Set to 0.
	2	Set to 0.

Table 21: Example of PDO default objects configuration

4.1.5 Static mapping configuration with user defined PDO objects

For this configuration variant, additionally to the basic service `EPLCN_IF_SET_CONFIG_REQ`, the user application has to create the complete PDO configuration objects (0x1400, 0x1600, 0x1800, 0x1A00 and the data objects). The application may also create additional user specific objects. Please refer to reference 3 (Object Dictionary V3 manual) for more details about the creation of objects.

Note: This configuration variant supports only the Start-by-Application mode, since the stack resets the objects in OD after each Channel Init.

The following sections describe the flags configuration needed for this variant and show the variant's specific configuration sequence.

4.1.5.1 Stack Configuration Flags

In the following table, only the parameters with a fixed value for this configuration variant are displayed. Other parameters follow the common rules.

Bit	Description	Value
D2	MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_USE_CUSTOM_PDO_OBJ	TRUE

Table 22: Stack Configuration Flags for static mapping configuration with user defined PDO objects

4.1.5.2 Feature Flags

In the following table, only the parameters with a fixed value for this configuration variant are displayed. Other parameters follow the common rules.

Bit	Description	Value
D6	EPL_FEATURE_FLAGS_DYNAMIC_PDO_MAPPING	FALSE

Table 23: Feature Flags for static mapping configuration with user defined PDO objects

4.1.5.3 Configuration sequence

For this configuration variant the sequence described in section *Configuration sequence with Start-by-Application mode* on page 53 within following variant-specific sequence has to be implemented.

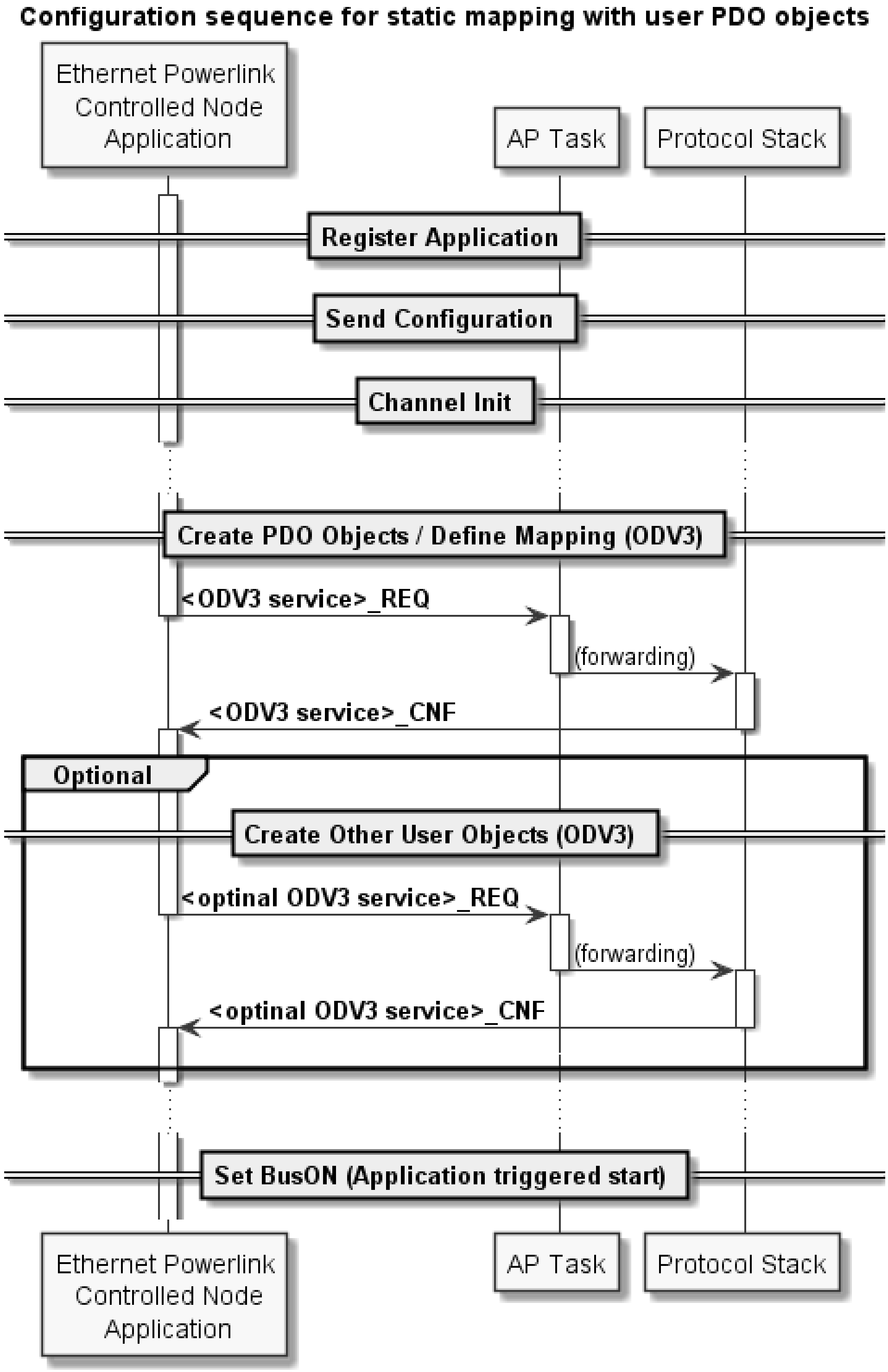

Figure 7: Sequence of static mapping configuration with user defined PDO objects

4.1.6 Dynamic mapping configuration

For this configuration variant, additionally to the basic service `EPLCN_IF_SET_CONFIG_REQ`, the user application has to create the default PDO configuration objects (0x1400, 0x1600, 0x1800, 0x1A00 and the data objects). The application may also create additional user specific objects. Please refer to Object Dictionary V3 manual (reference [3]) for more details about the creation of objects.

Note: This configuration variant supports only the Start-by-Application mode, since the stack resets the objects in OD after each Channel Init.

The default PDO configuration may be changed from the bus. If this happens, the application gets ODV3 specific indications for the write access. After the new mapping configuration is received, the application is now able to the new PDO size and mapping version to the stack by using the `EPLCN_IF_SET_PDO_SIZE_REQ` service.

The following sections describe the flags configuration needed for this variant and show the variant's specific configuration sequence. Additionally, the sections *Set PDO size service* (page 61) and *Set PDO Size sequence* (page 64) provide detailed description of the `EPLCN_IF_SET_PDO_SIZE_REQ` service within its sequence diagram.

4.1.6.1 Stack Configuration Flags

In the following table, only the parameters with a fixed value for this configuration variant are displayed. Other parameters follow the common rules.

Bit	Description	Value
D2	MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_USE_CUSTOM_PDO_OBJ	TRUE

Table 24: Stack Configuration Flags for dynamic mapping configuration

4.1.6.2 Feature Flags

In the following table, only the parameters with a fixed value for this configuration variant are displayed. Other parameters follow the common rules.

Bit	Description	Value
D6	EPL_FEATURE_FLAGS_DYNAMIC_PDO_MAPPING	TRUE

Table 25: Feature Flags for dynamic mapping configuration

4.1.6.3 Configuration sequence

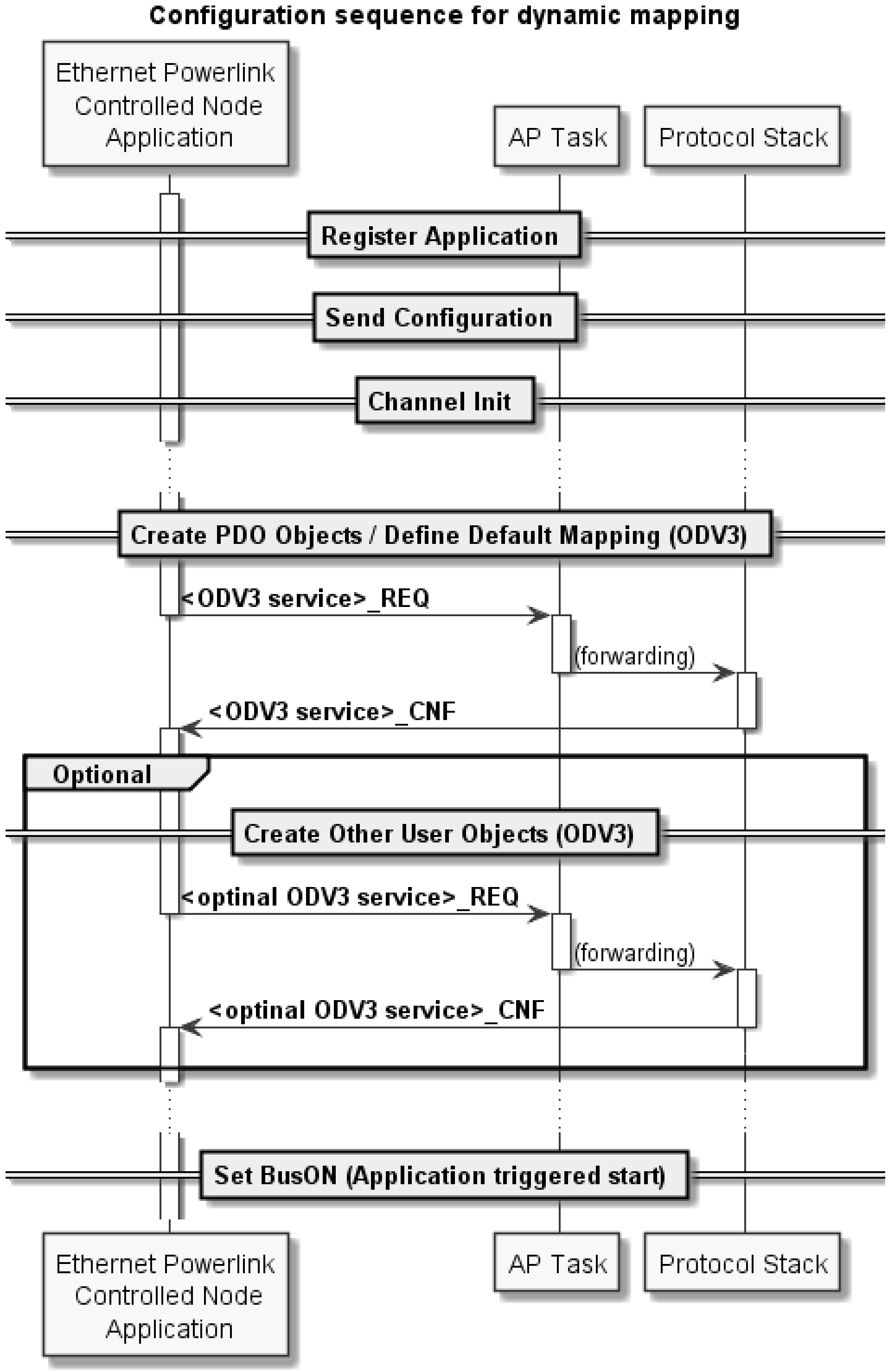

Figure 8: Sequence of dynamic mapping configuration

4.1.6.4 Set PDO size service

This service has to be used in dynamic PDO mapping devices to set the new PDO size and mapping version. This is needed after a new mapping configuration is received from the bus and is done in following steps:

- The application receives ODV3 write indications for the objects 0x1400, 0x1600, 0x1800 and 0x1A00.
- The application calculates the new PDO size
- The application sets the new PDO size and mapping version to the stack
- The new settings are now stored in the stack, but are only made active after a configuration reset command received from the bus

See next section for detailed sequence diagram of this service.

Packet structure reference

```
typedef struct EPLCN_IF_SET_PDO_SIZE_REQ_DATA_Ttag
{
  /** Poll Request data size (range 0 to 1490) */
  TLR_UINT16      usPReqDataSize;
  /** Poll Response data size (range 0 to 1490) */
  TLR_UINT16      usPResDataSize;
  /** PReq Mapping Version */
  TLR_UINT8       bPReqMappingVersion;
  /** PRes Mapping Version */
  TLR_UINT8       bPResMappingVersion;
} EPLCN_IF_SET_PDO_SIZE_REQ_DATA_T;

typedef struct EPLCN_IF_SET_PDO_SIZE_REQ_Ttag
{
  TLR_PACKET_HEADER_T               tHead;
  EPLCN_IF_SET_PDO_SIZE_REQ_DATA_T  tData;
} EPLCN_IF_SET_PDO_SIZE_REQ_T;
```

Packet description

Structure EPLCN_IF_SET_PDO_SIZE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	6	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		See section Status codes / Error codes
ulCmd	UINT32	0xA232	EPLCN_IF_SET_PDO_SIZE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EPLCN_IF_SET_PDO_SIZE_REQ_DATA_T			
usPReqDataSize	UINT16	0..1490	New Poll Request data size (Receive data).
usPResDataSize	UINT16	0..1490	New Poll Response data size (Transmit data).
bPReqMappingVersion	UINT8	0..255	Mapping version of the new Poll Request mapping configuration.
bPResMappingVersion	UINT8	0..255	Mapping version of the new Poll Response mapping configuration.

Table 26: EPLCN_IF_SET_PDO_SIZE_REQ – Set PDO Size request

Packet structure reference

```
typedef struct EPLCN_IF_SET_PDO_SIZE_CNF_Ttag
{
  TLR_PACKET_HEADER_T               tHead;
} EPLCN_IF_SET_PDO_SIZE_CNF_T;
```

Packet description

Structure EPLCN_IF_SET_PDO_SIZE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA233	EPLCN_IF_SET_PDO_SIZE_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 27: EPLCN_IF_SET_PDO_SIZE_CNF – Set PDO Size confirmation

4.1.6.5 Set PDO Size sequence

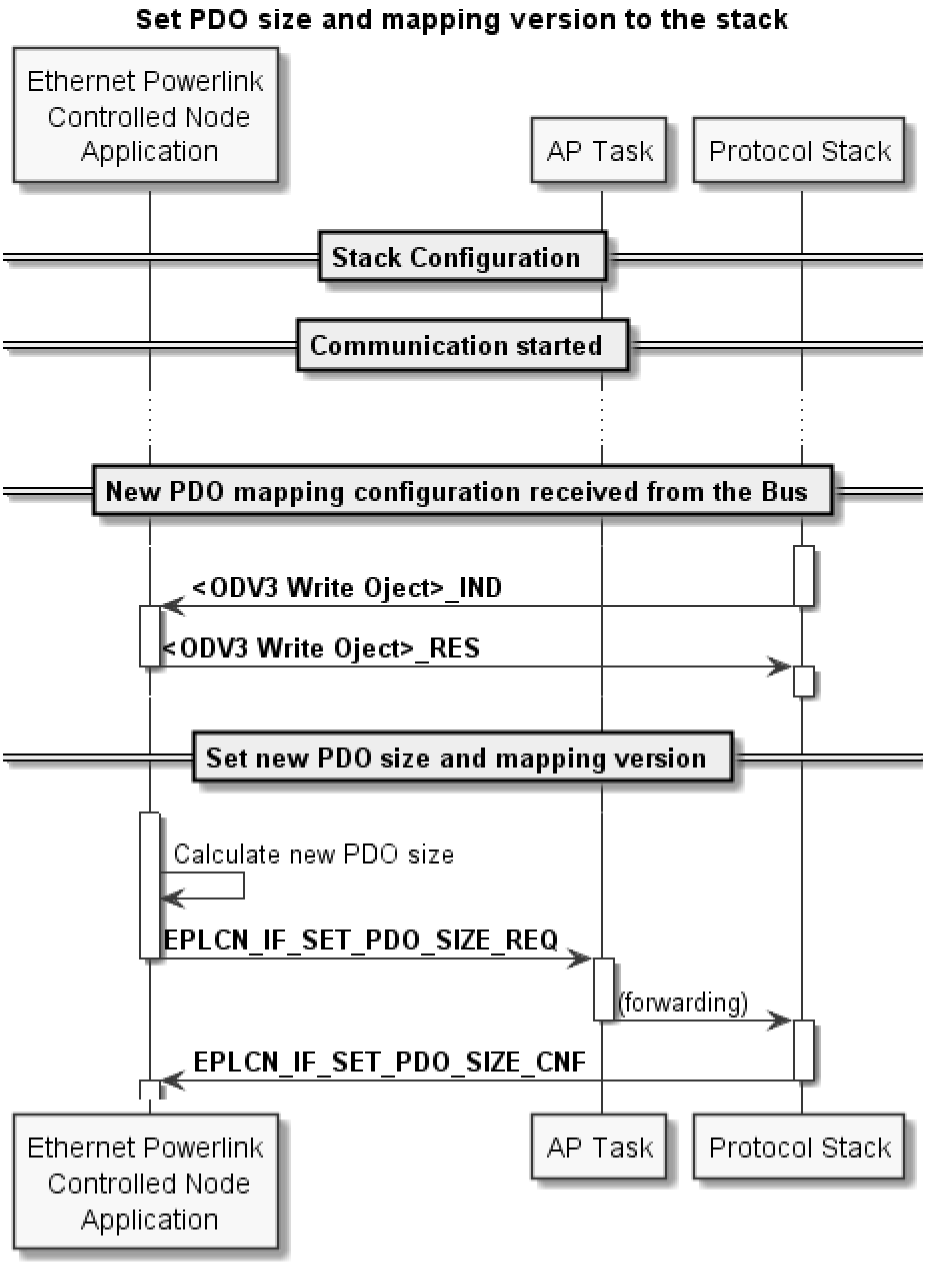

Figure 9: Sequence of Set PDO Size service

4.1.7 Optional Services

This chapter describes the optional services provided by the stack.

4.1.7.1 Configure NodeId

This service allows the application to reconfigure the NodeId without sending a new SetConfiguration packet. The value of the new NodeId will be activated after the device goes through the state NMT_GS_RESET_CONFIGURATION again. This is triggered by following events:

- NMT reset commands
- ChannelInit request packet
- Setting BusOff and BusOn with the StartStopCommunication packet.

Use-case example

Device provides DIP-Switch to configure the NodeId. After the CN was configured and is running, the user may set a new NodeId over the switch. In this case using this service allows setting the new NodeId to stack without going through the complete configuration process again.

Packet structure reference

```
typedef struct EPLCN_IF_SET_NODE_ID_REQ_DATA_Ttag
{
  TLR_UINT8   bNodeId;
} EPLCN_IF_SET_NODE_ID_REQ_DATA_T;

typedef struct EPLCN_IF_SET_NODE_ID_REQ_Ttag
{
  TLR_PACKET_HEADER_T               tHead;
  EPLCN_IF_SET_NODE_ID_REQ_DATA_T   tData;
} EPLCN_IF_SET_NODE_ID_REQ_T;
```

Packet description

Structure EPLCN_IF_SET_NODE_ID_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	1	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		See section Status codes / Error codes
ulCmd	UINT32	0xA234	EPLCN_IF_SET_NODE_ID_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EPLCN_IF_SET_PDO_SIZE_REQ_DATA_T			
bNodeId	UINT8	1..239	New NodeId value. This value will be activate while in NMT_GS_RESET_CONFIGURATION

Table 28: EPLCN_IF_SET_NODE_ID_REQ – Set NodeId request

Packet structure reference

```
typedef struct EPLCN_IF_SET_NODE_ID_CNF_Ttag
{
  TLR_PACKET_HEADER_T               tHead;
} EPLCN_IF_SET_NODE_ID_CNF_T;
```

Packet description

Structure EPLCN_IF_SET_NODE_ID_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA235	EPLCN_IF_SET_NODE_ID_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 29: EPLCN_IF_SET_NODE_ID_CNF – Set NodeId confirmation

4.2 NMT State control

4.2.1 Architecture of NMT State control

As described in section *State machine (NMT)* (page 21), following NMT state machine transitions have to be controlled by the application if the corresponding configuration flag is set (see section Stack Configuration Flags on page 41 `MSK_EPLCN_IF_CFG_STACK_CFG_FLAGS_NMT_TRIGGERED_BY_APP`):

Source NMT State	Target NMT State
NMT_GS_INITIALISING	NMT_GS_RESET_APPLICATION
NMT_GS_RESET_APPLICATION	NMT_GS_RESET_COMMUNICATION
NMT_GS_RESET_COMMUNICATION	NMT_GS_RESET_CONFIGURATION
NMT_GS_RESET_CONFIGURATION	NMT_CS_NOT_ACTIVE
NMT_CS_PRE_OPERATIONAL_2	NMT_CS_READY_TO_OPERATE

Table 30: NMT state transitions controlled by the application

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

In order to determine successful state change and current NMT state, the following service has to be used:

- Current NMT State indication (page 77)

Error Codes related to successful operation

- TLR_S_OK (0x00000000)
  The CN has accepted the new target state and will proceed to it.
- ERR_EPLCN_NMT_INVALID_STATE_CHANGE (0xC0E30001)

  This error may happen when the application triggers a state change which is exclusively triggered by the bus (i.e. Change from NMT_CS_NOT_PRE_OPERATIONAL_1 to NMT_CS_NOT_PRE_OPERATIONAL_2) or an invalid state change (i.e. from NMT_GS_INITIALISING directly to NMT_GS_RESET_CONFIGURATION).

Indications and setting the Target NMT State

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

Handling overview

The sequences in section *Set State sequence* (page 71) provide examples of the handling to be done for the NMT state control.

4.2.2 Services

4.2.2.1 Set NMT State service

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

If the packet has been returned with `ulSta` equal to `TLR_S_OK`, the CN will change its NMT state to the newly requested target state.

In order to determine successful state change and current NMT state, the service *Current NMT State indication* (page 77) has to be used.

Packet structure reference

```
typedef struct  EPLCN_IF_NMT_SET_STATE_REQ_DATA_Ttag
{
  TLR_UINT8   bTargetState;
} EPLCN_IF_NMT_SET_STATE_REQ_DATA_T;

typedef struct  EPLCN_IF_NMT_SET_STATE_REQ_Ttag
{
  TLR_PACKET_HEADER_T                  tHead;
  EPLCN_IF_NMT_SET_STATE_REQ_DATA_T    tData;
} EPLCN_IF_NMT_SET_STATE_REQ_T;
```

Packet description

Structure EPLCN_IF_NMT_SET_STATE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	1	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		See section Status codes / Error codes
ulCmd	UINT32	0xA202	EPLCN_IF_NMT_SET_STATE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EPLCN_IF_NMT_SET_STATE_REQ_DATA_T			
bTargetState	UINT8		Target NMT State. Refer to section *State machine (NMT)* (page 21) for information about the allowed state changes. Refer to section *Values for identifying NMT states in packets* (page 39) for information about coding the NMT state in the Packets

Table 31: EPLCN_IF_NMT_SET_STATE_REQ – Set NMT State request

Packet structure reference

```
typedef struct  EPLCN_IF_NMT_SET_STATE_CNF_Ttag
{
  TLR_PACKET_HEADER_T                  tHead;
} EPLCN_IF_NMT_SET_STATE_CNF_T;
```

Packet description

Structure EPLCN_IF_NMT_SET_STATE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA203	EPLCN_IF_NMT_SET_STATE_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 32: EPLCN_IF_NMT_SET_STATE_CNF – Set NMT State confirmation

4.2.3 Set State sequence

This section provides two sequence examples for the NMT state control handling.

4.2.3.1 Successful state change

The following sequence describes the NMT state control handling for successfully changing to next state by using the indication described in section *Current NMT State indication* (page 77) and the request described in section *Set NMT State service* (page 69).

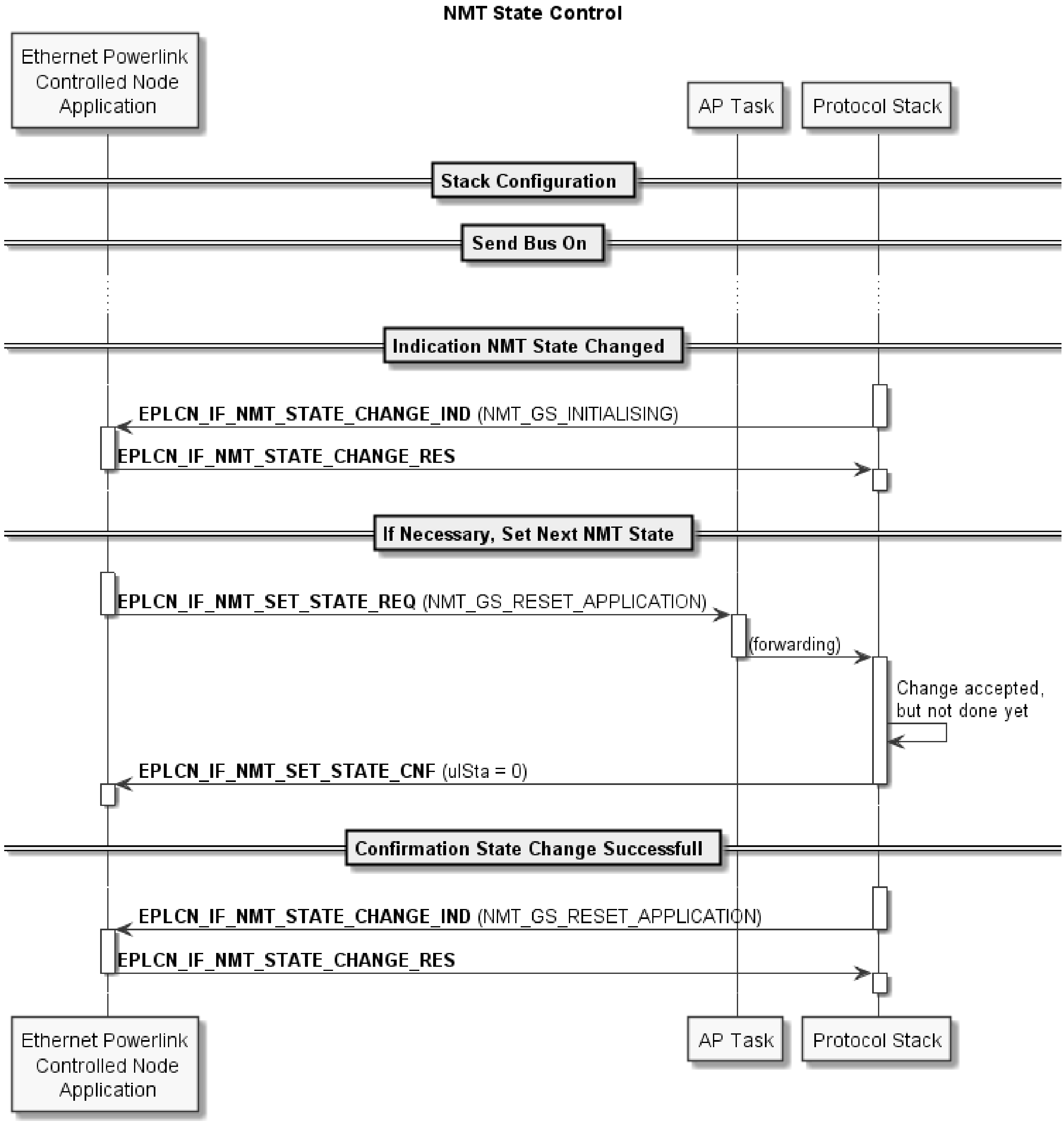

Figure 10: Example of successful NMT state change

4.2.3.2 Erroneous state change

The following sequence shows two examples of an erroneous usage of the NMT state control service described in section *Set NMT State service* on page 69. In the first example, the application tries to skip the state NMT_GS_RESET_APPLICATION, and set the next state. The second example shows the handling in case of trying to set an NMT state which shall not be triggered by the application.

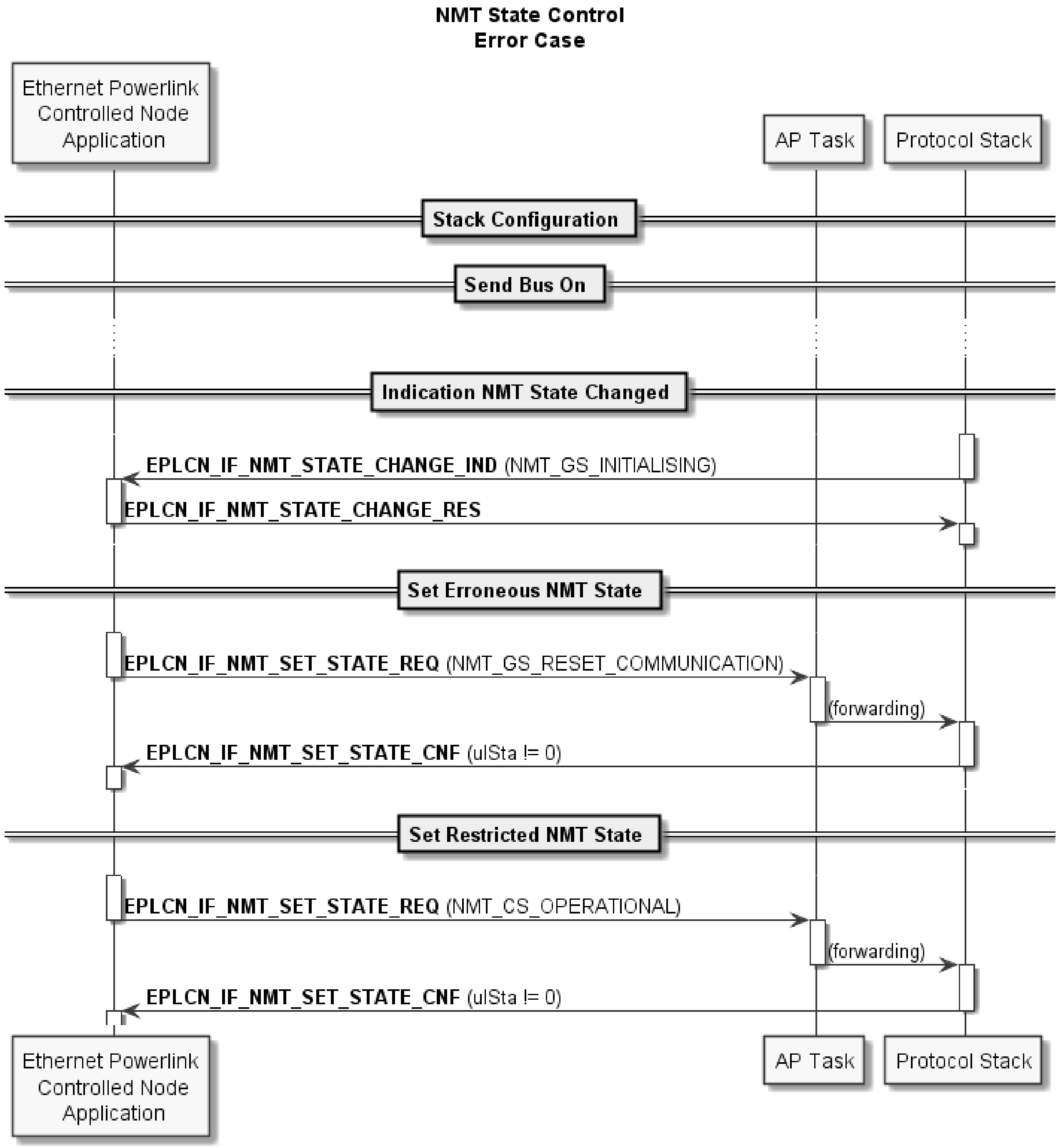

Figure 11: Example of erroneous NMT state change

4.3 Status indications

4.3.1 Registration and deregistration of Status Indications

4.3.1.1 Register for Status Indications service

This packet registers an application task for receiving status indications.

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

If the application does not want to receive those indications anymore, it has to use the service described in subsection *Unregister from Status Indications service* on page 75.

Packet structure reference

```
typedef struct RCX_REGISTER_APP_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} RCX_REGISTER_APP_REQ_T;
```

Packet description

Structure RCX_REGISTER_APP_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0x2F10	RCX_REGISTER_APP_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 33: RCX_REGISTER_APP_REQ – Register for Status Indications request

Packet structure reference

```
typedef struct RCX_REGISTER_APP_CNF_Ttag
{
  TLR_PACKET_HEADER_T   tHead;
} RCX_REGISTER_APP_CNF_T;
```

Packet description

Structure RCX_REGISTER_APP_CNF_T				**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**	
tHead - Structure TLR_PACKET_HEADER_T				
ulDest	UINT32		Destination queue handle, unchanged	
ulSrc	UINT32		Source queue handle, unchanged	
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	0	Packet Data Length in bytes	
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged	
ulSta	UINT32		See section Status codes / Error codes	
ulCmd	UINT32	0x2F11	RCX_REGISTER_APP_CNF – Command	
ulExt	UINT32	0	Extension, reserved	
ulRout	UINT32	x	Routing information, do not change	

Table 34: RCX_REGISTER_APP_CNF – Register for Status Indications confirmation

4.3.1.2 Unregister from Status Indications service

This packet deregisters an application task from receiving status indications.

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

Packet structure reference

```
typedef struct RCX_UNREGISTER_APP_REQ_Ttag
{
  TLR_PACKET_HEADER_T tHead;
} RCX_UNREGISTER_APP_REQ_T;
```

Packet description

Structure RCX_UNREGISTER_APP_REQ_T				**Type: Request**
Variable	**Type**	**Value / Range**	**Description**	
tHead - Structure TLR_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	0	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		See section Status codes / Error codes	
ulCmd	UINT32	0x2F12	RCX_UNREGISTER_APP_REQ - Command	
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons	
ulRout	UINT32	x	Routing, do not touch	

Table 35: RCX_UNREGISTER_APP_REQ – Unregister for Status Indications request

Packet structure reference

```
typedef struct RCX_UNREGISTER_APP_CNF_Ttag
{
  TLR_PACKET_HEADER_T   tHead;
} RCX_UNREGISTER_APP_CNF_T;
```

Packet description

Structure RCX_UNREGISTER_APP_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
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	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See section Status codes / Error codes
ulCmd	UINT32	0x2F13	RCX_UNREGISTER_APP_CNF – Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	x	Routing information, do not change

Table 36: RCX_UNREGISTER_APP_CNF – Unregister for Status Indications confirmation

4.3.2 Common indications

4.3.2.1 Current NMT State indication

This packet indicates the new NMT state of the stack. For details on how this indication relates to NMT state control, see section *NMT State control* (page 68).

This indication contains also information about the stack indicators (Status and Error LEDs). The application can derive the Status LED state from the current NMT state. The Error LED is indicated explicitly in fErrorLedIsOn. More information about indicators handling for Ethernet POWERLINK can be found in section *LED Status* (page 101).

Packet structure reference

```
typedef struct  EPLCN_IF_NMT_STATE_CHANGE_IND_DATA_Ttag
{
  TLR_UINT8   bCurrentState;
  TLR_UINT8   fErrorLedIsOn;
} EPLCN_IF_NMT_STATE_CHANGE_IND_DATA_T;

typedef struct  EPLCN_IF_NMT_STATE_CHANGE_IND_Ttag
{
  TLR_PACKET_HEADER_T                    tHead;
  EPLCN_IF_NMT_STATE_CHANGE_IND_DATA_T  tData;
} EPLCN_IF_NMT_STATE_CHANGE_IND_T;
```

Packet description

Structure EPLCN_IF_NMT_STATE_CHANGE_IND_T				**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**	
tHead - Structure TLR_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	2	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		See section Status codes / Error codes	
ulCmd	UINT32	0xA200	EPLCN_IF_NMT_STATE_CHANGE_IND - Command	
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons	
ulRout	UINT32	x	Routing, do not touch	
tData - Structure EPLCN_IF_NMT_STATE_CHANGE_IND_DATA_T				
bCurrentState	UINT8		Current NMT State indicated. Refer to section *Values for identifying NMT states in packets* (page 39) for information about coding of the NMT states	
fErrorLedIsOn	UINT8	0,1	Current Error LED state. 0 = Error LED Off 1 = Error LED On	

Table 37: EPLCN_IF_NMT_STATE_CHANGE_IND – Current NMT State indication

Packet structure reference

```
typedef struct  EPLCN_IF_NMT_STATE_CHANGE_RES_Ttag
{
  TLR_PACKET_HEADER_T                  tHead;
} EPLCN_IF_NMT_STATE_CHANGE_RES_T;
```

Packet description

Structure EPLCN_IF_NMT_STATE_CHANGE_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA201	EPLCN_IF_NMT_STATE_CHANGE_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 38: EPLCN_IF_NMT_STATE_CHANGE_RES – Current NMT State response

4.3.2.2 Enable Ready To Operate Command indication

This packet indicates the reception of the NMT command EnableReadyToOperate from the bus and indicates that the MN requests the change of state to NMT_CS_READY_TO_OPERATE. See section *State machine (NMT)* (page 21).

If the NMT state machine is triggered by the application (see section *Stack Configuration Flags* page 41), this command has to be confirmed by set NMT state to NMT_CS_READY_TO_OPERATE.

Packet structure reference

```
typedef struct  EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_IND_Ttag
{
  TLR_PACKET_HEADER_T                    tHead;
} EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_IND_T;
```

Packet description

Structure EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA210	EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 39: EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_IND – NMT Command EnableReadyToOperate indication

Packet structure reference

```
typedef struct  EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_RES_Ttag
{
  TLR_PACKET_HEADER_T                   tHead;
} EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_RES_T;
```

Packet description

Structure EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA211	EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 40: EPLCN_IF_NMT_CMD_ENABLE_RDY_TO_OPERATE_RES – NMT Command EnableReadyToOperate response

4.4 Diagnosis

This section describes the diagnosis mechanisms of Ethernet POWERLINK Controlled Node protocol and the corresponding API provided by the stack.

A detailed description of the diagnosis signaling is described in reference 2 (Ethernet Powerlink Communication Profile Specification; EPSG DS 301 V1.2.0; 2013, section 6.5)

An EPL Controlled Node may provide both communication profile (ref. 2, App. 3.9) and vendor or device profile specific diagnosis information.

The communication profile diagnosis (such as loss of frames errors) are detected and handled directly by the stack.

Vendor or device profile specific diagnosis has to be handled by the user application using the corresponding API functionalities.

Detected diagnosis and errors are described using one or more of the following information blocks:

- Static Error Bit Field
- Error Entry
- Status Entry

The Static Error Bit Field and Status Entry information are automatically set by the stack into the Status Response frame. The Error Entry information is always set into the Error History object (0x1003) of the Object Dictionary and, depending on the configuration of each entry, may be also included in the Status Response frame.

4.4.1 Error detection

This section explains how the different errors / diagnosis may be detected.

4.4.1.1 Communication profile errors

The communication profile errors are detected and handled directly by the stack.

Errors detected by the Data Link Layer (DLL)

In this lower layer of the firmware, the hardware and bus frames errors are detected. The errors detected here are:

- Loss of frames (SoC, SoA, PReq)
- Frame collision
- CRC errors in received frames
- SoC Jitter out of range

These errors may not be signaled each time they are detected, since they use a threshold mechanism. Every time one of these errors is detected a corresponding counter is incremented by 8. When this counter reaches the configured Threshold, the specific communication error is signaled using an Error Entry.

Every elapsed cycle without detection of the error, the counter is decremented by 1.

Note: The user application may configure the error Thresholds level and also deactivated some all of the error detections but the Loss of SoC.

Errors detected by the PDO unit (IF task)

These errors are related to the PDO data validity while in Operational mode. The errors detected by the Hilscher's here are:

- Received PDO are too short
- Mapping version of the received PDO is incorrect

Every time one of these errors is detected a specific communication error is signaled within an Error Entry.

4.4.1.2 Vendor and device profile specific errors / diagnosis

These diagnoses have to be detected and handled by the application using the corresponding API.

4.4.2 Error signaling mechanism

This section contains an overview of the error signaling mechanism provided by the EPL protocol.

4.4.2.1 Static Error Bit Field

The static error bit field is an 8 bytes data block containing general diagnosis information (without specific error codes). This area is included in the Status Response frame.

The following table shows the structure of this diagnosis information:

Byte	Description
6	Contents of the Error Register in OD (Object 0x1001)
7	Reserved
8-13	Errors specific to device profile or vendor

Table 41: Static Error Bit Field

Refer to Write Static Error Bit Field on page 85 for detailed information about the API for the static error bit field.

4.4.2.2 Error Entry

The error entry is used to signal that an error was detected. This entry may contain a communication profile, a vendor specific or a device profile error. All errors signaled with the error entry are set into the Error History object (0x1003). This is done by the stack automatically. Additionally, the error may be written into the Emergency Queue of the Status Response frame.

The structure of the Status Entry is explained in the section *Error / Status Entry format* on page 83.

Refer to section *Write Status Entry service* on page 91 for detailed information about the API for the error entries.

4.4.2.3 Status Entry

The status entry is a vendor specific entry to be written by the application. The stack sends this diagnosis data within the Status Response frame.

The structure of the Status Entry is explained in the section Error / Status Entry format on page 83.

Refer to section *Write Status Entry service* on page 91 for detailed information about the API for the status entries.

4.4.2.4 Error / Status Entry format

The following table shows the structure of an error entry (also used as status entry):

Byte No.	Field	Type
0-1	Entry type	UNSIGNED16
2-3	Error code	UNSIGNED16
4-11	Time stamp	UNSIGNED64
12-19	Additional information	UNSIGNED64

Table 42: Structure of an Error / Status Entry

Entry type

Entry type		
Bit	**Value**	**Description**
D15	**Determines whether it is an Error Entry or a Status Entry**	
	0	Error Entry. The entry will be set in the Error History automatically by the stack
	1	Status Entry. The entry will be set in the corresponding position in the Status Response frame
D14	**This is only checked if it is an Error Entry (D15=0) and determines if the entry should also set in the Status Response frame**	
	0	No. Error Entry appears only in the Error History
	1	Yes. Additionally to the Error History, the Error Entry appears also in the Status Response frame
D13 - D12	**Mode**	
	0	Entry is invalid. This is only internally by the stack for the error signaling mechanism. The user application is not allowed to write an Error or Status Entry with this mode to the stack.
	1	An error occurred and is still active.
	2	An active error was cleared. (Only for Error Entries)
	3	An error occurred (Only for Error Entries)

Entry type		
Bit	**Value**	**Description**
D11 – D0	**Profile. This determines how to decode the value of Error Code field.**	
	0x000	Reserved – do not use!
	0x001	Error Code indicates a vendor-specific code
	0x002	Error Code indicates a communication profile specific error. The possible values are defined in the Ethernet Powerlink specification (reference 2, App. 3.9 “Error Code Constants”).
	0x003 - 0xFFF	Error Code indicates a device profile specific error. The value depends from device profile.

Table 43: Entry type

Error code

This is the error code of the entry. The following table shows the supported and detectable communication errors (Profile = 1 in Entry Type) in the Hilscher’s EPL Controlled Node stack.

Category: Hardware errors (0x816n)	
Name	**Value**
E_DLL_COLLISION_TH	`0x8163`
E_DLL_CRC_TH	`0x8164`

Table 44: Error Entries within Status Response Frames - Hardware errors

Category: Frame size errors (0x821n)	
Name	**Value**
E_PDO_SHORT_RX	`0x8210`
E_PDO_MAP_VERS	`0x8211`

Table 45: Error Entries within Status Response Frames - Frame size errors

Category: Frame errors (0x824n)	
Name	**Value**
E_DLL_JITTER_TH	`0x8235`
E_DLL_LOSS_PREQ_TH	`0x8242`
E_DLL_LOSS_SOA_TH	`0x8244`
E_DLL_LOSS_SOC_TH	`0x8245`

Table 46: Error Entries within Status Response Frames – Loss of frame errors

Time Stamp

The time stamp contains the current SoC net time at exactly the cycle where the event occurred/ the error was detected.

Note: The time stamp for both Status Entries and Error Entries is automatically set in the stack. The application does not need this field.

Additional Info

This area may contain vendor-specific or device profile specific information. For communication profile errors, there is no additional info field.

4.4.3 Write Diagnosis to Stack

This section describes the services provided by the EPL Controlled Node stack to write a diagnosis.

4.4.3.1 Write Static Error Bit Field service

This service has to be used to write one or more bits of the Static Error Bit Field into the Status Response frame.

A successful confirmation means that the new bits' values are stored in non-volatile memory and will be included in the next update of the Status Response frame, when requested by the MN.

Error codes related to successful operation

- TLR_S_OK (0x00000000)
  New values are stored internally and waiting for next update of Status Response frame.
- ERR_EPLCN_NMT_INVALID_STATIC_FIELD_BIT_NUMBER (0xC0E60009)
  Specified bit number is invalid. Invalid Bits are:
  - Bit >= 64: Bit is outer field border. Static Error Bit Field has a length of 8 Bytes.
  - 8 <= Bit <= 15: These bits are reserved.

Packet structure reference

```
typedef struct EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ_DATA_ENTRY_Ttag
{
  TLR_UINT8                     bBitNumber;
  TLR_BOOLEAN8                  fBitValue;
} EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ_DATA_ENTRY_T;

typedef struct EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ_DATA_Ttag
{
  EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ_DATA_ENTRY_T  atEntry[1];
} EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ_DATA_T;

typedef struct EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ_Ttag
{
  TLR_PACKET_HEADER_T                                    tHead;
  EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ_DATA_T  tData;
} EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ_T;
```

Packet description

Structure EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	N * 2	2 = length of one entry N = Number of bits to be specified in the packet
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See section Status codes / Error codes
ulCmd	UINT32	0xA228	EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ_DATA_T			
atEntry[1]	Struct		Entry for one Bit. This entry specifies the bit number and its new value. To specify more than one bit, just write in the next indexes of atEntry. The number of entries is specified with the parameter ulLen.

Table 47: EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_REQ – Write Static Error Bit Field request

Packet structure reference

```
typedef struct EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_CNF_Ttag
{
  TLR_PACKET_HEADER_T                                   tHead;
} EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_CNF_T;
```

Packet description

Structure EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA229	EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 48: EPLCN_IF_NMT_WRITE_STATIC_ERROR_BIT_FIELD_CNF – Write Static Error Bit Field confirmation

4.4.3.2 Write Error Entry service

This packet has to be used to write an Error Entry to the stack. This will be automatically written in the Error History object and, depending on the entry type, will also send within the Status Response frame.

A successful confirmation means that the entry is already set in the Error History. The send of the diagnosis in within Status Response may not happen yet, since this is send only when requested by the MN.

Error Codes related to successful operation

- TLR_S_OK (0x00000000)
  The CN has processed the entry successfully.
- ERR_EPLCN_NMT_ENTRY_TYPE_IS_NOT_ERROR_ENTRY (0xC0E60005)
  Bit D15 of the Entry Type is not 0.

Packet structure reference

```
typedef struct EPLCN_IF_NMT_WRITE_ERROR_ENTRY_REQ_DATA_Ttag
{
  TLR_UINT16        usEntryType;
  TLR_UINT16        usErrorCode;
  TLR_UINT8         abAddInformation[8];
} EPLCN_IF_NMT_WRITE_ERROR_ENTRY_REQ_DATA_T;

typedef struct EPLCN_IF_NMT_WRITE_ERROR_ENTRY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                       tHead;
  EPLCN_IF_NMT_WRITE_ERROR_ENTRY_REQ_DATA_T tData;
} EPLCN_IF_NMT_WRITE_ERROR_ENTRY_REQ_T;
```

Packet description

Structure EPLCN_IF_NMT_WRITE_ERROR_ENTRY_REQ_T				Type: Request
Variable	**Type**	**Value / Range**	**Description**	
tHead - Structure TLR_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		See section Status codes / Error codes	
ulCmd	UINT32	0xA220	EPLCN_IF_NMT_WRITE_ERROR_ENTRY_REQ - Command	
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons	
ulRout	UINT32	x	Routing, do not touch	
tData - Structure EPLCN_IF_NMT_WRITE_ERROR_ENTRY_REQ_DATA_T				
usEntryType	UINT16		Entry Type. Refer to section 4.4.2.4 Error / Status Entry format for information.	
usErrorCode	UINT16		Error Code Refer to section 4.4.2.4 Error / Status Entry format for information.	
abAddInformation [8]	UINT8[]		Additional Information Refer to section 4.4.2.4 Error / Status Entry format for information.	

Table 49: EPLCN_IF_NMT_WRITE_ERROR_ENTRY_REQ – Write Error Entry request

Packet structure reference

```
typedef struct  EPLCN_IF_NMT_WRITE_ERROR_ENTRY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                   tHead;
} EPLCN_IF_NMT_WRITE_ERROR_ENTRY_CNF_T;
```

Packet description

Structure EPLCN_IF_NMT_WRITE_ERROR_ENTRY_CNF_T				Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**	
tHead - Structure TLR_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	0	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		See section Status codes / Error codes	
ulCmd	UINT32	0xA221	EPLCN_IF_NMT_WRITE_ERROR_ENTRY_CNF - Command	
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons	
ulRout	UINT32	x	Routing, do not touch	

Table 50: EPLCN_IF_NMT_WRITE_ERROR_ENTRY_CNF – Write Error Entry confirmation

4.4.3.3 Write Status Entry service

This packet has to be used to write a Status Entry to the stack. This will automatically be written in the Status Response frame in the index specified within the packet and signaled to the MN.

A successful confirmation means that the entry is stored in non-volatile memory and will be included in the next update of the Status Response frame, when requested by the MN.

Error codes related to successful operation

- TLR_S_OK (0x00000000)
  The CN has processed the entry successfully.
- ERR_EPLCN_NMT_ENTRY_TYPE_IS_NOT_STATUS_ENTRY (0xC0E60006)
  Bit D15 of the Entry Type is not 1.
- ERR_EPLCN_NMT_CONFIGURED_NUM_STATUS_ENTRIES_EXCEEDED (0xC0E60008)
  The value of usStatusEntryNumber exceeds the configured maximum number of status entries (parameter bNumberOfStatusEntries in the `EPLCN_IF_SET_CONFIG_REQ` service).

Packet structure reference

```
typedef struct EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ_DATA_Ttag
{
  TLR_UINT16        usStatusEntryNumber;
  TLR_UINT16        usEntryType;
  TLR_UINT16        usErrorCode;
  TLR_UINT8         abAddInformation[8];
} EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ_DATA_T;

typedef struct EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ_Ttag
{
  TLR_PACKET_HEADER_T                         tHead;
  EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ_DATA_T  tData;
} EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ_T;
```

Packet description

Structure EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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		See section Status codes / Error codes
ulCmd	UINT32	0xA224	EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ_DATA_T			
usStatusEntryNumber	UINT16	0..13	Index in the Status Response frame, where the entry has to appear
usEntryType	UINT16		Entry Type. Refer to section Error / Status Entry format on page 83 for information.
usErrorCode	UINT16		Error Code Refer to section Error / Status Entry format on page 83 for information.
abAddInformation [8]	UINT8[]		Additional Information Refer to section Error / Status Entry format on page 83 for information.

Table 51: EPLCN_IF_NMT_WRITE_STATUS_ENTRY_REQ – Write Status Entry request

Packet structure reference

```
typedef struct  EPLCN_IF_NMT_WRITE_STATUS_ENTRY_CNF_Ttag
{
  TLR_PACKET_HEADER_T                          tHead;
} EPLCN_IF_NMT_WRITE_STATUS_ENTRY_CNF_T;
```

Packet description

Structure EPLCN_IF_NMT_WRITE_STATUS_ENTRY_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA225	EPLCN_IF_NMT_WRITE_STATUS_ENTRY_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 52: EPLCN_IF_NMT_WRITE_STATUS_ENTRY_CNF – Write Status Entry confirmation

4.4.4 Diagnosis indications

This section describes the indications that the application receives from the stack each time a new diagnosis is signaled.

Information about how to register for the stack indications can be found in subsection 4.3.1 “Registration and deregistration of Status Indications”.

4.4.4.1 New Error Entry Indication service

This service indicates that a new Error Entry has been set in the stack. In the service packet, the elements Entry Type, Entry Code and Additional Information of the entry are provided. However there is no information about the component which signaled the error.

Note: The source of the packet is not the component which triggered the error.

Note: An Error Entry set by the application using the Write Error Entry service will also generate an Indication.

This service is just an information service. No reaction for the different errors is specified on stack side. The reaction may be application specific.

Packet structure reference

```
typedef struct EPLCN_IF_NMT_NEW_ERROR_ENTRY_IND_DATA_Ttag
{
  TLR_UINT16        usEntryType;
  TLR_UINT16        usErrorCode;
  TLR_UINT8         abAddInformation[8];
} EPLCN_IF_NMT_NEW_ERROR_ENTRY_IND_DATA_T;

typedef struct EPLCN_IF_NMT_NEW_ERROR_ENTRY_IND_Ttag
{
  TLR_PACKET_HEADER_T                     tHead;
  EPLCN_IF_NMT_NEW_ERROR_ENTRY_IND_DATA_T tData;
} EPLCN_IF_NMT_NEW_ERROR_ENTRY_IND_T;
```

Packet description

Structure EPLCN_IF_NMT_NEW_ERROR_ENTRY_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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		See section Status codes / Error codes
ulCmd	UINT32	0xA222	EPLCN_IF_NMT_NEW_ERROR_ENTRY_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EPLCN_IF_NMT_NEW_ERROR_ENTRY_IND_DATA_T			
usEntryType	UINT16		Entry Type. Refer to section Error / Status Entry format on page 83 for information.
usErrorCode	UINT16		Error Code Refer to section Error / Status Entry format on page 83 for information.
abAddInformation [8]	UINT8[]		Additional Information Refer to section *Error / Status Entry format* on page 83 for information.

Table 53: EPLCN_IF_NMT_NEW_ERROR_ENTRY_IND – New Error Entry indication

Packet structure reference

```
typedef struct  EPLCN_IF_NMT_NEW_ERROR_ENTRY_RES_Ttag
{
  TLR_PACKET_HEADER_T                     tHead;
} EPLCN_IF_NMT_NEW_ERROR_ENTRY_RES_T;
```

Packet description

Structure EPLCN_IF_NMT_NEW_ERROR_ENTRY_RES_T				**Type: Response**
Variable	**Type**	**Value / Range**	**Description**	
tHead - Structure TLR_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	0	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		See section Status codes / Error codes	
ulCmd	UINT32	0xA223	EPLCN_IF_NMT_NEW_ERROR_ENTRY_RES - Command	
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons	
ulRout	UINT32	x	Routing, do not touch	

Table 54: EPLCN_IF_NMT_NEW_ERROR_ENTRY_RES – New Error Entry response

4.4.4.2 New Status Entry Indication service

This service indicates that a new Status Entry has been set in the stack. In the service packet, the elements Entry Type, Entry Code and Additional Information of the entry are provided. However there is no information about the component which signaled the Status Entry.

Note: The source of the packet is not the component which triggered the error.

Note: An Error Entry set by the application using the Write Status Entry service will also generate an Indication.

This service is just an information service. No reaction for the different errors is specified on stack side. The reaction may be application specific.

Packet structure reference

```
typedef struct EPLCN_IF_NMT_NEW_STATUS_ENTRY_IND_DATA_Ttag
{
  TLR_UINT16        usStatusEntryNumber;
  TLR_UINT16        usEntryType;
  TLR_UINT16        usErrorCode;
  TLR_UINT8         abAddInformation[8];
} EPLCN_IF_NMT_NEW_STATUS_ENTRY_IND_DATA_T;

typedef struct EPLCN_IF_NMT_NEW_STATUS_ENTRY_IND_Ttag
{
  TLR_PACKET_HEADER_T                       tHead;
  EPLCN_IF_NMT_NEW_STATUS_ENTRY_IND_DATA_T  tData;
} EPLCN_IF_NMT_NEW_STATUS_ENTRY_IND_T;
```

Packet description

Structure EPLCN_IF_NMT_NEW_STATUS_ENTRY_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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		See section Status codes / Error codes
ulCmd	UINT32	0xA226	EPLCN_IF_NMT_NEW_STATUS_ENTRY_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EPLCN_IF_NMT_NEW_STATUS_ENTRY_IND_DATA_T			
usStatusEntryNumber	UINT16	0..13	Index, where the status entry appears in the Status Response frame on the bus.
usEntryType	UINT16		Entry Type. Refer to section Error / Status Entry format on page 83 for information.
usErrorCode	UINT16		Error Code Refer to section Error / Status Entry format on page 83 for information.
abAddInformation [8]	UINT8[]		Additional Information Refer to section *Error / Status Entry format* on page 83 for information.

Table 55: EPLCN_IF_NMT_NEW_STATUS_ENTRY_IND – New Status Entry indication

Packet structure reference

```
typedef struct EPLCN_IF_NMT_NEW_STATUS_ENTRY_RES_Ttag
{
  TLR_PACKET_HEADER_T                          tHead;
} EPLCN_IF_NMT_NEW_STATUS_ENTRY_RES_T;
```

Packet description

Structure EPLCN_IF_NMT_NEW_STATUS_ENTRY_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_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	0	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		See section Status codes / Error codes
ulCmd	UINT32	0xA227	EPLCN_IF_NMT_NEW_STATUS_ENTRY_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 56: EPLCN_IF_NMT_NEW_STATUS_ENTRY_RES – New Status Entry response

5 Status codes / Error codes

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0E60001	ERR_EPLCN_NMT_INVALID_STATE_CHANGE The target state change is not allowed in the current state or because this state change is not to be triggered by the application.
0xC0E60002	ERR_EPLCN_NMT_STACK_NOT_CONFIGURED Channel Init or Bus On are not allowed when the stack is not configured.
0xC0E60003	ERR_EPLCN_NMT_NOT_ALLOWED_IN_ACT_STATE Targeted action not allowed in current NMT state.
0xC0E60004	ERR_EPLCN_NMT_MAX_PDO_SIZE_EXCEEDED Configured PDO exceeds the maximum allowed or maximum configured PDO size.
0xC0E60005	ERR_EPLCN_NMT_ENTRY_TYPE_IS_NOT_ERROR_ENTRY The given entry is not an Error Entry (Bit 15 of Entry Type is not 0).
0xC0E60006	ERR_EPLCN_NMT_ENTRY_TYPE_IS_NOT_STATUS_ENTRY The given entry is not a Status Entry (Bit 15 of Entry Type is not 1).
0xC0E60007	ERR_EPLCN_NMT_MAX_NUM_STATUS_ENTRIES_EXCEEDED Value of parameter bNumberOfStatusEntries in the `EPLCN_IF_SET_CONFIG_REQ` service is out of the allowed range (0...13).
0xC0E60008	ERR_EPLCN_NMT_CONFIGURED_NUM_STATUS_ENTRIES_EXCEEDED The value of usStatusEntryNumber in `EPLCN_IF_WRITE_STATUS_ENTRY_REQ` service exceeds the configured maximum number of status entries (parameter bNumberOfStatusEntries in the `EPLCN_IF_SET_CONFIG_REQ` service).
0xC0E60009	ERR_EPLCN_NMT_INVALID_STATIC_FIELD_BIT_NUMBER Specified bit number is invalid in `EPLCN_IF_WRITE_STATIC_ERROR_BIT_FIELD_REQ` service is invalid.
0xC0E6000A	ERR_EPLCN_NMT_FSM_AUTO_RUN_ENABLED Stack is running in Auto Run mode (Bit 0 of parameter ulStackCfgFlags in the `EPLCN_IF_SET_CONFIG_REQ` service is False). State changes cannot be triggered by the application: `EPLCN_IF_NMT_SET_STATE_REQ` service is not allowed.
0xC0E6000B	ERR_EPLCN_NMT_CONFIGURED_CYCLE_LENGTH_TOO_LOW Value of ulCycleLength in the configuration packet is smaller than the minimum cycle length supported by the device. The minimum cycle length is provided by the parameter ulMinCycleLength of the configuration packet packet.
0xC0E6000C	ERR_EPLCN_NMT_CONFIGURED_MIN_CYCLE_LENGTH_TOO_LOW Configured value of ulMinCycleLength in the configuration packet is smaller than the current hardware/software combination specific minimums.
0xC0E6000D	ERR_EPLCN_NMT_INVALID_NODE_ID Configured NodeId is invalid.

Table 57: Status codes / Error codes

6 Appendix

6.1 LED Status

For the POWERLINK Controlled Node protocol, the communication LEDs **BS** (Bus Status) and **BE** (Bus Error) as well as the Ethernet LED **L/A** can assume the status described below. This description is valid from Ethernet POWERLINK protocol (reference 2, chapter 10 “Indicators”).

LED	Color	State	Meaning
BS (Bus Status) General name: **COM 0**	**Duo LED red/green**		
	(green)	On	Slave is in **‘Operational’** state
	(green)	Triple Flash	Slave is in **‘ReadyToOperate’** state
	(green)	Double flash	Slave is in **‘Pre-Operational 2’** state
	(green)	Single flash	Slave is in **‘Pre-Operational 1’** state
	(rgreen)	Flickering (10 Hz)	Slave is in **‘Basic Ethernet’** state
	(green)	Blinking (2,5 Hz)	Slave is in **‘Stopped’** state
	(off)	Off	Slave initializing
BE (Bus Error) General name: **COM 1**	**Duo LED red/green**		
	(off)	Off	Slave has no error
	(red)	On	Slave has detected an error
L/A Ch0 & Ch1	**LED green**		
	(green)	On	**Link:** The device is linked to the Ethernet, but does not send/receive Ethernet frames.
	(green)	Flickering (load dependent)	**Activity:** The device is linked to the Ethernet and sends/receives Ethernet frames.
	(off)	Off	The device has no link to the Ethernet.
Ch0 & Ch1	**LED yellow**		
	(off)	Off	This LED is not used.

Table 58: LED states for the POWERLINK Controlled Node

(Continued on next page.)

LED state	Definition
On	The indicator is constantly on.
Off	The indicator is constantly off.
Triple Flash	The indicator shows a sequence of three short flashes (each 200 ms), separated by a short off phase (200 ms). The sequence is finished by a long off phase (1,000 ms).
Double flash	The indicator shows a sequence of two short flashes (each 200 ms), separated by a short off phase (200 ms). The sequence is finished by a long off phase (1,000 ms).
Single flash	The indicator shows one short flash (200 ms) followed by a long "off" phase (1,000 ms).
Flickering (10 Hz)	The indicator turns on and off with a frequency of approximately 10 Hz: on for approximately 50 ms, followed by off for 50 ms. Red and green LEDs shall be on alternately.
Blinking (2,5 Hz)	The indicator turns on and off with a frequency of approximately 2.5 Hz: on for approximately 200 ms, followed by off for 200 ms. Red and green LEDs shall be on alternately.
Flickering (load dependent)	The indicator turns on and off with a frequency of approximately 10 Hz to indicate high Ethernet activity: on for approximately 50 ms, followed by off for 50 ms. The indicator turns on and off in irregular intervals to indicate low Ethernet activity.

Table 59: LED state definitions for the POWERLINK Controlled Node protocol

Note: If the node uses a combined LED (S/E), the red and green subfunctions are equivalent to the BS and BE LEDs described this chapter. However the BS part is dominant over the BE part, e.g. if BS flash is required, the BE will be turned off during the flash.

6.2 Device description file (XDD)

This chapter provides some information and hints about the device description files for Ethernet POWERLINK Controlled Nodes. These files are XML format based and called XDD (XML Device Description) files. Refer to 4 Ethernet Powerlink XML Device Description; EPSG DS 311 V1.0.0; 2007 for more details about the XDD files.

The following sections contain steps and examples how to customize the XDD files provided with the Hilscher EPL Controlled Node firmware.

Note: The information and configuration defined in the XDD file has to be compatible with the stack configuration done by the application.

6.2.1 Device identification information

The first to customize the XDD file is changing the vendor information. This information has to be edited in the following parts

Identity of the EPL Device Profile:

In the EPL Device Profile (ISO15745ProfileContainer->ISO15745Profile-> ProfileHeader-> ProfileIdentification = EPL_Device_Profile) edit the element …->ProfileBody->DeviceIdentity:

```
<DeviceIdentity>
   <vendorName>Hilscher Gesellschaft fuer Systemautomation mbH</vendorName>
   <vendorID>0x44</vendorID>
   <deviceFamily>
      <label lang="en" />
      <description lang="en" />
   </deviceFamily>
   <productName>NETX 51 RE/PLS</productName>
   <productID>0x1E</productID>
   <orderNumber />
</DeviceIdentity>
```

The vendorName and vendorID values are managed by the EPSG (see http://www.ethernet-powerlink.org). The elements productName, productID and orderName are defined by the vendor.

Identity of Powerlink Communication Profile

In the Powerlink Communication Profile (ISO15745ProfileContainer->ISO15745Profile-> ProfileHeader-> ProfileIdentification = Powerlink_Communication_Profile) edit the element …->ProfileBody-> ApplicationLayers-> identity:

```
<identity>
   <vendorID>0x44</vendorID>
   <productID>0x1E</productID>
</identity>
```

The vendorID value is managed by the EPSG (see http://www.ethernet-powerlink.org). The productID is defined by the vendor.

Identity Object in the OD

In the Powerlink Communication Profile (ISO15745ProfileContainer->ISO15745Profile->ProfileHeader-> ProfileIdentification = Powerlink_Communication_Profile) edit the object …->ProfileBody-> ApplicationLayers-> ObjectList-> Object->index = 1018:

```
<Object index="1018" name="NMT_IdentityObject_REC" objectType="9">
   <SubObject accessType="const" dataType="0005" defaultValue="0x4"
   name="NumberOfEntries" objectType="7" subIndex="00" />
   <SubObject accessType="const" dataType="0007" defaultValue="0x44" name="VendorId_U32"
   objectType="7" subIndex="01" />
   <SubObject accessType="const" dataType="0007" defaultValue="0x1E"
   name="ProductCode_U32" objectType="7" subIndex="02" />
   <SubObject accessType="const" dataType="0007" defaultValue="0x0" name="RevisionNo_U32"
   objectType="7" subIndex="03" />
   <SubObject accessType="const" dataType="0007" defaultValue="0x0" name="SerialNo_U32"
   objectType="7" subIndex="04" />
</Object>
```

The value of VendorId_U32 is managed by the EPSG (see http://www.ethernet-powerlink.org). The values ProductCode_U32, RevisionNo_U32 and SerialNo_U32 are defined by the vendor.

6.2.2 Define objects

Since the XDD file has to describe all objects implemented by the node, each new object created by the application has to be also described in the device description file.

For this, just add the new object to the object list (ISO15745ProfileContainer->ISO15745Profile->ProfileHeader->ProfileBody-> ApplicationLayers-> ObjectList in the Powerlink Communication Profile part) with the following structure

Object from type VAR

```
<Object accessType="" dataType="" defaultValue="" index="" name="" objectType="7" />
```

Object from type RECORD, ARRAY or DOMAIN

```
<Object index="3000" name="Example_REC" objectType="9">
   <SubObject accessType="const" dataType="0005" defaultValue="0x03"
name="NumberOfEntries" objectType="7" subIndex="00" />
   <SubObject accessType="const" dataType="0005" defaultValue="0x10" name="SubIdx_1"
   objectType="7" subIndex="01" />
   <SubObject accessType="ro" dataType="0006" defaultValue="0x0020" name="SubIdx_2"
   objectType="7" subIndex="02" />
   <SubObject accessType="rw" dataType="0007" defaultValue="0x00000030" name="SubIdx_3"
   objectType="7" subIndex="03" />
</Object>
```

6.2.3 Object Dictionary default values

The XDD file should contain a description of all objects implemented by the device within their default value. The object list can be found under ISO15745ProfileContainer->ISO15745Profile->ProfileHeader->ProfileBody-> ApplicationLayers-> ObjectList in the Powerlink Communication Profile part.

To change the default value of an object just search for the corresponding index and subindex and edit the parameter “defaultValue”.

Following objects have to be changed depending on the configuration send by the application:

- DeviceType, Index 0x1000. Change this if implementing some standard or vendor specific profiles.
- CycleLen, Index 0x1006. Same value as described in the configuration.
- Identity, Index 0x1018. Changes are described in the chapter above.
- Threshold for CNCollision errors, Index 0x1C0A, Subindex 3. This value has to be changed if the application configures its own value to the stack.
- Threshold for LossSoC errors, Index 0x1C0B, Subindex 3. This value has to be changed if the application configures its own value to the stack.
- Threshold for LossSoA errors, Index 0x1C0C, Subindex 3. This value has to be changed if the application configures its own value to the stack.
- Threshold for LossPReq errors, Index 0x1C0D, Subindex 3. This value has to be changed if the application configures its own value to the stack.
- Threshold for CNCrc errors, Index 0x1C0F, Subindex 3. This value has to be changed if the application configures its own value to the stack.
- IP Settings, Index 0x1E40, Subindex 2. This value has to be changed to 192.168.100.NodeId
- Default Gateway, Index 0x1E40, Subindex 5. Same value as described in the configuration.
- FeatureFlags, Index 0x1F82. Same value as described in the configuration.
- NodeId, Index 0x1F93, Subindex 1. Same value as described in the configuration.

6.2.4 PDO configuration

This section provides the steps to be followed to changed successfully the PDO configuration in the XDD file and make it compatible with the configuration defined by the application.

Note: For devices with static PDO mapping and default objects created by the stack, the PDO layout and mapping has to follow the rules described in section *PDO objects configuration* on page 56.

Create user data objects

First of all, create the PDO objects (range 0x2000 – 0x5FFF) with the desired PDO layout. Refer to section *Define objects* on page 105 for more information about defining new objects in the XDD file.

Create the PDO mapping configuration

Adjust the mapping configuration to the new PDO layout by editing the objects 0x1600 and 0x1A00 beginning with index 1. The coding of the mapping entries is described in details in section *Configuring a receive PDO* (page 34) and *Configuring transmit PDO* (page 35).

Define the PDO mapping version (Optional)

Change the mapping version in the objects 0x1400 and 0x1800 subindex 2.

Change PDO size

Adjust the PDO size and max size in the object 0x1F98, Subindices 1, 2, 4 and 5.

6.3 List of figures

6.4 List of tables

6.5 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.
69500 Bron
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr
Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
Phone: +91 8888 750 777
E-Mail: info@hilscher.in

Italy
Hilscher Italia S.r.l.
20090 Vimodrone (MI)
Phone: +39 02 25007068
E-Mail: info@hilscher.it
Support
Phone: +39 02 25007068
E-Mail: it.support@hilscher.com

Japan
Hilscher Japan KK
Tokyo, 160-0022
Phone: +81 (0) 3-5362-0521
E-Mail: info@hilscher.jp
Support
Phone: +81 (0) 3-5362-0521
E-Mail: jp.support@hilscher.com

Korea
Hilscher Korea Inc.
Seongnam, Gyeonggi, 463-400
Phone: +82 (0) 31-789-3715
E-Mail: info@hilscher.kr

Switzerland
Hilscher Swiss GmbH
4500 Solothurn
Phone: +41 (0) 32 623 6633
E-Mail: info@hilscher.ch
Support
Phone: +49 (0) 6190 9907-99
E-Mail: ch.support@hilscher.com

USA
Hilscher North America, Inc.
Lisle, IL 60532
Phone: +1 630-505-5301
E-Mail: info@hilscher.us
Support
Phone: +1 630-505-5301
E-Mail: us.support@hilscher.com