Protocol API

EtherNet/IP Adapter

DOC150401APIV5.4.0.4EN | Revision V5.4.0.4 | English | Released | Public | 2024-04-02

empowering communication

Table of Contents

1 Introduction 5
1.1 About this document 5
1.2 System requirements 6
1.3 Target group 6
1.4 Specifications 6
1.5 Terms, abbreviations and definitions 7
1.6 Input and output data conventions 8
1.7 Scope and limits of this document 9
1.7.1 Intended audience 9
1.7.2 Examples of implementation-defined behavior 9
1.8 References to documents 10
2 Hilscher EtherNet/IP stack capabilities 11
2.1 Loadable Firmware (LFW) 11
2.2 Available object classes 12
2.2.1 Introduction 12
2.2.2 Class attributes 13
2.2.3 Instance attributes 14
2.2.4 Services 14
2.2.5 Identity Object (class code: 0x01) 15
2.2.6 Message Router Object (class code: 0x02) 17
2.2.7 Assembly Object (class code: 0x04) 18
2.2.8 Connection Manager Object (class code: 0x06) 19
2.2.9 Time Sync Object (class code: 0x43) 20
2.2.10 Device Level Ring Object (class code: 0x47) 23
2.2.11 Quality of Service Object (class code: 0x48) 24
2.2.12 TCP/IP Interface Object (class code: 0xF5) 26
2.2.13 Ethernet Link Object (class code: 0xF6) 29
2.2.14 LLDP Management Object (class code: 0x109) 31
2.2.15 Predefined Connection Object (class code: 0x401) 32
2.2.16 Diagnosis Object (class code: 0x403) 36
2.2.17 IO Mapping Object (class code: 0x402) 37
2.3 Ethernet MAC address 38
2.4 Device data 39
2.4.1 Device serial number 39
2.5 Status information 41
2.5.1 DPM communication status 41
2.5.2 DPM COS flags 42
2.5.3 Other DPM status bits 42
2.6 Module and network status 43
2.6.1 Module status 43
2.6.2 Network status 44
2.7 Handshake modes 45
2.7.1 Input handshake mode / output handshake mode 45
2.7.2 Synchronization handshake mode 46
2.7.3 Configuration 46
2.8 Quality of Service 47
2.8.1 Introduction 47
2.8.2 DiffServ 47
2.8.3 802.1D/Q Protocol 48
2.8.4 The QoS Object 48
2.9 Device Level Ring 50
2.9.1 Ring supervisors 50
2.9.2 Beacon and announce frames 51
2.9.3 Ring nodes 51
2.9.4 Normal network operation 51
2.9.5 Rapid fault/restore cycles 51
2.10 CIP device protection 52

2.10.1 Introduction 52
2.10.2 Protection modes........ 52
2.10.3 Protection policy........ 52
2.11 Module and Network Status LEDs........ 54
2.12 DHCP/BOOTP Client........ 55
2.12.1 DHCP Behavior 55
2.12.2 DHCP Device Level Behavior 55
2.12.3 Packet API........ 55
2.12.4 DHCP Options........ 56
2.13 QuickConnect 58
3 Getting started / configuration........ 59
3.1 Configuration methods 59
3.2 Host application behavior........ 60
3.2.1 Startup........ 61
3.2.2 Operational........ 61
3.2.3 Configuration 61
3.2.4 Reset........ 61
3.3 Configuration using the packet API........ 62
3.3.1 Basic configuration packet set 62
3.3.2 Extended configuration packet set........ 64
3.4 Configuraion using Sycon.net 66
3.4.1 Configuration sequence 66
3.5 Remanent data........ 67
3.5.1 Remanent data purpose........ 67
3.5.2 Remanent data responsibility........ 67
3.5.3 Remanent data state........ 68
3.5.4 Remanent data flow........ 69
3.5.5 Remanent data content........ 71
3.6 Bus State........ 72
3.6.1 Purpose 72
3.6.2 BusOn and BusOff States 72
3.6.3 BusOn and Producing Assembly Run Status........ 72
4 Application interface........ 73
4.1 Packet usage........ 73
4.1.1 Value range........ 73
4.2 Configuring the EtherNet/IP Adapter 74
4.2.1 Set Configuration Parameters service 75
4.2.2 Set Parameter Flags service 82
4.2.3 Finish configuration of CIP objects 83
4.2.4 Register an additional object class........ 84
4.2.5 Register a new Assembly instance........ 86
4.2.6 Register service........ 93
4.2.7 Set Parameter 94
4.2.8 CIP Service request........ 96
4.2.9 Set Watchdog Time........ 99
4.2.10 Register/Unregister Application........ 99
4.2.11 Start/Stop Communication 99
4.2.12 Channel Init........ 99
4.3 Acyclic events indicated by the stack........ 100
4.3.1 Application compliance 100
4.3.2 Indication of a reset request from the network........ 101
4.3.3 Connection State Change indication........ 103
4.3.4 Configuration Assemblies........ 108
4.3.5 NULL ForwardOpen........ 109
4.3.6 Acyclic Data Transfer indication........ 110
4.3.7 CIP Object Change indication........ 114
4.3.8 Link Status Change........ 118
4.3.9 Module Network Status Change........ 119
4.3.10 Forward_Open indication 120

4.3.11 Forward_Open_Completion indication 124
4.3.12 Forward_Close indication 125
4.3.13 Store Remanent Data indication 128
4.4 Additional services requested by the application 129
4.4.1 Get Module Status/ Network Status 130
4.4.2 Set Watchdog Time 131
4.4.3 Get Watchdog Time 131
4.4.4 Get DPM I/O Information 131
4.4.5 Delete Configuration 131
4.4.6 Lock/Unlock Configuration 131
4.4.7 Get Firmware Identification 131
4.4.8 Get Component Information 131
4.4.9 Set Remanent Data request 132
4.4.10 Set Trigger Type 132
4.4.11 Get Trigger type 134
4.4.12 Force LED State service 135
4.4.13 Enable Attribute service 136
4.4.14 Set Attribute Permission service 137
4.4.15 Enable Attribute Notification service 138
4.4.16 Enable/Disable Attribute Protection service 139
4.4.17 Terminate CIP Connection service 140
5 Resource and feature configuration via tag list 141
6 Status/error codes 142
6.1 Stack-specific error codes 142
6.2 General EtherNet/IP error codes 146
7 Appendix 148
7.1 Legal notes 148
7.2 Third party software license 152
7.3 Contacts 153

Chapter 1 Introduction

1.1 About this document

This manual describes the user interface of the EtherNet/IP Adapter implementation on the netX companion chip. The aim of this manual is to support the integration of netX-based devices with customer applications via the DPM interface.

The general approach of exchanging data between the Host CPU and the netX companion chip is independent of the EtherNet/IP protocol. This general procedure, for the purpose of issuing commands toward the companion chip and receiving events from it, are subject to the protocol-independent netX DPM Interface manual [1].

1.2 System requirements

This software package has following system requirements to its environment:

- netX chip as CPU hardware platform

V5 firmware compatibility between EtherNet/IP Adapter firmware and Maintenance Firmware

This compatibility statement is important for use case C firmware for netX 90, which uses a Flash file system: To ensure proper usage of Flash sectors during file write operations, a Flash Translation Layer (FTL) has been integrated in the operating system of the firmware. The Flash file system layout has changed and is not compatible to earlier versions.

IMPORTANT Starting with firmware V5.2.0.0, the firmware requires a Flash file system in the new format and Maintenance Firmware V1.3.0.0 (or higher).

In case you intent to update the EtherNet/IP Slave firmware from version 5.1 to version 5.2, this requires

1. to update the Maintenance Firmware to V1.3.0.0 (or higher),
2. to update the EtherNet/IP Adapter firmware to V5.2.0.0 (or higher) and finally
3. to reformat the file system using a full format request (HIL_FORMAT_REQ).

The full format request is a system service in the EtherNet/IP Adapter firmware V5.2.0.0 (or higher).

1.3 Target group

This manual is intended for software developers with knowledge of:

- the netX DPM Interface manual
- the Common Industrial Protocol (CIP™) Specification, volume 1
- the Common Industrial Protocol (CIP™) Specification, volume 2

1.4 Specifications

The EtherNet/IP Adapter firmware specifications can be found in the corresponding firmware datasheet, see reference [11].

1.5 Terms, abbreviations and definitions

Term	Description
ACD	Address Conflict Detection
AP	Application on top of the Stack
API	Actual Packet Interval or Application Programmer Interface
ARP	Address Resolution Protocol
AS	ASsembly object
BLOB	Binary Large OBject
BOOTP	Boot Protocol
CIP	Common Industrial Protocol
CM	Connection Manager
DDP	Device Data Provider
DHCP	Dynamic Host Configuration Protocol
DiffServ	Differentiated Services
DLR	Device Level Ring (i.e. ring topology on device level)
DPM	Dual Port Memory
DSCP	Differentiated Services Code Point
EIM	Ethernet/IP Scanner (= Master)
EIP	Ethernet/IP
EIS	Ethernet/IP Adapter (= Slave)
ENCAP	Encapsulation Layer
ERC	Extended Error Code
FDL	Flash Device Label
GRC	Generic Error Code
IANA	Internet Assigned Numbers Authority
ID	Identity Object
IP	Internet Protocol
LSB	Least Significant Byte
MR	Message Router Object
MS	Module Status
MSB	Most Significant Byte
NS	Network Status
O2T	Direction of data flow in CIP I/O connections: Originator to Target
ODVA	Open DeviceNet Vendors Association
OEM	Original Equipment Manufacturer
OSI	Open Systems Interconnection (according to ISO 7498)
PHB	Per-hop behavior
PLC	Programmable Logic Controller
QoS	Quality of Service
RPI	Requested Packet Interval
T2O	Direction of data flow in CIP I/O connections: Target to Originator
TCP/IP	Transmission Control Protocol / Internet Protocol
TOS byte	Type of Service byte
UCMM	Unconnected Message Manager
UDP/IP	User Datagram Protocol / Internet Protocol
VLAN	Virtual Local Area Network

Table 1. Terms, abbreviations and definitions

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

1.6 Input and output data conventions

In certain cases, EtherNet/IP and netX use different naming schemes. To avoid problems, this section clarifies the naming conventions:

EtherNet/IP	netX	EtherNet/IP stack	Description
Producing/Input (assembly data)	Application writes data into the output area of the process data memory	Producing/Input assembly	Data sent to EtherNet/IP Scanner (e.g. PLC).
Consuming/Output (assembly data)	Application reads data from the input area of the process data memory	Consuming/Output assembly	Data received from EtherNet/IP Scanner (e.g. PLC).

1.7 Scope and limits of this document

1.7.1 Intended audience

This is an application interface manual primarily addressing application and system developers. The manual may be of limited use for other audiences, e.g. software testers, since it deliberately leaves many aspects open. In particular, this manual does not elaborate on error cases much, but mostly describes the good cases on the functional level.

Thus, it has to be emphasized that this is not a collection of requirements and that it is a poor basis for deriving requirements that are complete, consistent and verifiable.

This document primarily instructs how an application is supposed to behave and not to specify every behavioral aspect of the protocol stack. The focus is on the description of the abstractions, plus providing a reference manual. Therefore, while it can be a valuable source also for software testers, requirements engineers, etc., it has to be considered inherently incomplete from their points of view.

1.7.2 Examples of implementation-defined behavior

In general, the behavior of the packet services is in fact undefined in all but the few error cases that are explicitly addressed in this document. This is in line with the design goal to provide a robust software platform, which is convenient to use. General error cases exist: a packet that is too short will always be rejected, and excess or inconsistent data will be rejected or truncated. These details are not formally covered at this level of abstraction.

Therefore, we introduce the term *implementation-defined behavior* to describe the undefined behavior within the scope of this document. It is important to note that our focus lies primarily on the normal operation of the packet API, with limited emphasis on error cases. However, we do provide some examples of *implementation-defined behavior* below:

- Testing of parameter ranges

For the request packets described in this document, parameter value ranges are specified in each particular table. Correct applications shall select from these value ranges. Most of the times, in case a parameter value outside of the specified range is given, or any other misbehavior of the application is detected, the EtherNet/IP Adapter will explicitly reject the packet with an error status. However, this is not a general requirement.

For instance, there are parameters that logically resemble a boolean value encoded as a single byte, and the `Value/Range` column reads {0, 1}. The EtherNet/IP Adapter implementation is likely to interpret all values equal or larger than 1 as TRUE, so that the value is naturally clamped to a maximum of 1. We consider this *implementation-defined behavior*, which is not further described in this document. The value range is specified towards the application developer to choose from.

Another example would be a bitmask parameter, where only a few bits are defined. It is *implementation-defined behavior* whether such packets will be rejected in case undefined bits are set or whether the undefined bits are just ignored, unless it is explicitly stated for a particular packet.

Also, values of pad bytes typically are ignored where the application is instructed to set them to zero in the value tables, so that the `Value/Range` field would simply be a recommendation.

- Length of response packets

A different example of *implementation-defined behavior* is the length of response packets. Tables that describe response packets, mostly provide a fixed size constant value for the response length (tHead.ulLen). This depicts the successful case. The behavior is undefined in case of errors. The implementation may just return a packet with a zero tHead.ulLen for some error cases and for others it may return the whole packet data. The host application will assess the status code and accesses the packet's data part not before verifying also the packet length to be sufficiently large.

> **NOTE** *Implementation-defined behavior* hardly can be subject of more formal work, such as structured software tests. While it is acknowledged that there is a lack of detail specifically to the error cases, it is not the intent of this document to bloat the provided information for better accessibility and lower maintenance.

1.8 References to documents

This document refers to the following documents:

[1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Manual, netX Dual-Port Memory Interface, Revision 17, English, 2020.

[2] Hilscher Gesellschaft für Systemautomation mbH: Protocol API, Socket Interface, Packet Interface, Revision 7, English, 2021.

[3] Hilscher Gesellschaft für Systemautomation mbH: Protocol API, Ethernet Interface, Packet Interface, Revision 12, English, 2022.

[4] Hilscher Gesellschaft für Systemautomation mbH: Application Note: CIP Sync, Revision 5, English, 2015.

[5] ODVA: The CIP Networks Library, Volume 1, “Common Industrial Protocol (CIP™)”, Edition 3.33, November 2022.

[6] ODVA: The CIP Networks Library, Volume 2, “EtherNet/IP Adaptation of CIP”, Edition 1.31, November 2022.

[7] The Common Industrial Protocol (CIP™) and the Family of CIP Networks, Publication Number: PUB00123R1, downloadable from ODVA website (http://www.odva.org/).

[8] Hilscher Gesellschaft für Systemautomation mbH: Tag List Editor - Operating Instruction Manual - Revision V1.5, English, 2020.

[9] Hilscher Gesellschaft für Systemautomation mbH: Packet API, netX Dual-Port Memory, Packet-based services, For

Firmware version EtherNet/IP Adapter V3 (netX 50/51/52/100/500-based firmware) : Revision 5, English, 2021
For Firmware version EtherNet/IP Adapter V5 (netX 90/4000/4100-based firmware) : Revision 7, English, 2022

[11] Hilscher Gesellschaft für Systemautomation mbH: Firmware datasheet, located next to the corresponding firmware file

[12] ODVA - "Brand Standards + Identity Guidelines", Publication Number: PUB00036R10, downloadable from ODVA website (http://www.odva.org/).

Chapter 2 Hilscher EtherNet/IP stack capabilities

This section describes the Hilscher EtherNet/IP stack interfaces, introduces the implemented CIP objects, and specifies the provided CIP services and their availability via the different interfaces.

2.1 Loadable Firmware (LFW)

When running the LFW, the netX chip serves as a dedicated communication processor while the host application uses its own processor.

The host application exchanges process data using the mechanism described in the DPM Interface manual [1]. In addition, the host application uses the firmware packet interface (which is subject to this manual) for configuration and event handling.

Figure Interfaces of the EtherNet/IP stack (LFW) shows that the firmware provides two interfaces:

1. The **EIP-API** (DPM/packet interface) toward the host application for exchange of commands, event notification and process data via the Hilscher DPM interface.
2. The **EtherNet/IP Network Interface** acc. to the CIP specification. Based on TCP/IP and UDP/IP, external devices communicate with the protocol stack via one of the 3 supported connection types:
   - Unconnected Explicit Messaging (UCMM)
   - Connected Explicit Messaging (class 3)
   - Implicit Messaging (class 0/1)

NOTE Depending on the type of firmware, there might be other provided interfaces (e.g. Ethernet API, Socket-API). Those interfaces are not in the scope of this document. For the available interfaces of your firmware, please refer to the corresponding firmware datasheet [11].

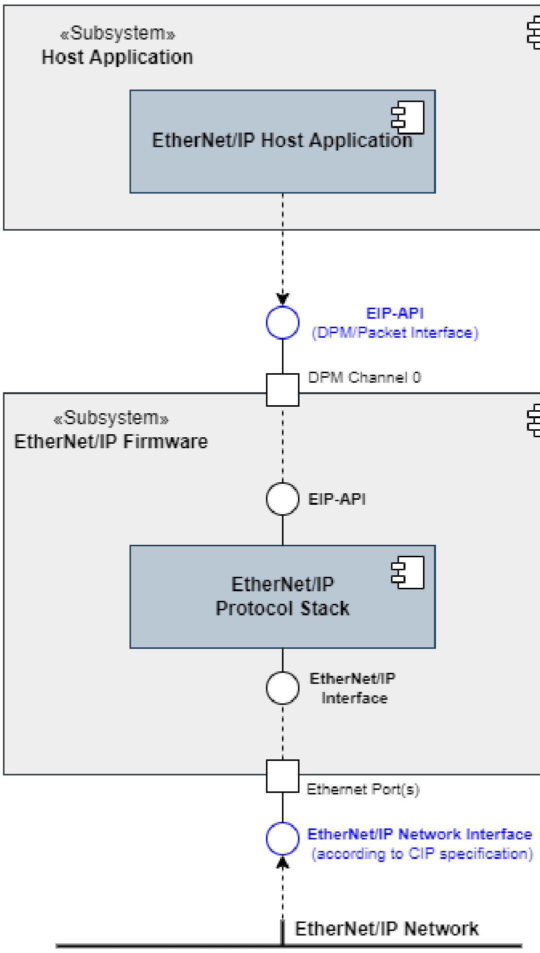


Figure 1. Interfaces of the EtherNet/IP stack (LFW)

2.2 Available object classes

The following subsections describe all default CIP object classes available within the Hilscher EtherNet/IP stack. We synonymously refer to this set of objects as *built-in* objects or *default* objects. Figure Default Hilscher Device object model gives an overview of the available CIP objects and their instances in the default configuration of the protocol stack.

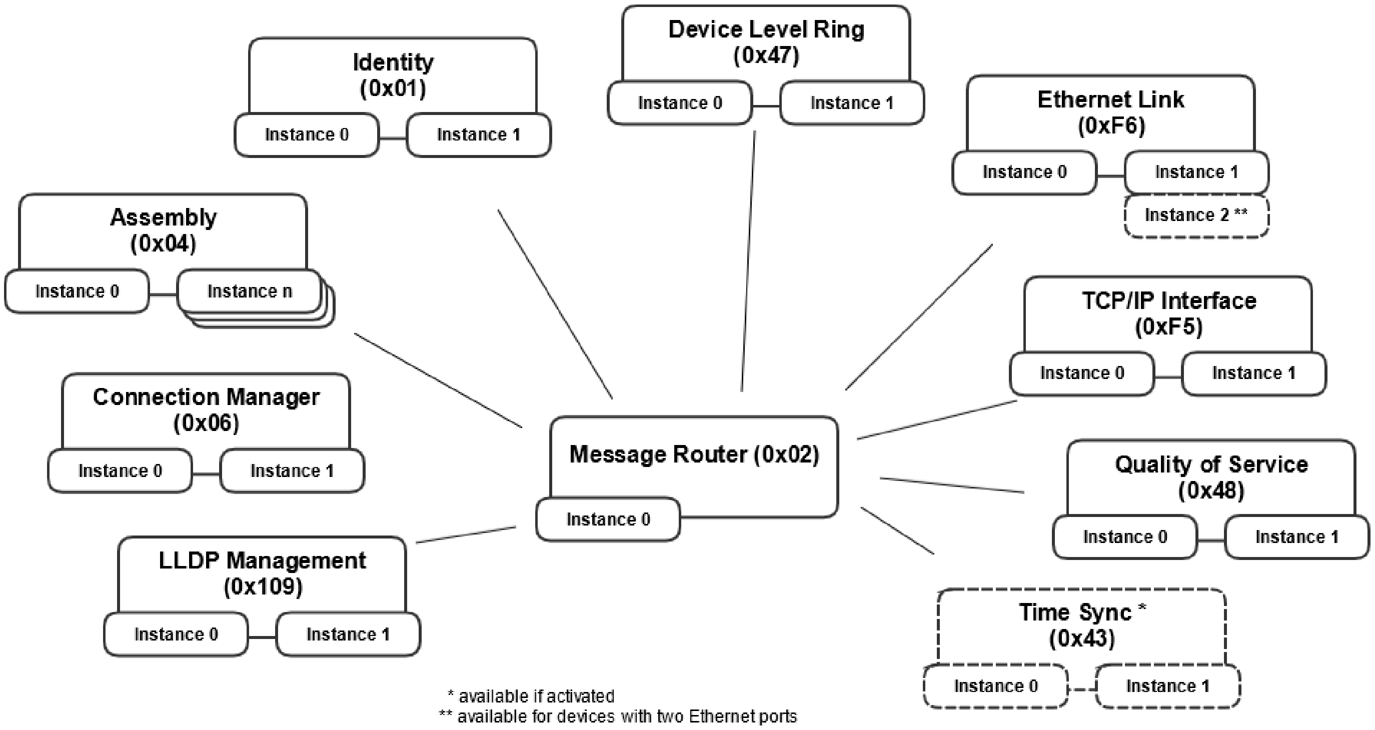


Figure 2. Default Hilscher Device object model

2.2.1 Introduction

Speaking of CIP object classes means to distinguish between class and instance level. Each object exists at class level and, additionally, may have one or more instances. CIP services address a certain object class or instance by means of a specified Instance ID. An Instance ID value of zero addresses the object class, whereas Instance IDs larger than zero address the corresponding instance of that object class.

Each CIP object class and instance consists of a set of attributes and services. Of course, the attributes each object class provides at class and instance levels differ from each other. The most common services are the Get_Attribute_Single and Set_Attribute_Single services to read or write the attributes of the addressed object class or instance.

The following sections use four tables to describe each supported object class:

1. Class attributes
2. Instance attributes
3. Services available to the host application
4. Services available to EtherNet/IP clients over the network

2.2.2 Class attributes

Class attributes are defined using the following notation:

Class attributes (instance 0)

Attr ID	Name	Access		Description	Default value	Supported by default
		from network	from host			
1	2	3	4	5	6	7

Table 2. Introduction of class attribute description

1. The **Attribute ID** is an integer identification value assigned to an attribute. Use the Attribute ID in the Get_Attributes and Set_Attributes services list. The Attribute ID identifies the particular attribute being accessed.
2. **Name** specifies the name of the class attribute.
3. **Access from network** specifies the access permission of the attribute when the service is sent from the EtherNet/IP network (protocol stack EtherNet/IP interface – see Loadable Firmware (LFW)). The definitions are:
   - Set (Settable) - The attribute is accessible by at least one of the set services (Set_Attribute_Single/ Set_Attribute_All).
   - Get (Gettable) - The attribute is accessible by at least one of the get services (Get_Attribute_Single/ Get_Attribute_All).
4. **Access from host** specifies the access permission of the attribute when the service is sent from the host application using the DPM/Packet Interface see Loadable Firmware (LFW) of the stack (see description of packet EIP_OBJECT_CIP_SERVICE_REQ).
   Definitions for access rules:
   - Set (Settable) - The attribute is accessible by at least one of the set services (Set_Attribute_Single/ Set_Attribute_All).
   - Get (Gettable) - The attribute is accessible by at least one of the get services (Get_Attribute_Single/ Get_Attribute_All).
5. **Description** contains a descriptive text on the attribute.
6. **Default value** specifies the default value of the attribute.
7. **Supported by default** indicates whether the stack supports this attribute in a default configuration.
   In a default configuration, the EtherNet/IP stack implements certain attributes, which are not accessible from the EtherNet/IP network. In order to access these attributes via the network, the host application has to activate them using a specific service EIP_OBJECT_ENABLE_ATTRIBUTE_REQ.
   ✔ → The attribute is supported and activated by default.
   ⚠ → The attribute is supported and deactivated by default. The host can activate it.
   ✖ → The attribute is not supported. The host cannot activate it.

2.2.3 Instance attributes

An instance attribute is an attribute that is specific to an object class instance. Instance attributes are defined in the same notation as class attributes.

Instance attributes (Instance [1..N])

Attr ID	Name	Access		Description	Default value	Supported by default
		from network	from host			
1	2	3	4	5	6	7

Table 3. Introduction of instance attribute description

2.2.4 Services

Services can address the class level (Instance ID 0) or the instance level (Instance ID [1..N]) of a CIP object. Services may be issued by the host application or by a client on the EtherNet/IP network.

For each object, services will be presented in a table of the following format:

Service code	Name	Addressing the object		Description
		class level	instance level	
1	2	3	4	5

Table 4. Introduction of service description

1. **Service code** is a unique identifier for the CIP service. The range of integer values [0..255] defines service codes according to the EtherNet/IP specification.
2. **Name** specifies the name of the service.
3. Addressing the class level of the object

   ✅ → The stack supports this service at object class level (Instance ID 0).
   ❎ → The stack does not support this service at class level.
4. Addressing the instance level of the object

   ✅ → The stack supports this service at object instance level (instance 1-n).
   ❎ → The stack does not support this service at instance level.
5. **Description** contains descriptive text on the service.

2.2.5 Identity Object (class code: 0x01)

The Identity object provides identification and general information about the device. The EtherNet/IP protocol stack implements the Identity object at class level and a single instance with Instance ID 1.

2.2.5.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(2)	✓
2	Max. Instance	Get	Get	Max. instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Max. ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Max. ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(19)	✓

Table 5. Identity Object - class attributes

2.2.5.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Vendor ID	Get	Get/Set	Vendor Identification	(0x011B) Hilscher	✓
2	Device Type	Get	Get/Set	Indication of general type of product	(1)	✓
3	Product Code	Get	Get/Set	Identification of a particular product of an individual vendor	(12345)	✓
4	Revision	Get	Get/Set	Revision of the product	(1.1)	✓
5	Status	Get	Get	Summary status of device		✓
6	Serial Number	Get	Get	Serial number of device	See section Device serial number	✓
7	Product Name	Get	Get/Set	Human readable identification See [12] for information about restrictions regarding product naming.	“netX”	✓
8	State	Get	Get	Present state of the device		✓
9	Conf. Consist. Value	Get	Get/Set	Configuration Consistency Value	0	⚠
10	Heartbeat Interval	Get	Get/Set	The nominal interval between heartbeat messages in seconds	0	⚠
19	Protection Mode	Get	Get/Set	Current protection mode of the device (see section CIP device protection for more information)	0	✓

Table 6. Identity Object - instance attributes

2.2.5.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x01	Get Attribute All	✓	✓	Retrieve all attribute values
0x05	Reset[1]	✓	✓	Reset the device
0x4B	Flash LEDs	✗	✓	Flash the device's LEDs for identification
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✓	✓	Modify attribute value

[1] In case the Safety Network Number is activated (see section Instance attributes), the reset service will not be supported for any instance. In that case the service will be rejected with general status code 0x08 "Service not supported".

Table 7. Identity Object - common services

2.2.6 Message Router Object (class code: 0x02)

The Message Router Object is responsible for dispatching service requests toward the addressed object class or object class instance. The EtherNet/IP protocol stack implements the Message Router object exclusively at class level.

2.2.6.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(0)	✓

Table 8. Message Router Object - Class attributes

2.2.6.2 Instance attributes

The EtherNet/IP protocol stack implements the Message Router object exclusively at class level. It does not provide any instances.

2.2.6.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	☒	✓	Modify attribute value

Table 9. Message Router Object - Common services

2.2.7 Assembly Object (class code: 0x04)

The Assembly object stores process data for exchange with other EtherNet/IP devices over the network and with the host application.

2.2.7.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(2)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(0)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(0)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(4) 1)	✓

1) The default value of the maximum instance attribute ID (attribute 7) is determined by the activated instance attributes. In certain cases, this value may be zero, e.g. if no assemblies are present.

Table 10. Assembly Object - Class attributes

2.2.7.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Number of Member	Get	Get	Number of members in List	n.a.	✓ / ✗ 1)
2	Member	Get	Get	Member list	n.a.	✓ / ✗ 1)
3	Data	Get/Set	Get/Set	Current process data snapshot	n.a.	✓
4	Size	Get	Get	Process data size in number of bytes	n.a.	✓
769	Parameter	None	Get	Assembly parameter	n.a.	⚠
770	Status	None	Get	Status of the assembly	n.a.	⚠

1) Attributes 1 and 2 are not available for configuration assembly instances.

Configuration assembly instances are added by using the flag `EIP_AS_TYPE_CONFIG`.
For more information, see section EIP_OBJECT_AS_REGISTER_REQ.

Table 11. Assembly Object - Instance attributes

2.2.7.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✗	✓	Modify attribute value
0x18	Get Member	✗	✓	Get a member of instance attribute 2

Table 12. Assembly Object - Common services

2.2.8 Connection Manager Object (class code: 0x06)

The Connection Manager Class manages class 0/1 implicit I/O and class 3 explicit connections.

2.2.8.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(0)	✓

Table 13. Connection Manager Object - Class attributes

2.2.8.2 Instance attributes

The EtherNet/IP protocol stack does not provide any instance attributes for the connection manager object.

2.2.8.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✗	✓	Modify attribute value
0x54	Forward Open (1)	✗	✓	Open new connection
0x4E	Forward Close (1)	✗	✓	Close connection

(1) This service is only available to remote EtherNet/IP clients. Initiated from the host application, the service will be rejected with an appropriate error code.

Table 14. Connection Manager Object - Common services

2.2.9 Time Sync Object (class code: 0x43)

The Time Sync Object (used for CIP SYNC) provides a CIP interface to the IEEE 1588 (IEC 61588) Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, commonly referred to as the Precision Time Protocol (PTP). When starting the stack, this object is not available right away. The host application has to activate the TimeSync object using the packet EIP_OBJECT_MR_REGISTER_REQ.

NOTE Please check the data sheet of the firmware used to see whether it supports CIP Sync (see reference [11]).

NOTE The TimeSync object has to be registered during the stack configuration sequence, before the EIP_APS_CONFIG_DONE_REQ packet. Registration during runtime leads to undefined behavior.

For details on CIP Sync and its use with the EtherNet/IP protocol stack and your host application, refer to the corresponding Application Note [4].

2.2.9.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(3)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(768)	✓

Table 15. Time Sync Object - Class attributes

2.2.9.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	PTPEnable	Get/Set	Get/Set	PTP Enable	0 (Disabled)	✓
2	IsSynchronized	Get	Get	Local clock is synchronized with master	0	✓
3	SystemTimeMicroseconds	Get	Get	Current value of system_time in microseconds	unsynchronized clock counts from zero	✓
4	SystemTimeNanoseconds	Get	Get	Current value of system_time in nanoseconds	unsynchronized clock counts from zero	✓
5	OffsetFromMaster	Get	Get	Offset between local clock and master clock	0	✓
6	MaxOffsetFromMaster	Get/Set	Get/Set	Maximum offset between local clock and master clock since last reset of this value.	0	✓
7	MeanPathDelayToMaster	Get	Get	Mean path delay to master	0	✓
8	GrandMasterClockInfo	Get	Get	Grandmaster Clock Info	n.a.	✓
9	ParentClockInfo	Get	Get	Parent Clock Info	all 0	✓
10	LocalClockIno	Get	Get	Local Clock Info	n.a.	✓

11	NumberOfPorts	Get	Get	Number of ports	1	✓
12	PortStateInfo	Get	Get	Port state info	disabled	✓
13	PortEnableCfg	Get/Set	Get/Set	Port enable cfg	enabled	✓
14	PortLogAnnounceIntervalCfg	Get/Set	Get/Set	Port log announce interval cfg	n.a.	✓
15	PortLogSyncIntervalCfg	Get/Set	Get/Set	Port log sync interval cfg	0	✓
18	DomainNumber	Get/Set	Get/Set	Domain number	0	✓
19	ClockType	Get	Get	Clock type	n.a.	✓
20	ManufactureIdentity	Get	Get	Manufacture identity	all 0	✓
21	ProductDescription	Get	Get	Product description	n.a.	✓
22	RevisionData	Get	Get	Revision data	n.a.	✓
23	UserDescription	Get	Get	User description	n.a.	✓
24	PortProfileIdentityInfo	Get	Get	Port profile identity info	00-21-6C-00-01-00	✓
25	PortPhysicalAddressInfo	Get	Get	Port physical address info	Filled in automatically according to device's MAC address	✓
26	PortProtocolAddressInfo	Get	Get	Port protocol address info	Filled in automatically according to device's IP address	✓
27	StepsRemoved	Get	Get	Steps removed	0	✓
28	SystemTimeAndOffset	Get	Get	System time and offset	n.a.	✓
768	SyncParameters	Get/Set[1]	Get/Set[1]	Synchronization Parameters	See below	✓

[1] The time sync parameter attribute (attribute 768) is not available through the GetAttributesList and SetAttributesList services

Table 16. Time Sync Object - Instance attributes

2.2.9.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x03	Get Attributes List	✗	✓	The Get_Attribute_List service returns the contents of the selected attributes of the specified object class or instance
0x04	Set Attributes List	✗	✓	The Set_Attribute_List service sets the contents of selected attributes of the specified object class or instance
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✗	✓	Modify attribute value

Table 17. Time Sync Object - Common services

2.2.9.4 Instance attributes

Attribute 6 - MaxOffsetFromMaster

Attribute 6 of the Time Sync object specifies the maximum deviation between the local clock and the master clock. The attribute can only be set to zero in order to reset the value. Any other values will be rejected.

Attribute 768 (0x300) - Sync parameters

Attribute 768 of the Time Sync object controls synchronization-related parameters. These are used to adjust intervals and offsets of the hardware synchronization signals Sync 0 and Sync 1.

The Sync 0 signal is the interrupt the host application will receive to retrieve the current system time. On each event, the EtherNet/IP stack writes the current system time into the extended data area of the DPM interface. For details, see CIP Sync Application Note [4]).

NOTE | Currently, only Sync 0 can be used.

The following table describes "Time Sync Object- Attribute 768 (0x300)".

Variable	Type	Value/Range	Description
`ulSync0Interval`	UINT32	0, 5000 ... 500000000 Default: 500000000	Sync0 Interval in nanoseconds This parameter specifies the interval of the Sync 0 signal in nanoseconds. The value 0 means the signal is deactivated. The starting point of the Sync0 signal is dependent on the Sync0 Offset (see parameter ulSync0Offset). Please be aware that the interval must be a multiple or fraction of `ulSync1Interval`.
`ulSync0Offset`	UINT32	smaller than `ulSync0Interval` Default: 0	Sync 0 Offset in nanoseconds This parameter specifies a nanosecond offset for the Sync 0 signal relative to the system time (Time of the Sync Master).
`ulSync1Interval`	UINT32	0, 5000 ... 500000000 Default: 500000000	Sync1 Interval in nanoseconds This parameter specifies the interval of the Sync 1 signal in nanoseconds. The value 0 means the signal is deactivated. The starting point of the Sync1 signal is dependent on the Sync1 Offset (see parameter ulSync1Offset). Please be aware that the interval must be a multiple or fraction of `ulSync0Interval`.
`ulSync1Offset`	UINT32	smaller than `ulSync1Interval` Default: 150	Sync 1 Offset in nanoseconds This parameter specifies a nanosecond offset for the Sync 1 signal relative to the system time (Time of the Sync Master).
`ulPulseLength`	UINT32	1 ... 500 AND smaller than the minimum of the values `ulSync0Interval` and `ulSync1Interval`, when converted to microseconds. Default: 4	Pulse length of the trigger signals in microseconds

Table 18. Time Sync Object – Attribute 768 (0x300)

2.2.10 Device Level Ring Object (class code: 0x47)

The Device Level Ring (DLR) Object provides the configuration of the DLR protocol. DLR is used for Ethernet Ring topology.

2.2.10.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(3)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(12)	✓

Table 19. DLR Object - Class attributes

2.2.10.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Network Topology	Get	Get	Current network topology	0 – Linear	✓
2	Network Status	Get	Get	Current network status	0 – Normal	✓
10	Active Supervisor	Get	Get	Active Supervisor Address	(0)	✓
12	Capability Flags	Get	Get	DLR capability of the device	0x82 (Beacon based Ring Node, Flush Table frame support)	✓

Table 20. DLR Object - Instance attributes

2.2.10.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x01	Get Attribute All	☒	✓	Returns content of instance or class attributes
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value

Table 21. DLR Object - Common services

2.2.11 Quality of Service Object (class code: 0x48)

The Quality of Service (QoS) Object provides the configuration of frame priorities. Ethernet frame priorities are set at the Differentiate Service Code Points (DSCP) or at the 802.1Q Tag.

NOTE The 802.1Q VLAN tagging is not supported and the corresponding attribute 1 of the QoS object remains disabled/unavailable.

2.2.11.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(8)	✓

Table 22. QoS Object - Class attributes

2.2.11.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Tag Enable	Get/Set	Get/Set	Enables or disables sending 802.1Q frames on CIP and IEEE 1588 messages. Since the feature is not supported with the current stack version, this attribute is disabled/unavailable.	(0)	☒
2	DSCP PTP Event	Get/Set	Get/Set	DSCP value for PTP Event frames	(59)	✓
3	DSCP PTP General	Get/Set	Get/Set	DSCP value for PTP general frames	(47)	✓
4	DSCP Urgent	Get/Set	Get/Set	DSCP value for implicit messages with urgent priority	(55)	✓
5	DSCP Scheduled	Get/Set	Get/Set	DSCP value for implicit messages with scheduled priority	(47)	✓
6	DSCP High	Get/Set	Get/Set	DSCP value for implicit messages with high priority	(43)	✓
7	DSCP Low	Get/Set	Get/Set	DSCP value for implicit messages with low priority	(31)	✓
8	DSCP Explicit	Get/Set	Get/Set	DSCP value for explicit messages	(27)	✓

Table 23. QoS Object - Instance attributes

2.2.11.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value

0x10	Set Attribute Single	☒	✓	Modify attribute value

Table 24. Quality of Service Object - Common services

2.2.12 TCP/IP Interface Object (class code: 0xF5)

The TCP/IP Interface Object provides an interface to control a device’s TCP/IPv4 network configuration, most importantly the device’s IP Address, Network Mask, and Gateway Address.

The EtherNet/IP Adapter stack supports exactly one instance of the TCP/IP Interface Object.

2.2.12.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(4)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(14)	✓

Table 25. TCP/IP Interface Object - Class attributes

2.2.12.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Status	Get	Get/Set	Interface status		✓
2	Configuration Capability	Get	Get/Set	Interface capability flags	(0x95)	✓
3	Configuration Control	Get/Set	Get/Set	Interface control flags. This allows to select between static and dynamic IP configuration.	(0)	✓
4	Physical Link Object	Get	Get	Path to physical link object	(0x20 0xF6 0x24 0x01)	✓
5	Interface Configuration	Get/Set	Get/Set	Interface Configuration (IP address, subnet mask, gateway address etc.). This attribute reflects the current IP address of the device. In case of dynamic IP configuration, write/set access to this attribute is prohibited. For static IP configuration, the attribute has transactional semantics: An IP configuration can actively be set, and any IP configuation that is passively set, will be reflected as the current attrbute value. See section DHCP/BOOTP Client for details.	(0)	✓
6	Host Name	Get/Set	Get/Set	The Host Name attribute contains the device’s host name, which can be used for informational purposes.	(“”)	✓
7	Safety Network Number 1)	Get	Get/Set	See CIP Safety Specification, volume 5, section 3	(0xFF 0xFF 0xFF 0xFF 0xFF 0xFF)	⚠
8	TTL Value	Get/Set	Get/Set	TTL value for EtherNet/IP multicast packets	(1)	✓

9	Mcast Config	Get/Set	Get/Set	IP multicast address Configuration	(0)	✓
10	SelectAcd	Get/Set	Get/Set	Activates the use of ACD	(1)	✓
11	LastConflictDetected	Get/Set	Get/Set	Structure containing information related to the last conflict detected	(0)	✓
12	EtherNet/IP Quick Connect	Get/Set	Get/Set	Enable/Disable of Quick Connect feature	(0)	⚠
13	Encapsulation Inactivity Timeout	Get/Set	Get/Set	Number of seconds till TCP connection is closed on encapsulation inactivity	(120)	✓
14	IANA Port Admin	Get	Get/Set	IANA port admin configuration	tcp: 44818 udp: 44818 udp: 2222	✓
768	Client Identifier	Get/Set	Get/Set	Client Identifier used for DHCP Option 61	see below	⚠
769	DHCP Discover Transmission Rate	Get/Set	Get/Set	Maximum transmission interval of DHCP discovery frames in seconds	see below	⚠
1) Activating the Safety Network Number will automatically switch off the support of the Identity object's reset service. The reset service will be reject with general status 0x08 "Service not supported"						

Table 26. TCP/IP Interface Object - Instance attributes

Attribute 768 (0x300) - Client Identifier

Attribute 768 of the TCP/IP Interface Object controls the client identifier used in DHCP option 61. Per default, DHCP option 61 is deactivated. Once the client identifier is set to a nonzero value, DHCP option 61 will be included in the DHCP frames so that a DHCP server can use it to assign the client IP address. For correct client identification it is expected that a unique client identifier is used on the subnet to which the device is attached. The attribute is disabled by default and will be enabled by the host application on demand.

Variable	Type	Value/Range	Description
`bNumBytes`	UINT8	0: Option 61 deactivated 1-48: Option 61 activated 49-255: Reserved for future use Default: 0	Indicates the number of bytes used in the DHCP Client Identifier array below
`abClientId[48]`	UINT8	0 (Default)	DHCP Client Identifier

Table 27. TCP/IP Interface Object – Attribute 768 (0x300)

Attribute 769 (0x301) - DHCP Discover Transmission Rate

Attribute 769 of the TCP/IP Interface Object is of UINT8 data type and controls the maximum transmission interval between two DHCP discovery frames. The application can use this attribute in case the recommended default value (60 seconds) is not suitable and/or cannot be tolerated by a DHCP server. This attribute accepts values between 4 and 60, in seconds. The attribute is disabled by default and will be enabled by the host application on demand.

NOTE The DHCP discovery transmission interval should only be reconfigured if there are significant reasons to deviate from the RFC standard and the implications have been carefully considered.

2.2.12.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Table 28. TCP/IP Interface Object - Common services

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x01	Get Attribute All	☒	✓	Returns content of instance or class attributes
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	☒	✓	Modify attribute value
0x32[1)]	Force Network Interface Reset on next Reconfigure	✓	✓	Request the device to reset its logical link on the next configuration (see: BusOn and BusOff States). This is mostly used for CIP Safety devices.

[1)] Can only be called by the host application

2.2.13 Ethernet Link Object (class code: 0xF6)

The Ethernet Link Object maintains link-specific status information for the Ethernet communications interface. If the device is a multi-port device, it holds more than one instance of this object. Usually, when using the Dual-Port Virtual Ethernet Switch, instance 1 refers to Ethernet port 0 and instance 2 to Ethernet port 1.

2.2.13.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(4)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(2)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(2)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(768)	✓

Table 29. Ethernet Link Object - Class attributes

2.2.13.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Interface Speed	Get	Get	Interface speed currently in use	(100)	✓
2	Interface Flags	Get	Get	Interface status flags	(0x20)	✓
3	Physical Address	Get	Get	MAC layer address		✓
4	Interface Counters	Get	Get	Interface specific counters		✓
5	Media Counters	Get	Get	Media specific counters		✓
6	Interface Control	Get/Set	Get/Set	Configuration for physical interface	(0)	✓
7	Interface Type	Get	Get/Set	Type of interface: twisted pair, fiber	(0x02)	✓
8	Interface State	Get	Get	Current state of interface	(0)	✓
9	Admin State	Get/Set	Get/Set	Administrative state: 1 EIP_EN_INTF_STATE_ENABLE Enable interface 2 EIP_EN_INTF_STATE_DISABLE Disable Interface	(disable)	✓
10	Interface Label	Get	Get/Set	Human readable identification	(“port1”,”port2”)	✓
11	Interface Capability	Get	Get/Set	Indication of capabilities of the interface	10 / HD, 10 / FD, 100 / HD, 100 / FD	✓

768	MDIX	Get/Set	Get/Set	MDIX configuration Format: uint8_t, range [1 .. 3] 1 – EIP_EN_INTF_MDIX_AUTO – Auto detect 2 – EIP_EN_INTF_MDIX_MDI – Explicit MDI 3 – EIP_EN_INTF_MDIX_MDIX – Explicit MDIX	1	✓

Table 30. Ethernet Link Object - Instance attributes

2.2.13.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object’s		Description
		Class Level	Instance Level	
0x01	Get Attribute All	✗	✓	Returns content of instance or class attributes
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✗	✓	Modify attribute value

Table 31. Ethernet Link Object - Common services

2.2.13.4 Class-specific services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object’s		Description
		Class Level	Instance Level	
0x4C	Get and Clear	✗	✓	Retrieves attribute value and subsequently sets the attribute value to zero (only for attributes Interface-Counters and Media-Counters).

Table 32. Ethernet Link Object - Class-specific services

2.2.14 LLDP Management Object (class code: 0x109)

The LLDP Management Object provides the CIP-level interface for the Firmware's implementation of the LLDP protocol.

All information about neighboring devices that is stored in the data tables of the LLDP protocol stack can currently only be accessed using the SNMP protocol (LLDP-MIB, OID 1.0.8802.1.1.2.1). There is no interface for the host application to read the neighboring device information directly.

2.2.14.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(5)	✓

Table 33. LLDP Management Object - Class attributes

2.2.14.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	LLDP Enable	Get/Set	Get/Set	Enables/Disables LLDP global or per port.	All ports enabled	✓
2	msgTxInterval	Get/Set	Get/Set	From 802.1AB-2016. The interval in seconds for transmitting LLDP frames from this device.	(30)	✓
3	msgTxHold	Get/Set	Get/Set	From 802.1AB-2016. A multiplier of msgTxInterval to determine the value of the TTL TLV sent to neighboring devices.	(4)	✓
4	LLDP Datastore	Get	Get	An indication of the retrieval methods for the LLDP database supported by the device.	(0x02) (SNMP)	✓
5	Last Change	Get	Get	The value of sysUpTime taken the last time any entry in the local LLDP database changed.	(0)	✓

Table 34. LLDP Management Object - Instance attributes

2.2.14.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x01	Get Attribute All	☒	✓	Retrieve all attribute values
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	☒	✓	Modify attribute value

Table 35. LLDP Management Object - Common services

2.2.15 Predefined Connection Object (class code: 0x401)

The Predefined Connection Object (PDC) defines and maintains the implicit (class 0/1) connections of the EtherNet/IP Adapter. It is a Hilscher-specific CIP object, which is not covered by the CIP specification.

The PDC object has two purposes:

1. During the configuration phase, let the host application define the set of implicit connections the EtherNet/IP Adapter supports. For each connection, the following parameters are defined:
   - The connection endpoints, a.k.a. Assembly instances for the Input and Output data directions
   - The allowed range of packet intervals (RPI), further limiting the range the protocol stack is technically capable of, if intended
   - The set of connection trigger types supported by the connection
   - The connection type (class 0, class 1, listen only, input only)
2. During the runtime phase, provide information about the current state of the connections.

2.2.15.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device.	0	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class.	0	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(3)	✓

Table 36. Predefined Connection Object - Class attributes

2.2.15.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	State	Get	Get	Provides information about the current state of the connection. FREE (0), UNCONNECTED (1), CONNECTED (2), TIMEOUT (3)	0	✓
2	Count	Get	Get	Indicates how many connections of that type are currently opened.	0	✓
3	Configuration	Get	Get/Set	Connection configuration (see Structure of PDC Instance Attibute 3 - Configuration)		✓

Table 37. Predefined Connection Object - Instance attributes

2.2.15.3 Configuration - Attribute 3

The Configuration attribute 3 indicates a specific implicit connection that can be opened to the EtherNet/IP Adapter.

Name	Byte Size	Description
Consumer Connection Point	4	Connection point addressing the O2T (Originator to Target) direction. Typically, this is an assembly instance number. The value 0xFFFFFFFF serves a wildcard (don't care) purpose. If the wildcard is given, any Assembly of the proper data direction, type and size, will be accepted as the connection endpoint. Specifying explicit connection endpoints is to be preferred over using the wildcard feature for the sake of a clearer system design.
Producer Connection Point	4	Connection point addressing the T2O (Target to Originator) direction. Typically, this is an assembly instance number. The value 0xFFFFFFFF serves a wildcard (don't care) purpose. If the wildcard is given, any Assembly of the proper data direction, type and size, will be accepted as the connection endpoint. Specifying explicit connection endpoints is to be preferred over using the wildcard feature for the sake of a clearer system design.
Configuration Connection Point	4	Connection point addressing a configuration assembly instance. The value 0xFFFFFFFF serves a wildcard (don't care) purpose. If the wildcard is given, any Configuration Assembly of the proper size, will be accepted. Specifying explicit connection endpoints is to be preferred over using the wildcard feature for the sake of a clearer system design.
Minimum O2T RPI	4	Min. RPI of the consuming direction in microseconds
Maximum O2T RPI	4	Max. RPI of the consuming direction in microseconds
Minimum T2O RPI	4	Min. RPI of the producing direction in microseconds
Maximum T2O RPI	4	Max. RPI of the producing direction in microseconds
Supported Trigger Types	1	Supported trigger types of the connection. There can be up to 3 trigger types supported, see section Supported Trigger Types.
Connection type	1	This field specifies the connection application type. The following types are available:

```
#define CIP_CTYPE_EXCLUSIVE_OWNER 0x01
#define CIP_CTYPE_LISTEN_ONLY     0x03
#define CIP_CTYPE_INPUT_ONLY      0x04
```

For more information about application types see [5].

Table 38. Structure of PDC Instance Attibute 3 - Configuration

2.2.15.4 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✗	✓	Modify attribute value
0x08	Create 1)	✓	✗	Create new predefined connection instance
0x09	Delete 1)	✗	✓	Delete predefined connection instance

1) This service is only available to the host application. Initiated from the network, the service will be rejected with general status code 0x08 “Service not supported”.

Table 39. Predefined Connection Object - Common services

2.2.15.5 Create (0x08)

The “Create” service creates a new instance of the Predefined Connection object.

> **NOTE** This service may execute successfully with faulty configurations, e.g. invalid assembly instances. However, inconsistent configuration might lead to a non-functioning connection.

Request Service Data Field Parameters:

The request service data equals the PDC instance attribute 3 structure (see Structure of PDC Instance Attibute 3 - Configuration).

Successful Response Service Data Field Parameters:

The response data to the “Create” service provides the CIP instance number of the newly created Predefined Connection object instance.

Name	Byte Size	Description
CIP instance number that has been created	2	CIP Instance that has been created inside the Predefined Connection class.

Unsuccessful Response Service Data Field Parameters:

The unsuccessful response does not provide any data.

2.2.15.6 Delete (0x09)

The “Delete” service deletes an instance of the Predefined Connection object. Deleting of an instance is only possible if the instance is not participating in an active connection. Otherwise, the service will be answered with general status code 0x0C “Bad Object Mode”.

Request Service Data Field Parameters:

The service does not accept any parameters.

Success Response Service Data Field Parameters:

The service has no response parameters.

Unsuccessful Response Service Data Field Parameters:

The unsuccessful response does not provide any data.

2.2.15.7 Supported Trigger Types

The following trigger type flags can be used:

```
#define CIP_PDC_TTYPE_CYCLIC     0x01 /* Cyclic */
#define CIP_PDC_TTYPE_COS        0x02 /* Change of State */
#define CIP_PDC_TTYPE_APPLICATION 0x04 /* Application Triggered */
```

NOTE the trigger type only affects the message production in the T2O (producing) direction. Which trigger type is used for the connection depends on what the originator of the connection (e.g. PLC) is requesting. Here, we only configure what types the specific connection supports. The following description of the different trigger types reference the “Transmission Trigger Timer” and the “Production Inhibit Timer”. These timers are described in more detail below.

Cyclic

The Transmission Trigger Timer triggers the Message production.

In that case, the message production on the network is completely independent to when the host application updates the data in the DPM (e.g. via xChannelIoWrite). Therefore, the host application can update the producing data at their own rate without having influence on the frames sent on the network.

Application Triggered

Message production is triggered when the application updates the application production data (e.g. via xChannelIoWrite) and by the Transmission Trigger Timer.

The message production triggered by the application additionally depends on the Production Inhibit Timer (see below).

Change of State

Message production is triggered when the application production data has changed (e.g. via xChannelIoWrite) and by the Transmission Trigger Timer. Note: the protocol stack will not check for production data changes. Therefore, the host application is responsible to update production data only if it has changed. The message production triggered by the application additionally depends on the Production Inhibit Timer (see below).

Transmission Trigger Timer

The Transmission Trigger Timer is using the RPI rate the connection originator (e.g. PLC) requested during connection establishment. The expiration of this timer will result in the production of the producing data on the network regardless of the connection’s trigger type.

Production Inhibit Timer

The Production Inhibit Timer applies only to “Change of State” or “Application Triggered” connections. The timer is started only when the application updates the production data (e.g. xChannelIoWrite). Data produced due to the expiration of the Transmission Trigger Timer will not result in a restart of the Production Inhibit Timer (one shot). While the timer is running, the protocol stack suppresses new message production to the network. If one or more new data events occur while this timer is running the protocol stack will produce the most recent new data immediately when it expires. The mechanism intends to limit the production intervals to the lower levels. The originator of the connection can configured the timer via a “Production Inhibit Time” segment attached to the ForwardOpen message. If this segment is not present, the stack will set the timer value to ¼ of the RPI (as defined by CIP).

2.2.16 Diagnosis Object (class code: 0x403)

The diagnosis object provides diagnostic information on the product. Any user may read the diagnostic information through the EtherNet/IP network or the host interface and provide it to the Hilscher support team, precisely identifying the affected product. The diagnostis object is a Hilscher-specific CIP object.

2.2.16.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(9)	✓

Table 40. Diagnosis Object - Class attributes

2.2.16.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Chip info	Get	Get	Name of the used netX chip, data type SHORT_STRING	Product specific	✓
2	OS info	Get	Get	Name of the used operating system, data type SHORT_STRING	Product specific	✓
3	Stack info	Get	Get	Name/Version of the used protocol stack core component, data type SHORT_STRING	Product specific	✓
4	Firmware info	Get	Get	Name/Version of the used EtherNet/IP firmware, data type SHORT_STRING	Product specific	✓
6	Build date	Get	Get	Build date of the used EtherNet/IP firmware, data type SHORT_STRING	Product specific	✓
7	Build type	Get	Get	Build type of the used EtherNet/IP firmware, data type SHORT_STRING	“release”	✓
8	Build host	Get	Get	Build machine name of the used EtherNet/IP firmware, data type SHORT_STRING	Product specific	✓
9	Uptime	Get	Get	Device uptime in seconds, data type UDINT	0	✓

Table 41. Diagnosis Object - Instance attributes

2.2.16.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object’s		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value

Table 42. Diagnosis Object - Common services

2.2.17 IO Mapping Object (class code: 0x402)

The IO Mapping Object is responsible for partitioning of the DPM I/O input and output areas and mapping of those partitions, i.e. members, to the related instances of the Assembly object. This is a Hilscher-specific CIP object, which is not covered by the CIP specification.

2.2.17.1 Class attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get	Revision of this object	(1)	✓
2	Max. Instance	Get	Get	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get	The number of instances currently created in this class	(1)	✓
6	Maximum ID Number Class Attributes	Get	Get	The attribute ID number of the last class attribute of the class definition implemented in the device.	(7)	✓
7	Maximum ID Number Instance Attributes	Get	Get	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(3)	✓

Table 43. IO Mapping Object - Class attributes

2.2.17.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Status	Get	Get	Status of I/O data (Data direction, State of connection)		✓
2	Length	Get	Get	Length of I/O data		✓
3	Data	Get	Get	I/O data		✓

Table 44. IO Mapping Object - Instance attributes

2.2.17.3 Common services

These services are available to the host application and remote EtherNet/IP clients.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✗	✓	Modify attribute value

Table 45. IO Mapping Object - Common services

2.3 Ethernet MAC address

The protocol stack requires one MAC address to operate. This MAC address, as reflected in the corresponding attribute 3 of the CIP Ethernet Link object, is naturally unique in the physical network in which the device operates. Each vendor is responsible for guaranteeing the uniqueness of their device's MAC addresses by assigning them from a certain, IEEE-registered, range.

Per default, the protocol stack applies the MAC address from the underlying Device Data Provider (DDP), which in turn fetches it from either the Security-Memory or Flash Device Data (FDL) sources. The host application cannot set the MAC address CIP attribute directly.

Speaking for the firmware, three to four MAC addresses may be required:

- one for the EtherNet/IP protocol stack itself (reflected in CIP Ethernet-Link attribute 3)
- two additional Port-Mac addresses that are used by LLDP (Link Layer Discovery Protocol)
- one for the Ethernet API on DPM Comm channel 1, which can be enabled statically by means of the taglist (see section Resource and feature configuration via tag list).

The table Ethernet MAC addresses shows the use of the DDP MAC addresses.

Anyway, if the host application seeks to set its own MAC addresses number, e.g. if no SecMem is available, the firmware has to be taglist-modified accordingly as well (refer to section Resource and feature configuration via tag list). Then, it uses the DDP MAC addresses attribute to set a range of custom MACs and set the DDP active. The following pseudo code shows this approach:

```
/* optionally when initial DDP state is passive: set DDP base device parameters: MAC addresses */
HIL_DDP_SERVICE_SET_REQ_T *ptReq = (HIL_DDP_SERVICE_SET_REQ_T*)&myPacket;
uint8_t abMyComMacAddresses[8][6] =
{
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x0 }, /* This is the first chassis MAC address which is used by EtherNet/IP */
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x1 }, /* Port 0 MAC Address used for LLDP                              */
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x2 }, /* Port 1 MAC Address used for LLDP                              */
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x3 }, /* This is the second chassis MAC (Ethernet API)                   */
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x4 },
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x5 },
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x6 },
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x7 },
};
HIL_DDP_SERVICE_SET_REQ_T *ptReq = (HIL_DDP_SERVICE_SET_REQ_T*)&myPacket;
memset(&ptReq->tHead, 0, sizeof(ptReq->tHead));
ptReq->tHead.ulCmd = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen = sizeof(ptReq->tData.ulDataType) + sizeof(abMyComMacAddresses);

/* Set MAC address for the protocol stack (override pre-defined MAC address from FDL) */
ptReq->tData.ulDataType = HIL_DDP_SERVICE_DATATYPE_MAC_ADDRESSES_COM;
memcpy(ptReq->tData.uDataType.atMacAddress, abMyComMacAddresses, sizeof(abMyComMacAddresses));
SendPacket(&myPacket, mychannel);
/* required when inital DPP state is passive: Set DDP active now */
ptReq->tHead.ulCmd = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen = sizeof(ptReq->tData.ulDataType) + sizeof(ptReq->tData.uDataType.ulValue);
ptReq->tData.ulDataType = HIL_DDP_SERVICE_DATATYPE_STATE;
ptReq->tData.uDataType.ulValue = HIL_DDP_SERVICE_STATE_ACTIVE;
SendPacket(&myPacket, mychannel);
```

MAC address	Used for	Mapped to Flash Device Label	Required
1st DDP MAC address	EtherNet/IP communication (DPM channel 0) Socket API communication (DPM channel 1)	MAC address 1 (communication CPU)	Yes
2nd and 3rd DDP MAC address	LLDP communication. One MAC address for each ethernet port	MAC address 2 and 3 (communication CPU)	Yes
4th DDP MAC address	Ethernet API (DPM channel 1)	MAC address 4 (communication CPU)	Required if the Ethernet API is taglist-activated.

Table 46. Ethernet MAC addresses

2.4 Device data

The FDL contains device-specific data that is set during the production of the device. While the firmware starts, it reads this data into the DDP.

The following table lists the device data and describes how the EtherNet/IP Adapter stack maps this data to EtherNet/IP.

Name	EtherNet/IP mapping
Manufacturer ID	Not mapped to EtherNet/IP.
Device class	Not mapped to EtherNet/IP.
Device number	Not mapped to EtherNet/IP.
Serial number	Mapped to Identity Object, attribute 6.
Hardware compatibility number	Not mapped to EtherNet/IP.
Hardware revision number	Not mapped to EtherNet/IP.
Production date	Not mapped to EtherNet/IP.

Table 47. Basic device data in the FDL

The FDL allows storing OEM-specific device data. If used, a consistent set of parameters needs to be provided, i.e. all OEM parameters need to be set and activated, even if the firmware does not use some of the OEM parameters. The following table lists the mapping of the OEM-specific device data to EtherNet/IP.

Name	EtherNet/IP mapping	EtherNet/IP coding
OEM data option flags	In case the parameters from basic device data shall be used, this field shall be set to zero. In case the parameters from OEM identification shall be used, this field shall be set to 0xF.	-
OEM serial number	Mapped to Identity Object, attribute 6.	Null-terminated c string with decimal values "1" ... "4294967295"
OEM order number	Not mapped to EtherNet/IP.	-
OEM hardware revision	Not mapped to EtherNet/IP.	-
OEM production date/time	Not mapped to EtherNet/IP.	-

Table 48. OEM identification in the FDL

2.4.1 Device serial number

Together with the vendor ID, the device serial number (as reflected in the CIP Identity Object, attribute 6) forms a unique identifier for each device on any CIP network. Each vendor is responsible for guaranteeing the uniqueness of the serial number across all devices.

Per default, the protocol stack applies the serial number from the underlying DDP which in turn fetches it from either the SecMem or FDL data sources. The host application cannot set the serial number attribute directly. In the EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ, it may have to parameterize a value of zero for the serial number. This should be fine for most applications.

If the host application seeks to set its own serial number, e.g. if no SecMem is available, the firmware has to be taglist-modified accordingly (refer to section Resource and feature configuration via tag list). Then, it uses the OEM serial number attribute of the DDP to set a custom serial number, to render this data valid, and finally to activate the DDP.

The following pseudo-code illustrates this approach:

```
/* optionally when initial DDP state is passive:
   set additional (OEM) DDP base device parameters*/
HIL_DDP_SERVICE_SET_REQ_T* ptReq = (HIL_DDP_SERVICE_SET_REQ_T*)&myPacket;

memset(ptReq, 0, sizeof(*ptReq));

/* serial number */
char*    szSerialNumber = "76543";
ptReq->tHead.ulCmd      = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen      = sizeof(ptReq->tData.ulDataType) + strlen(szSerialNumber) + 1;
```

```
ptReq->tData.ulDataType = HIL_DDP_SERVICE_DATATYPE_OEM_SERIALNUMBER;

memcpy(ptReq->tData.uDataType.szString, szSerialNumber, strlen(szSerialNumber) + 1);
SendPacket(&myPacket, mychannel);

/* order number */
char*   szOrderNum    = "34567";
ptReq->tHead.ulCmd     = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen     = sizeof(ptReq->tData.ulDataType) + strlen(szOrderNum) + 1;
ptReq->tData.ulDataType = HIL_DDP_SERVICE_DATATYPE_OEM_ORDERNUMBER;

memcpy(ptReq->tData.uDataType.szString, szOrderNum, strlen(szOrderNum) + 1);
SendPacket(&myPacket, mychannel);

/* hardware revision */
char*   szHwRev       = "123";
ptReq->tHead.ulCmd     = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen     = sizeof(ptReq->tData.ulDataType) + strlen(szHwRev) + 1;
ptReq->tData.ulDataType = HIL_DDP_SERVICE_DATATYPE_OEM_HARDWAREREVISION;

memcpy(ptReq->tData.uDataType.szString, szHwRev, strlen(szHwRev) + 1);
SendPacket(&myPacket, mychannel);

/* production date */
char* szProductionDate  = "4321";
ptReq->tHead.ulCmd     = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen     = sizeof(ptReq->tData.ulDataType) + strlen(szProductionDate) + 1;
ptReq->tData.ulDataType = HIL_DDP_SERVICE_DATATYPE_OEM_PRODUCTIONDATE;

memcpy(ptReq->tData.uDataType.szString, szProductionDate, strlen(szProductionDate) + 1);
SendPacket(&myPacket, mychannel);

/* also set the OEM identification "valid" */
ptReq->tHead.ulCmd           = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen           = sizeof(ptReq->tData.ulDataType)
                          + sizeof(ptReq->tData.uDataType.ulValue);
ptReq->tData.ulDataType       = HIL_DDP_SERVICE_DATATYPE_OEM_OPTIONS;
ptReq->tData.uDataType.ulValue = 0xF; /* set OEM identification valid */

SendPacket(&myPacket, mychannel);

/* required when initial DPP state is passive: Set DDP active now */
ptReq->tHead.ulCmd           = HIL_DDP_SERVICE_SET_REQ;
ptReq->tHead.ulLen           = sizeof(ptReq->tData.ulDataType)
                          + sizeof(ptReq->tData.uDataType.ulValue);
ptReq->tData.ulDataType       = HIL_DDP_SERVICE_DATATYPE_STATE;
ptReq->tData.uDataType.ulValue = HIL_DDP_SERVICE_STATE_ACTIVE;

SendPacket(&myPacket, mychannel);
```

NOTE OEMization is EtherNet/IP-specific. Other software components will reflect the Hilscher serial number from the basic device data anyway instead of the OEM-data, e.g. the netIdent/EtherNetDeviceConfig subsystem.

2.5 Status information

2.5.1 DPM communication status

This section describes how the EtherNet/IP Adapter uses the communication status. The communication status is located in the DPM as described in netX DPM Interface manual [1].

State	Description
OFFLINE	The device is not configured. No frames are generated.
STOP	The device is configured and Bus OFF is set. The device is not responsive to network communication.
IDLE	The device is configured and Bus ON is set, but the device has no open connections (class0, class1 or class3).
OPERATE	The device is configured and Bus ON is set. The device has at least one open connection (class0, class1 or class3). During this communication state, also the COM Bit (NCF_COMMUNICATING) will be set.

Table 49. Communication states

The following figure shows how the communication status transitions depend on specific events.

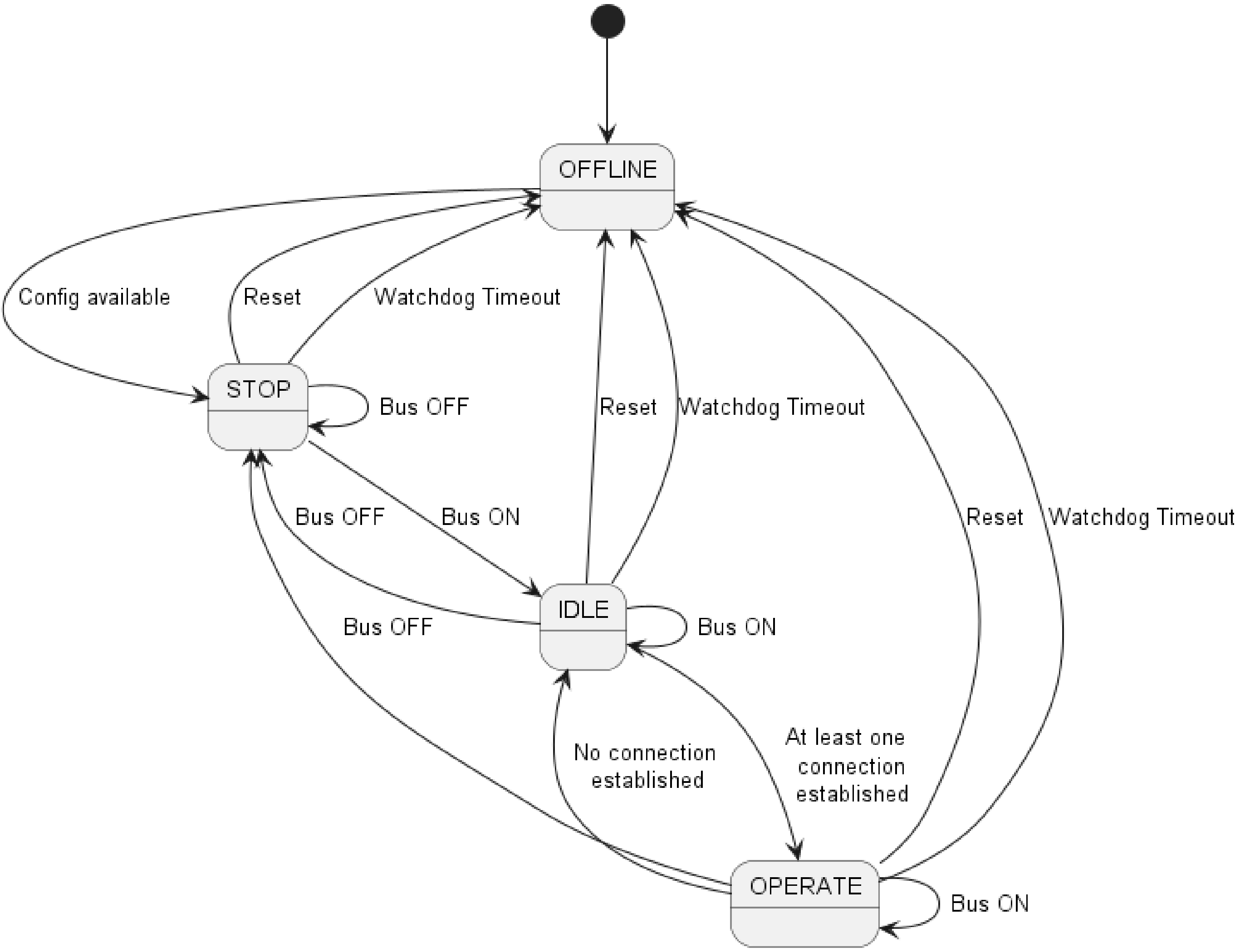


Figure 3. Communication status transitions

2.5.2 DPM COS flags

Flag	Description
DPM flag CONFIGURATION NEW	The EtherNet/IP protocol stack does not handle the DPM flag HIL_COMM_COS_CONFIG_NEW located in the communication status field (ulCommunicationCOS).
DPM flag RESTART REQUIRED	The EtherNet/IP protocol stack does not handle the DPM flag HIL_COMM_COS_RESTART_REQUIRED located in the communication status field (ulCommunicationCOS).
DPM flag RESTART REQUIRED ENABLE	The EtherNet/IP protocol stack does not handle the DPM flag HIL_COMM_COS_RESTART_REQUIRED_ENABLE located in the communication status field (ulCommunicationCOS).

Table 50. DPM COS flags

2.5.3 Other DPM status bits

Bit `NCF_COMMUNICATING` will be set for the DPM channel if at least one CIP class 0, class 1 or class 3 connection is open for which we are the target.

The protocol stack updates the bit in a low-priority path, asynchronously with the processing of high-priority I/O data. Thus, it is not suitable for the application to decide on whether or not input data of the channel is “valid”. Instead, the feature `EIP_AS_OPTION_MAP_RUNIDLE` can be used to retrieve such status information. Unless `EIP_AS_OPTION_HOLDLASTSTATE` is used, the input data of a connection will read all zeroes when no connection is established or a connection is not (yet) in the RUN status.

2.6 Module and network status

This section describes the LED signaling of EtherNet/IP Adapter devices.

Two LEDs display status information:

- the module status LED (MS)
- the network status LED (NS)

2.6.1 Module status

The following table lists the possible module status values and their meanings (Parameter `ulModuleStatus`):

Symbolic name	Numeric value	Meaning
EIP_MS_NO_POWER	0	**No power** If no power is supplied to the device, the module status indicator is permanently off.
EIP_MS_SELFTEST	1	**Self-test** While the device is performing its power-on testing, the module status indicator is flashing green/red.
EIP_MS_STANDBY	2	**Standby** If the device has not been configured, the module status indicator is flashing green.
EIP_MS_OPERATE	3	**Device operational** If the device is operating correctly, the module status indicator is permanently green.
EIP_MS_MAJOR_RECOVERABLE_FAULT	4	**Major recoverable fault** If the device has detected a major recoverable fault, the module status indicator is flashing red. **Note**: An incorrect or inconsistent configuration is considered a major recoverable fault.
EIP_MS_MAJOR_UNRECOVERABLE_FAULT	5	**Major unrecoverable fault** If the device has detected an unrecoverable major fault, the module status indicator is permanently red.

Table 51. Possible values of the module status

2.6.2 Network status

The following table lists the possible network status values and their meaning (Parameter `ulNetworkStatus`):

Symbolic name	Numeric value	Meaning
`EIP_NS_NO_POWER`	0	**Not powered, no IP address** Either the device is not powered or it is powered, but no IP address has been configured yet. The network status indicator LED is off.
`EIP_NS_NO_CONNECTION`	1	**No connections** An IP address has been configured, but no CIP connections are established, and an exclusive owner connection has not timed out. The network status indicator is flashing green.
`EIP_NS_CONNECTED`	2	**Connected** At least one CIP connection of any transport class is established, and an exclusive owner connection has not timed out. The network status indicator is permanently green.
`EIP_NS_TIMEOUT`	3	**Connection timeout** An exclusive owner connection for which this device is the target has timed out. The network status returns to `EIP_NS_CONNECTED` when connections to all those consumer connection points are reestablished, whose connections previously timed out.
`EIP_NS_DUPIP`	4	**Duplicate IP** The device has detected that its IP address is already in use. The network status indicator is permanently red.
`EIP_NS_SELFTEST`	5	**Self-Test** The device is performing its power-on self-test (POST). During POST, the network status indicator is flashing green and red alternately.

Table 52. Possible values of the network status

2.7 Handshake modes

The handshake mode is get and set by means of the services HIL_GET_TRIGGER_TYPE_REQ and HIL_SET_TRIGGER_TYPE_REQ. The EtherNet/IP protocol stack offers the following handshake modes for exchange of process data with the host application and global time synchronization:

Input handshake mode	Output handshake mode	Synchronization handshake mode
Free-running Receive (RX) triggered	Free-running	Disabled Enabled

Table 53. Supported handshake modes

2.7.1 Input handshake mode / output handshake mode

Free-running handshake mode: This is the default handshake mode for input and output data, i.e. assemblies of the types `EIP_AS_TYPE_INPUT` and `EIP_AS_TYPE_OUTPUT`. In this mode, when the host reads I/O data from or writes I/O data to the protocol stack, control over the input or output areas, is given to the protocol stack which immediately copies the current process data into (or out of) the DPM and returns the control over the area to the host. If no valid input data is available, e.g. if the Run bit was not set in the previous process data telegram, unless otherwise specified, the copied data read as zeroes. The free-running handshake mode is the only supported mode for the output handshake.

Receive (RX) triggered handshake mode: This mode is available for assemblies that consume data from the network, i.e. assemblies of type `EIP_AS_TYPE_INPUT`. If this handshake mode is configured, at least one input assembly has to be present and be marked as a trigger assembly with the option flag `EIP_AS_OPTION_RXTRIGGER`, see table Assembly Types and Option Flags. In this mode, when the host attempts to read process data from the protocol stack, control over the DPM input area is given to the protocol stack. The stack will then wait for the next (current) process data from the network directed toward one of the trigger assemblies, copy that data into the DPM and pass control over the input area back to the application. If there are multiple trigger assemblies, it is undefined which subset of them has possibly been updated and the host application has to take further means to decide on the age of the data segment of each assembly. For details, see assembly option flag `EIP_AS_OPTION_MAP_SEQCOUNT` in table Assembly Types and Option Flags.

NOTE The handshake mode, once configured, is kept until explicitly reconfigured to another mode or the protocol stack reboots (due to a physical reset or a "warm restart" of the firmware).

NOTE During the design phase of your project we encourage you to decide on whether a time-triggered (sync handshake) or an event-triggered (receive-triggered) system suits your needs best and to opt for one type. Although it is technically possible to combine both modes, using only one method leads to a clear design and a less complex implementation.

NOTE If the handshake mode “receive (RX) triggered” is used and if currently no I/O connection is established towards any assembly that is eligible for triggering, control over the input area will not be returned to the host until a new connection is established and a first I/O frame is received.

2.7.2 Synchronization handshake mode

Irrespective of the input / output handshake modes, a synchronization handshake based on the EtherNet/IP TimeSync Module according to the CIPSync device specification is available in the following two modes:

Synchronization handshake disabled: The protocol stack will not trigger a synchronization handshake. This is the default.

Synchronization handshake enabled: A synchronization interrupt is generated periodically in the synchronization interval as specified by the TimeSync Object synchronization parameters while the clock is synchronized to the master clock.

For a detailed description of CIP Sync, see Application Note [4].

2.7.3 Configuration

To configure the handshake mode, use Set Trigger Type Request HIL_SET_TRIGGER_TYPE_REQ. For details, see section HIL_SET_TRIGGER_TYPE_REQ or reference [9].

Example 1

Configure input handshake mode “Free-running” and "Synchronization handshake enabled"

```
HIL_SET_TRIGGER_TYPE_REQ_T tReq;
tReq.tData.usPdInHskTriggerType  = HIL_TRIGGER_TYPE_PDIN_NONE;
tReq.tData.usPdOutHskTriggerType = HIL_TRIGGER_TYPE_PDOUT_NONE;
tReq.tData.usSyncHsdkTriggerType = HIL_TRIGGER_TYPE_SYNC_TIMED_ACTIVATION;
tReq.tHead.ulLen             = HIL_SET_TRIGGER_TYPE_REQ_SIZE;
tReq.tHead.ulCmd              = HIL_SET_TRIGGER_TYPE_REQ;
tReq.tHead.ulId              = 0;
SendPacket(&tReq);
```

Example 2

Configure input handshake mode “RX triggered” and "Synchronization handshake disabled"

```
HIL_SET_TRIGGER_TYPE_REQ_T tReq;
tReq.tData.usPdInHskTriggerType  = HIL_TRIGGER_TYPE_PDIN_RX_DATA_RECEIVED;
tReq.tData.usPdOutHskTriggerType = HIL_TRIGGER_TYPE_PDOUT_NONE;
tReq.tData.usSyncHsdkTriggerType = HIL_TRIGGER_TYPE_SYNC_NONE;
tReq.tHead.ulLen             = HIL_SET_TRIGGER_TYPE_REQ_SIZE;
tReq.tHead.ulCmd              = HIL_SET_TRIGGER_TYPE_REQ;
tReq.tHead.ulId              = 0;
SendPacket(&tReq);
```

2.8 Quality of Service

2.8.1 Introduction

Quality of Service (QoS) is a mechanism that treats data streams according to their delivery characteristics. The most important characteristic is the priority of the data stream. In the context of EtherNet/IP, QoS is priority-dependent control of Ethernet data streams.

QoS is of special importance for advanced time-critical applications such as CIP Sync and CIP Motion and is mandatory for DLR (see section Device Level Ring).

Introducing QoS in a network affects the whole network infrastructure such as switches to consider each data stream's priority. QoS-capable devices write priority information into the frames, and are able to process different priorities when such prioritized frames are received.

TCP/IP-based protocols, such as EtherNet/IP, have two standard mechanisms for implementing QoS which we describe in the following subsections:

- Differentiated Services (DiffServ)
- 802.1D/Q protocols (not supported with the current version of the EtherNet/IP stack)

2.8.2 DiffServ

In the definition of an IP v4 frame, the second byte is denominated as TOS. See figure below:

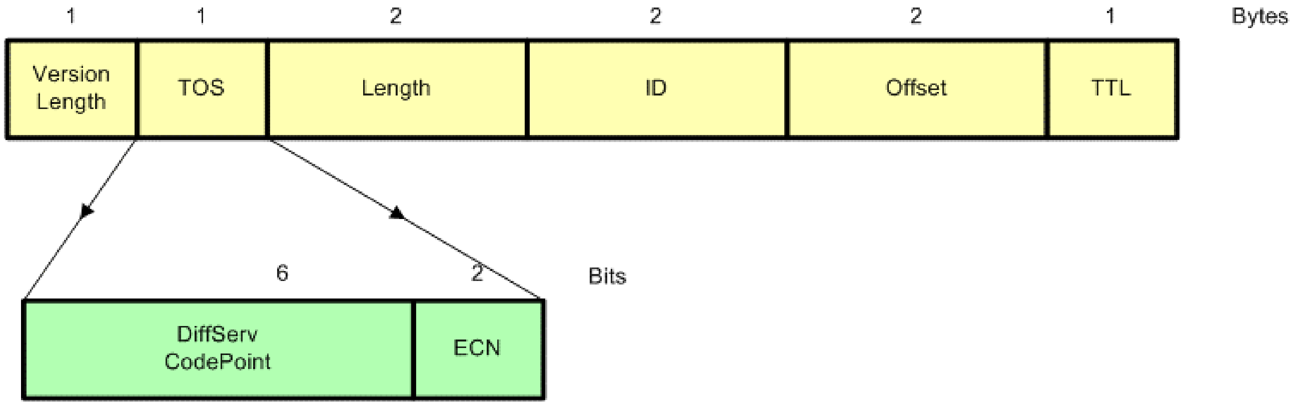


Figure 4. TOS Byte in IP v4 Frame Definition

DiffServ is a schematic model for the priority-based classification of IP frames based on an alternative interpretation of the TOS byte. DiffServ is specified in RFC 2474.

The idea of DiffServ consists in redefining 6 bits (i.e. bits 8 to 13 of the whole IP v4 frame) and using them as a code point. Thus, these 6 bits are denominated as DSCP (*Differentiated Services Code Point*) in the context of DiffServ. These 6 bits allow addressing 63 predefined routing behaviors, which can be applied for routing the frame at the next router, and specify exactly how to process the frame there. These routing behaviors are called PHBs (Per-Hop behavior). Many PHBs have been predefined and the IANA has assigned DSCPs to them. For a list of these DSCPs and the assigned PHBs, see http://www.iana.org/assignments/dscp-registry/dscpregistry.xhtml.

Mapping of DSCP to EtherNet/IP

The following table shows the default assignment of DSCPs to different kinds of data traffic in EtherNet/IP (according to the CIP specification).

Traffic type	CIP priority	DSCP (numeric)	DSCP (bin)
CIP class 0 and 1	Urgent (3)	55	110111
	Scheduled (2)	47	101111
	High (1)	43	101011
	Low (0)	31	011111
CIP class 3 CIP UCMM All other encapsulation messages	All	27	011011

Table 54. Default assignment of DSCPs in EtherNet/IP

2.8.3 802.1D/Q Protocol

802.1Q uses another possibility. IEEE 802.1Q is a standard for defining virtual LANs (VLANs) on an Ethernet network. It introduces an additional header (the IEEE 802.1Q header) located between Source MAC and Ethertype and Size in the standard Ethernet frame.

The IEEE 802.1Q header has the Ethertype 0x8100. It allows specifying

- the ID of the VLAN (VLAN ID, 12-bit wide) and
- the priority (defined in 802.1D)

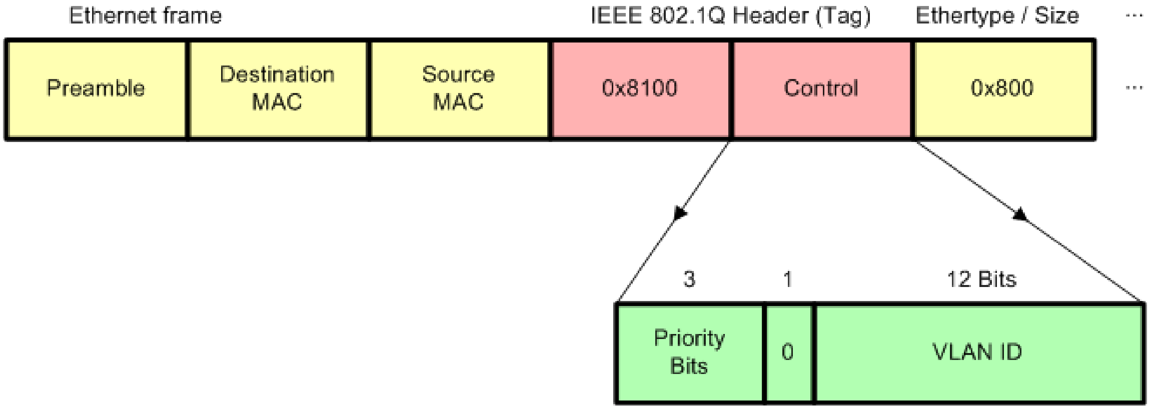


Figure 5. Ethernet frame with IEEE 802.1Q header

Since the header definition reserves only 3 bits for the priority, only 8 priorities (levels from 0 to 7) can be used here.

Mapping of 802.1D/Q to EtherNet/IP

The following table shows the default assignment of 802.1D priorities to different kinds of data traffic in EtherNet/IP (according to the CIP specification).

Traffic type	CIP priority	802.1D priority
CIP class 0 and 1	Urgent (3)	6
	Scheduled (2)	5
	High (1)	5
	Low (0)	3
CIP class 3 CIP UCMM All other encapsulation messages	All	3

Table 55. Default assignment of 802.1D/Q priorities in EtherNet/IP

2.8.4 The QoS Object

The Quality of Service object provides the CIP interface towards the QoS subsystem.

DiffServ is always active and the DiffServ parameters (DSCP Values) for the different communication priorities are set through the attributes of the object.

Importantly, the 802.1Q feature for VLAN tagging of CIP and IEEE1588 frames is not supported with this version of the EtherNet/IP Stack. The feature was supported in older generations of the product and will be made available with a future stack version. Currently however, attribute 1 of the QoS object is disabled/unavailable for its purpose of enabling/disabling the 802.1Q mechanism.

For details on the QoS object in the Hilscher EtherNet/IP Adapter protocol stack, see section Quality of Service Object (class code: 0x48).

2.8.4.1 Enable 802.1Q (VLAN tagging)

The feature is currently not supported. Attribute 1 of the QoS object would allow to enable and disable the mechanism for firmwares with full 802.1Q support.

2.9 Device Level Ring

This section gives a brief overview of the basics and concepts of the Device Level Ring (DLR) networking technology supported by Hilscher's EtherNet/IP Adapter protocol stack.

DLR is a technology for creating a single ring topology with media redundancy. DLR is based on Layer 2 (Data link) of the ISO/OSI model of networking and thus transparent for higher layers (except the existence of the DLR object providing configuration and diagnosis).

In general, there are two types of devices in the network:

- Ring supervisors
- Ring nodes

DLR requires that all devices be equipped with two Ethernet ports and internal switching technology. Each sent frame is forwarded on both ports, in both directions, through the ring.

On receiving the frame, each device within the DLR network checks whether the target address in the frame matches the MAC address of the device.

- If the frame is targeting the MAC address of the device, the device will process the frame, which will not be forwarded any further through the ring.
- If the frame targets another MAC, the device forwards the frame on the other port to the next ring node.

To technically achieve a line topology and prevent looping frames, the active ring supervisor disables one of its ports.

2.9.1 Ring supervisors

Two types of supervisors are defined:

- active supervisors
- back-up supervisors

NOTE | The Hilscher EtherNet/IP stack does not support the ring supervisor mode.

Active supervisors

Tasks:

- It periodically sends beacon and announce frames.
- It permanently verifies the ring integrity.
- It reconfigures the ring to ensure operation in case of single faults.
- It collects diagnostic information from the ring.

Exactly one active ring supervisor is required within a DLR network.

Back-up supervisors

It is recommended (but not required) that each DLR network have at least one back-up supervisor. If the active supervisor of the network fails, the back-up supervisor will take over and become the active ring supervisor. For this purpose, a precedence value is assigned to each supervisor. The supervisor with the highest precedence becomes the active ring supervisor. The others remain passive in the role of back-up supervisors.

2.9.2 Beacon and announce frames

Beacon frames and announce frames both serve to inform the devices within the ring of the transition (i.e. the topology change) from linear operation to the ring operation of the network.

They differ in:

Direction

- Beacon frames are sent in both directions.
- Announce frames are sent only in one direction of the ring.

Frequency

- Beacon frames are sent periodically at every beacon interval (typically at intervals of 400 microseconds). Announce frames are sent once per second.

Support for precedence number

- Only beacon frames contain the internal precedence number of the supervisor which sent them

Support for network fault detection

- The loss of beacon frames allows the active supervisor to detect and discriminate various types of network faults in the ring.

2.9.3 Ring nodes

This subsection deals with the ring modules without supervisor capabilities, the so-called (normal) ring nodes.

The network has two types of normal ring nodes:

- beacon-based nodes
- announce-based nodes

A DLR network may contain an arbitrary number of normal nodes.

Capabilities

Beacon-based nodes:

- implement the DLR protocol, but without ring supervisor capability
- must be able to process beacon frames with hardware assistance

Announce-based nodes:

- implement the DLR protocol, but without ring supervisor capability
- forward beacon frames without processing them
- must be able to process announce frames
- are often only a software solution

NOTE | The EtherNet/IP firmware always runs as a beacon-based ring node.

2.9.4 Normal network operation

In normal operation, the supervisor sends beacon and announce frames to monitor the state of the network. Usual ring nodes and back-up supervisors receive these frames and react. The supervisor sends announce frames once per second and, additionally, if an error is detected.

2.9.5 Rapid fault/restore cycles

Sometimes a series of rapid faults and restore cycles may occur in the DLR network, e.g. if a connector is faulty. If the supervisor detects 5 faults within 30 seconds, it will set a flag (Rapid fault/Restore cycles). The user then has to reset this flag explicitly via the Clear Rapid Faults service.

2.10 CIP device protection

2.10.1 Introduction

CIP device protection refers to the mechanism that protects the configuration of a device against changes, which would disrupt the operational state. As a prime example, it is to avoid changes to the IPv4 configuration settings while a device is participating in an I/O connection.

This section gives an overview of the device protection, protection modes, protection policy and Hilscher's implementation of device protection.

For the complete description of device protection mode, see: *CIP specification, volume 1, section 5A-2 - Identity object*. [5]

2.10.2 Protection modes

The protection mode is mapped to attribute 19 of the Identity object.

The CIP specification defines two different protection modes:

- **Implicit protection**
  - enabled implicitly when at least one active I/O connection is established with the device
  - disabled as soon as the last I/O connection closes
  - not modifiable programmatically
- **Explicit protection**
  - may be set explicitly by the application on demand using bit `CIP_ID_PROTECTION_MODE_EXPLICIT_PROTECTION` in the value of the attribute

The effect of the enabled device protection (implicit or explicit) is that a well-known set of object attributes becomes immutable and certain services become unavailable. The following sections describe this in detail.

2.10.3 Protection policy

Enabled device protection, regardless of whether it is due to implicit a/o explicit protection, has the following effects:

1. A request to the **Reset service** of the Identity Object will be rejected with the reply `CIP_GSR_DEV_IN_WRONG_STATE` (0x10).
2. A **Set Attribute Single** request will be rejected with the reply `CIP_GSR_DEV_IN_WRONG_STATE` (0x10) if the request targets an attribute that is subject to the protection policy.

Per default, the attributes of the following table are subject to the protection policy:

Class	Instance ID	Attribute ID	Attribute name
TCP/IP interface (0xF5)	1	3	Configuration Control
		5	Interface Configuration
		6	Host Name
		8	TTL Value
		9	Mcast Config
		10	SelectAcd
		12	EtherNet/IP QuickConnect
Ethernet Link (0xF6)	1, 2	6	Interface Control
		9	Admin State
		768	MDIX Config

Class	Instance ID	Attribute ID	Attribute name
Quality of Service (0x48)	1	1	802.1Q Tag Enable
		2	DSCP PTP Event
		3	DSCP PTP General
		4	DSCP Urgent
		5	DSCP Scheduled
		6	DSCP High
		7	DSCP Low
		8	DSCP Explicit

Table 56. Hilscher’s default protection policy

The user application can modify the protection policy as described in section EIP_OBJECT_ENABLE_DISABLE_ATTRIBUTE_PROTECTION_REQ.

2.11 Module and Network Status LEDs

A CIP device typically has two LED status indicators, which we refer to as MS-LED and NS-LED, and which are supposed to reflect the Module Status and Network Status of the Device.

Normally, the LEDs are implicitly controlled by the protocol stack to indicate the current behaviorial state of the device according to a default mapping. The host application may however take explicit control of the LEDs and implement a different mapping. This is further described in section EIP_OBJECT_FORCE_LED_STATE_REQ.

Basically, the default mapping of the device state to the LED indicators is defined by the CIP specification. Since these definitions are rather unspecific, we give a more detailed description of the LEDs from a vendor-specific perspective in the following table.

<table>
<tr><th>Device state</th><th>MS-LED</th><th>NS-LED</th></tr>
<tr><td>The device is not powered</td><td>off</td><td>off</td></tr>
<tr><td>Self-test due to power-on, reboot or (CIP) reset</td><td>red/green blinking</td><td>red/green blinking</td></tr>
<tr><td>A major recoverable fault has occurred</td><td>red blinking</td><td>undefined</td></tr>
<tr><td>A major unrecoverable fault has occurred</td><td>solid red</td><td>undefined</td></tr>
<tr><td>No (valid) configuration has been applied.</td><td>blinking green</td><td>off</td></tr>
<tr><td>A (valid) configuration has been applied through either:<ul><li>A database that has been downloaded and applied</li><li>The simple configuration packets EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ and HIL_CHANNEL_INIT_REQ</li><li>Extended configuration steps finished with EIP_APS_CONFIG_DONE_REQ</li></ul></td><td>solid green</td><td>According to the following table
<table>
<tr><th>Network status</th><th>NS-LED state</th></tr>
<tr><td>No valid IP is yet assigned to the device’s network interface</td><td>off</td></tr>
<tr><td>An IP address conflict has been detected by the ACD</td><td>solid red</td></tr>
<tr><td>A valid IP was assigned to the device’s network interface</td><td>blinking green</td></tr>
<tr><td>At least one CIP class 0/1/3 connection has been opened in the device</td><td>solid green</td></tr>
<tr><td>At least one exclusive owner connection that has been open previously has timed out and was not reopened yet</td><td>blinking red</td></tr>
</table></td></tr>
</table>

Table 57. Default mapping between device states and LED indicators

2.12 DHCP/BOOTP Client

The Hilscher EtherNet/IP provides the following methods for IP configuration:

1. Static configuration, where a fixed IP address is set into attribute 5 of the TCP/IP object instance.
2. Dynamic configuration through either the BOOTP or DHCP protocols.

The configuration method is selected through attribute 3 of the CIP TCP/IP Interface Object as described in section TCP/IP Interface Object (class code: 0xF5).

This section specifically describes the aspects of dynamic configuration, i.e. retrieving an IP configuration from the a BOOTP/DHCP server. Typically, this server maintains a static list to uniquely map a set of MAC addresses to IP addresses and assigns these relations on client request. The DHCP protocol is defined in RFC2131.

Subsequently, we will be using the term DHCP exclusively to describe both, the DHCP and BOOTP client protocols implemented in the EtherNet/IP stack. BOOTP can be thought of a subset of DHCP. Where the behavior or capabilities differ, it is mentioned explicitly.

2.12.1 DHCP Behavior

When configured at the TCP/IP Interface Object, DHCP will become active immediately. If a previous, static IP configuration was active, all active connections will close and the network interface is set down, disrupting any IP communication in the netX-based system.

The DHCP state machine will then try to discover a DHCP server in increasing intervals of up to 60 seconds until a DHCP server responds and assigns the device an IP configuration. After optionally ACD-probing, this IP address will eventually be used by the device and therefore set into attribute 5 of the TCP/IP Interface object. The maximum discovery interval can be modified through TCP/IP Interface Object Attribute 769. We recommend a maximum interval of 60 seconds.

If the DHCP server has assigned lease time, renewal time or rebinding time intervals, the DHCP state machine will become active again and perform the corresponding operation in time. This may lead to the previously assigned IP configuration to be dropped autonomously (typically only on misconfiguration or DHCP server failure). The DHCP client state machine will also get active in case of the events described below.

2.12.2 DHCP Device Level Behavior

The DHCP Client will typically perform **DHCP-DISCOVERY**, based on UDP broadcast from source IP address 0.0.0.0 in case of at least the following events:

1. On PowerOn, after the first successful configuration, if DHCP mode is configured.
2. A CIP Reset is performed through the Identity Object.
3. The network link was lost and is reestablished, may it be through a cable disconnect or an internal reconfiguration of the network PHY which interrupted the link.
4. The value of either attribute 768 or 769 of the TCP/IP interfcae object is modified.
5. DHCP-Rebinding failed.

The DHCP Client will perform **DHCP-RENEWAL**, based on UDP unicast from a valid previously assigned IP address in case of at least the following events:

1. A BusOff/BusOn cycle is performed (see Bus State).
2. Channel initialization (see HIL_CHANNEL_INIT_REQ).
3. A DHCP-Renewal interval has been assigned by the DHCP server and has exceeded

The DHCP Client will typically perform **DHCP-REBINDING**, based on UDP broadcast from a valid previously assigned IP address in case of at least the following events:

1. A DHCP-Rebinding interval has been assigned by the DHCP server and has exceeded.
2. DHCP-Renewal failed.

2.12.3 Packet API

The host application cannot control the DHCP client directly, but through the CIP TCP/IP Interface Object only. The behavior and state of the DHCP client is mostly transparent as well and cannot be observed explicitly.

As an exception, in case the EtherNet/IP-Stack drops the IP address (which is common if a fresh DHCP DISCOVERY cycle is started, or when the lease expires) the following effect can be observed at the packet API and at the CIP interface level:

The IP address of the TCP/IP interface will be set to 0.0.0.0 in order to DHCP-discover from that IP. The TCP/IP interface is then unusable for communication, until a new DHCP lease has been acquired. This IP 0.0.0.0 will be set into the CIP object dictionary (TCP/IP Interface Object instance attribute 5) in the meantime and a EIP_OBJECT_CIP_OBJECT_CHANGE_IND for the attribute will be generated to indicate the condition to the host application, as it is done for all changes of this attribute.

However, we have to emphasize the special semantics of this change indication for its indicative and temporary character. The attribute 5 of the TCP/IP interface Object ambiguously serves two purposes and has transactional semantics:

1. In case of static configuration, write access to the attribute allows the host application to set the IP configuration
2. Read access to the attribute allows the host application to query the current IP configuration. This applies for both configuration methods, static and, more importantly, dynamic IP configuration.

The value of attribute 5 of the TCP/IP interface Object is normally stored remanently, i.e the last valid IP address of the device is retained over power cycles. As an exception to this behavior, an IP address of 0.0.0.0, when set into attribute 5 to indicate a ongoing DHCP cycle or a missing ethernet link, will never be stored remanently. It has a strictly indicative purpose. All nonzero IPs, in contrast, will be stored remanently when set into attribute 5. The value of attribute 5 may change actively when set by the host application (only in case of static IP configuration) and/or will change passively due to either:

1. A CIP Identity reset
2. A CIP client modifying attribute 3 over the network.
3. A CIP client modifying attribute 5 over the network.
4. DHCP/BOOTP restarting DHCP-discovery
5. Link loss: A lost ethernet link which also disrupts the IP-level
6. Any other netX subsystem, e.g. the Hilscher Ethernet Device Configuration Tool

Only valid (nonzero) IP configurations will be stored remanently and to indicate otherwise, the EIP_OBJECT_CIP_OBJECT_CHANGE_IND packet will have the flag `EIP_OBJECT_CIP_OBJECT_CHANGE_NV_STORING_BYPASSED` set in the field `ulInfoFlags`.

2.12.4 DHCP Options

Next to the IP address, the EtherNet/IP Stack's DHCP client is capable of assigning further IP configuration parameters as per at least the following DHCP-options:

Option ID	Option name	Description/Usage
1	subnet mask	Assigns a subnet mask. If no such option is given, a class A, B or C network subnet mask will be set according to the given IP. The subnet mask will be set into the TCP/IP Interface Object instance attribute 5
3	router	Assigns a gateway/router IP address. The gateway address will be set into the TCP/IP Interface Object instance attribute 5. The gateway will be subsequently be addressed for IP communication for target addresses which are not in the device's subnet.
6	dns server	Assigns up to four name server IP addresses of which the first two will be set into the TCP/IP Interface Object instance attribute 5. Since no DNS features are implemented, the name servers have no further effects in the device.
12	hostname	Assigns a hostname. If given, the host name is set into the TCP/IP Interface Object instance attribute 6. Since no DNS features are implemented, the hostname has no further effect in the device.
15	domainname	Assigns a domainname. If given, the domain name is set into TCP/IP Interface Object instance attribute 5. Since no DNS features are implemented, the domainname has no further effect in the device.
51	lease time	Assigns a lease time for the given configuration. The lease time will not be directly observable through the APIs. If assigned, the device will drop the IP address after the given lease time expires.

Option ID	Option name	Description/Usage
58	T1, renewal time	Assigns a renew interval to the DHCP state machine so that it will renew the acquired IP lease at the DHCP server when expired. The renewal time is not directly observable through the APIs. If renewal fails, the state machine will fall back to rebinding state. Renews use UDP unicasts with a nonzero source IP address.
59	T2, rebinding time	Assigns a rebinding interval to the DHCP state machine so that it will rebind the current IP address at a DHCP server. The rebinding time is not directly observable through the APIs. Rebinds use UDP broadcasts with a nonzero source IP address.
61	DHCP Client identifier	Configured through TCP/IP Interface object attribute 768, if set to a non-empty value, the DHCP client will include the given client identifier in DHCP Request and ACK messages so that the DHCP server may assign the device's IP address based on that identifier.

Table 58. DHCP options that are at least supported by the EtherNet/IP Stack

2.13 QuickConnect

QuickConnect serves to establish an I/O connection quickly after power-on of the device.

After power-on, a connection to the adapter has to be established in 500 ms or less. The device has to be ready to accept TCP connections in less than 350 ms after power-on. This is required by the CIP specification.

When the system starts, the QuickConnect function enables the network PHYs with fixed duplex modes and MAU types. This saves linking time and costs for automatic detection.

QuickConnect devices must be able to set the forced speed/duplex mode (at least for 10/100 MB Ethernet) via the Ethernet Link object. On devices with two external Ethernet ports, each port has a name to allow identification: Port 1 (channel 0) and port 2 (channel 1). The configuration of port 1 (channel 0) is MDI, that of port 2 (channel 1) is MDIX.

QuickConnect allows the protocol stack to skip the ACD probing stage of 2 s after power-on (during which the device figures out whether it can use the IP address without risking collisions).

QuickConnect limits the number of ARP announces of the ACD mechanism to the network to max. 40 (1 per 25 ms) or to enter the detection phase as soon as the I/O connection is established.

The use of the QuickConnect function requires adding specified keywords and values to a device EDS file.

The TCP/IP object attribute 12 enables/disables the QuickConnect function.

For configuring QuickConnect, the application can use one of the two configuration packet sets:

- To control the QuickConnect configuration, attribute bQuickConnectFlags provides two bits. For details, see table EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T – Configuration Parameter Set V3.
- For the extended configuration set, the application has to manually enable the attribute (see section EIP_OBJECT_ENABLE_ATTRIBUTE_REQ) and set it to the desired value during the configuration phase with service EIP_OBJECT_CIP_SERVICE_REQ, Set_Attr_Single.

When enabled, QuickConnect becomes effective with the next power-on.

When effective, attributes 6 and 768 of the EtherNetLink object will be set to fixed values during system start (Port 1: 100 Mbit/s FDX, MDI and Port 2: 100 Mbit/s, FDX, MDIX), overwriting the current device settings. When QuickConnect is disabled, the previous settings will get active again.

While QuickConnect is active, changes to attribute 6 of the EtherNetLink object are rejected.

Chapter 3 Getting started / configuration

3.1 Configuration methods

The EtherNet/IP Adapter stack requires configuration parameters. The protocol stack offers the following configuration methods:

1. The application can set the configuration parameters using the Basic Configuration Packet Set (see section Configuration using the packet API).
2. The application can set the configuration parameters using the Extended Configuration Packet Set (see section Configuration using the packet API).
3. The stack can be configured by using the configuration software SYCON.net. This tool creates a database that is loaded into the file system of the netX.

3.2 Host application behavior

The following diagram gives an overview of the different scenarios of host application behavior.

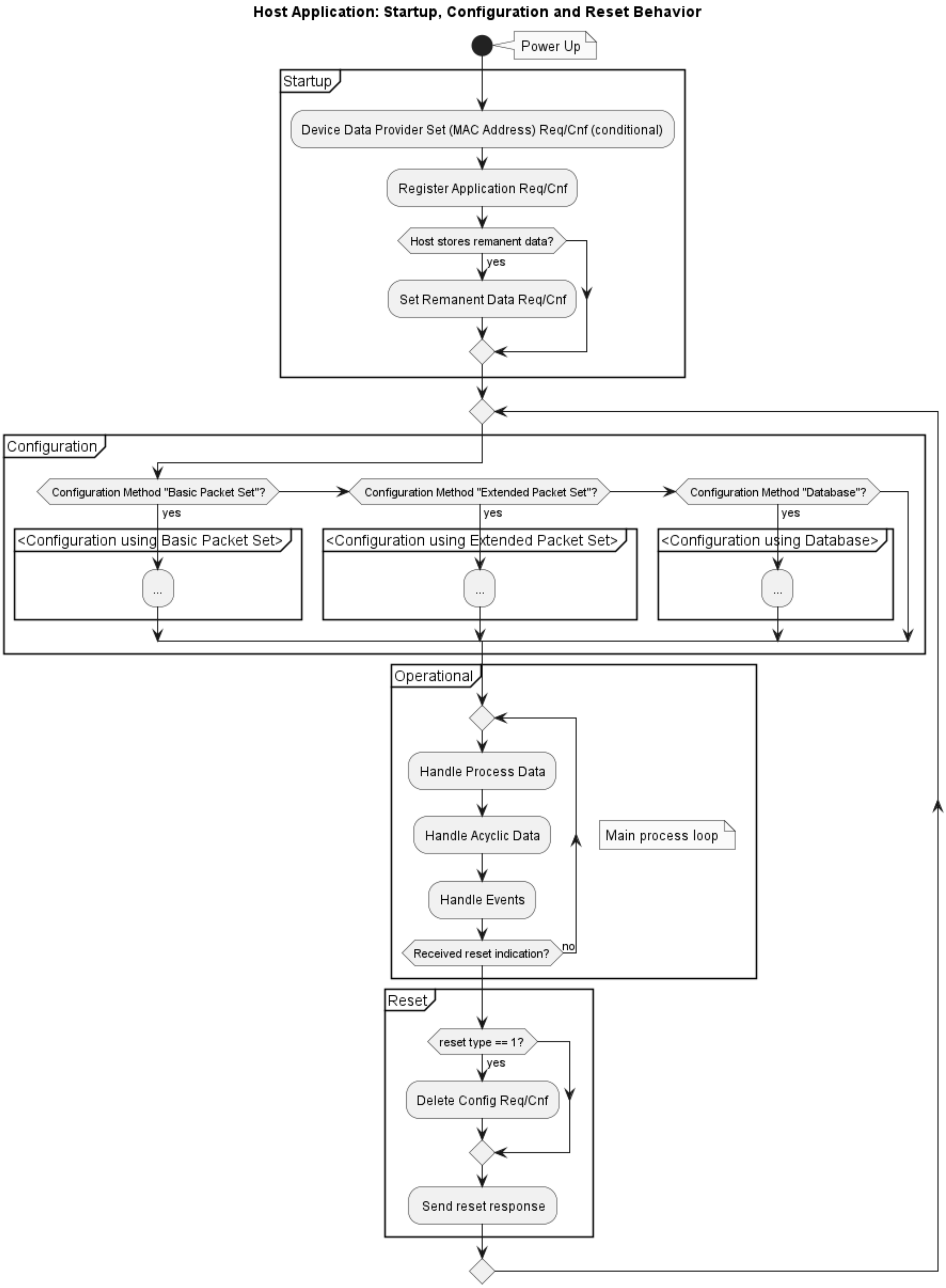

Host Application: Startup, Configuration and Reset Behavior
Power Up
Startup
Device Data Provider Set (MAC Address) Req/Cnf (conditional)
Register Application Req/Cnf
Host stores remanent data?
yes
Set Remanent Data Req/Cnf
Configuration
Configuration Method "Basic Packet Set"?
yes
Configuration Method "Extended Packet Set"?
yes
Configuration Method "Database"?
yes
<Configuration using Basic Packet Set>
...
<Configuration using Extended Packet Set>
...
<Configuration using Database>
...
Operational
Handle Process Data
Handle Acyclic Data
Handle Events
Main process loop
Received reset indication?
no
Reset
reset type == 1?
yes
Delete Config Req/Cnf
Send reset response

Figure 6. Host application: Startup, configuration and reset behavior

3.2.1 Startup

NOTE Setting the MAC address is conditional.
For details, see section Ethernet MAC address.

NOTE Setting remanent data is conditional.
For details, see section Remanent data.

3.2.2 Operational

In the operational state, the host application enters its main process loop which includes I/O data handling, protocol stack event handling.

3.2.3 Configuration

The configuration behavior depends on the chosen method (see section Configuration methods). The sections that explain the specific configuration methods show the different configuration sequences.

For configuration via

- Basic Configuration Packet Set, see section Configuration sequence
- Extended Configuration Packet Set, see section Configuration sequence
- SYCON.net, see section Configuration sequence

3.2.4 Reset

The reset behavior is independent of the chosen configuration method. For more information on “Reset indication” and “Delete Config” handling, see section EIP_OBJECT_RESET_IND and HIL_DELETE_CONFIG_REQ.

3.3 Configuration using the packet API

This section explains the configuration process which uses the Packet API of the EtherNet/IP stack.

Section Hilscher EtherNet/IP stack capabilities describes the default Hilscher CIP Object Model. The configuration of the EtherNet/IP protocol stack will create instances of these CIP object classes and initialize their attribute data according to the provided configuration information.

The stack offers two different configuration sequences based on two sets of packets: The **basic** and the **extended** configuration packet sets. Choose the configuration packet set according to the requirements of your device.

Table Configuration packet sets shows the available configuration packet sets and outlines the capabilities of each of the two configuration variants.

Configuration packet set	Description
Basic	This set provides a basic functionality - Cyclic communication/implicit messaging (transport class1 and Class0). Two assembly instances are available, one for input and one for output data. - Acyclic access (explicit messaging) to all predefined Hilscher CIP objects (unconnected/connected). - Support of DLR protocol. - Support of ACD - Implementation of additional CIP objects, which might be mandatory when a special CIP profile is used. These objects are also accessible via acyclic/explicit messages. A default CIP object model, as illustrated in Default Hilscher Device object model, is established with the basic configuration packet set. If your device needs advanced functionality that the basic configuration set does not cover, use the extended configuration set described below. Limitations when using this configuration packet set: - Max. 2 assembly instances are supported - No configuration assembly instances are supported - CIP Sync is not supported
Extended	Using this configuration packet set, the host application is free to extend the CIP object model of the device in all aspects. In addition to the functionality available the basic configuration packet set, this extended configuration variant: - Allows more assembly instances. The exact number of instances can be found in the data sheet of the firmware. This also includes configuration assembly instances. - Allows optional configuration assemblies (necessary if the device needs configuration parameters from the scanner/originator/PLC before going into cyclic communication). - Supports CIP Sync. For this purpose, the CIP Time Sync object has to be activated (see section EIP_OBJECT_MR_REGISTER_REQ). The extended configuration allows establishing configurations that are a superset of those that can be established the basic configuration packet set.

Table 59. Configuration packet sets

3.3.1 Basic configuration packet set

3.3.1.1 Configuration packets

To configure the EtherNet/IP stack via the basic configuration packet set, the following packets are necessary:

Packet name	Command code (REQ/CNF)
HIL_REGISTER_APP_REQ	0x2F10/0x2F11
EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ	0x3612/0x3613
HIL_SET_REMANENT_DATA_REQ (conditional)	0x2F8C/0x2F8D
HIL_CHANNEL_INIT_REQ	0x2F80/0x2F81

Table 60. Basic configuration packet set - configuration packets

3.3.1.2 Optional request packets

In addition to the request packets required for configuration, the application can optionally issue the following requests during the configuration phase. If your application uses these optional packets, we recommend an application-controlled start as configurable per member `ulSystemFlags` of request packet EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ.

Packet name
EIP_APS_SET_PARAMETER_REQ
EIP_APS_GET_MS_NS_REQ
EIP_OBJECT_MR_REGISTER_REQ
EIP_OBJECT_REGISTER_SERVICE_REQ
EIP_OBJECT_SET_PARAMETER_REQ
EIP_OBJECT_FORCE_LED_STATE_REQ

Table 61. Additional request packets using the basic configuration packet set

3.3.1.3 Indication packets the host application has to handle

The EtherNet/IP protocol stack might generate the following indication packets which the host application has to process and to which the application has to reply with the corresponding response packet:

Packet name	Command code (IND/RES)
EIP_OBJECT_RESET_IND	0x1A24/0x1A25
EIP_OBJECT_CONNECTION_IND	0x1A2E/0x1A2F
EIP_OBJECT_CL3_SERVICE_IND	0x1A3E/0x1A3F
EIP_OBJECT_CIP_OBJECT_CHANGE_IND	0x1AFA/0x1AFB
HIL_STORE_REMANENT_DATA_IND (conditional)	0x2F8E/0x2F8F
HIL_LINK_STATUS_CHANGE_IND	0x2F8A/0x2F8B

Table 62. Indication packets using the basic configuration packet set

3.3.1.4 Configuration sequence

Figure Configuration sequence using the basic configuration packet set below illustrates the configuration packet sequence when using the Basic Configuration Packet Set. For details on how to integrate the configuration sequence into the general behavior of the host application, see section Host application behavior.

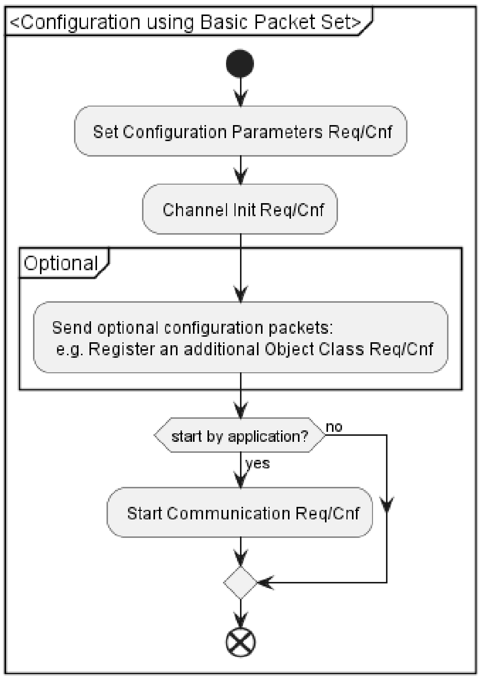

<Configuration using Basic Packet Set>
Set Configuration Parameters Req/Cnf
Channel Init Req/Cnf
Optional
Send optional configuration packets:
e.g. Register an additional Object Class Req/Cnf
start by application?
no
yes
Start Communication Req/Cnf

Figure 7. Configuration sequence using the basic configuration packet set

3.3.2 Extended configuration packet set

3.3.2.1 Configuration packets

To configure the EtherNet/IP stack via the extended configuration packet set, the following packets are necessary:

Packet name	Command code (REQ/CNF)
HIL_REGISTER_APP_REQ	0x2F10/0x2F11
EIP_OBJECT_CIP_SERVICE_REQ	0x1AF8/0x1AF9
EIP_OBJECT_AS_REGISTER_REQ	0x1A0C/0x1A0D
EIP_APS_CONFIG_DONE_REQ	0x3614/0x3615
HIL_SET_REMANENT_DATA_REQ (conditional)	0x2F8C/0x2F8D

Table 63. Extended configuration packet set - configuration packets

3.3.2.2 Optional request packets

In addition to the request packets required for configuration, the application can optionally issue the following requests during the configuration phase:

Packet name	Command code (REQ/CNF)
EIP_APS_SET_PARAMETER_REQ	0x360A/0x360B
EIP_APS_GET_MS_NS_REQ	0x360E/0x360F
EIP_OBJECT_MR_REGISTER_REQ	0x1A02/0x1A03
EIP_OBJECT_REGISTER_SERVICE_REQ	0x1A44/0x1A45
EIP_OBJECT_FORCE_LED_STATE_REQ	0x1A40/0x1A41

Table 64. Additional request packets using the extended configuration packet set

3.3.2.3 Indication packets the host application has to handle

The EtherNet/IP protocol stack might generate the following indication packets toward the host application, which it has to process and reply to with the corresponding response packet:

Packet name	Command code (IND/RES)
EIP_OBJECT_RESET_IND	0x1A24/0x1A25
EIP_OBJECT_CONNECTION_IND	0x1A2E/0x1A2F
EIP_OBJECT_CL3_SERVICE_IND	0x1A3E/0x1A3F
EIP_OBJECT_CIP_OBJECT_CHANGE_IND	0x1AFA/0x1AFB
HIL_STORE_REMANENT_DATA_IND (conditional)	0x2F8E/0x2F8F
HIL_LINK_STATUS_CHANGE_IND	0x2F8A/0x2F8B

Table 65. Indication packets using the extended configuration set

3.3.2.4 Configuration sequence

Figure Configuration sequence using the extended configuration set below illustrates the configuration packet sequence when using the Extended Configuration Packet Set. For details on how to integrate the configuration sequence into the general behavior of the host application, see section Host application behavior.

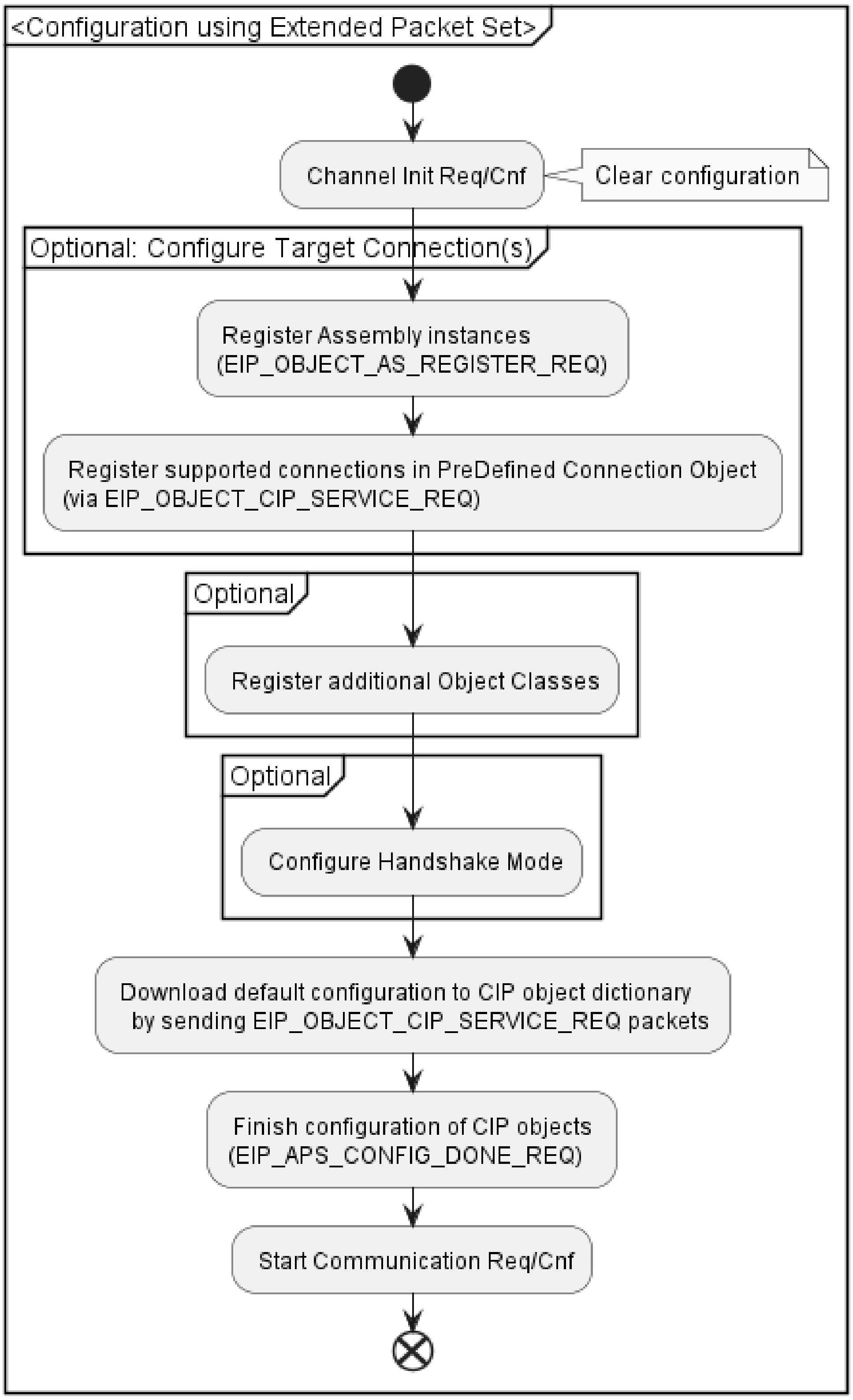

<Configuration using Extended Packet Set>
Channel Init Req/Cnf
Clear configuration
Optional: Configure Target Connection(s)
Register Assembly instances (EIP_OBJECT_AS_REGISTER_REQ)
Register supported connections in PreDefined Connection Object (via EIP_OBJECT_CIP_SERVICE_REQ)
Optional
Register additional Object Classes
Optional
Configure Handshake Mode
Download default configuration to CIP object dictionary by sending EIP_OBJECT_CIP_SERVICE_REQ packets
Finish configuration of CIP objects (EIP_APS_CONFIG_DONE_REQ)
Start Communication Req/Cnf

Figure 8. Configuration sequence using the extended configuration set

3.4 Configuraion using Sycon.net

3.4.1 Configuration sequence

Figure Configuration sequence using the database below illustrates the configuration packet sequence when using a database. For details on how to integrate the configuration sequence into the general behavior of the host application, see section Host application behavior.

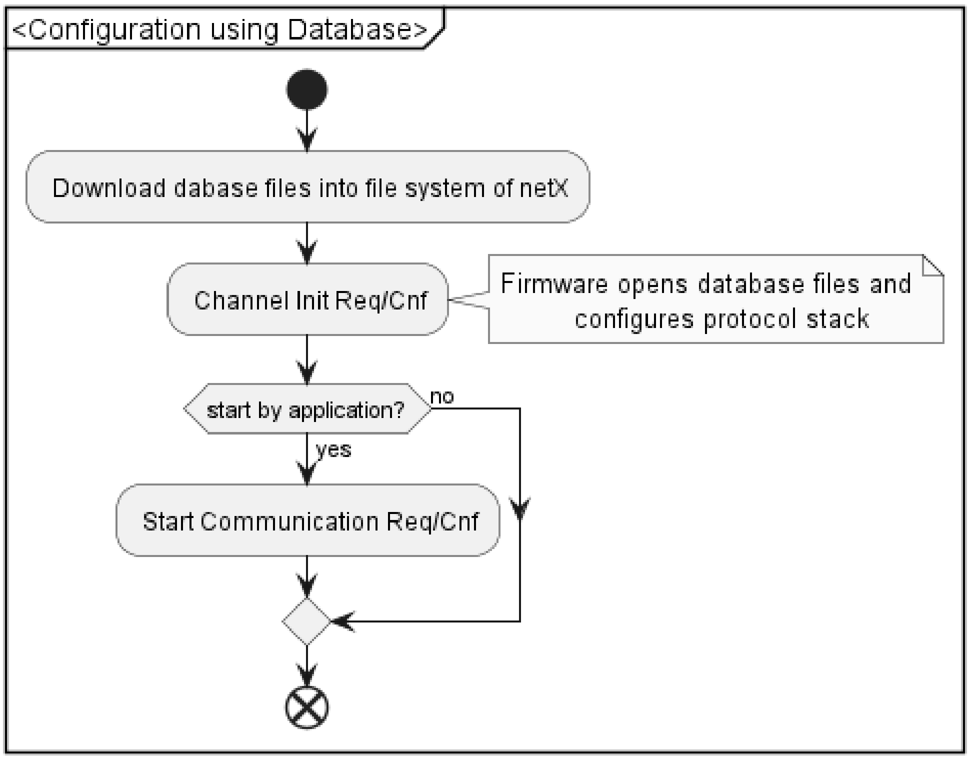

<Configuration using Database>
Download dabase files into file system of netX
Channel Init Req/Cnf
Firmware opens database files and configures protocol stack
start by application?
no
yes
Start Communication Req/Cnf

Figure 9. Configuration sequence using the database

3.5 Remanent data

3.5.1 Remanent data purpose

In an EtherNet/IP device, it is common that attribute values of implemented CIP objects change due to services issued towards the device via the network. These attributes may contribute to the configuration of the device and thus, this *runtime configuration* may change on the fly.

CIP object attributes are:

- **volatile** (their value will not be retained via power cycles) or
- **non-volatile** (the attribute values are persistently stored in the device)

The set of persistently stored attributes of the EtherNet/IP protocol stack is called remanent data. The protocol stack automatically updates the remanent data whenever a non-volatile attribute changes.

Modifying non-volatile attributes via network or host interface causes a Flash write access. When the device is reconfigured, the previously stored state of these attributes will be applied in addition to the configuration of the device.

The CIP Identity object offers a service that allows a reset of the device configuration to the *factory default configuration*. Basically, this is implemented by deleting the remanent data before the reset. To delete remanent data, use the command HIL_DELETE_CONFIG_REQ.

3.5.2 Remanent data responsibility

When you design your application, you have to decide who stores the remanent data, the **protocol stack** or the **application.**

If the system designer decides that the application stores the remanent data, the taglist of the firmware must be modified as described in section Resource and feature configuration via tag list.

NOTE The Hilscher EtherNet/IP stack is capable of handling remanent data for the built-in CIP objects only. If the host application implements further CIP objects with non-volatile attributes, they will have to be handled completely within the scope of the host application. Since stack mechanisms do not support the latter case, this is not subject of this manual.

Remanent data is stored by	Description
Protocol stack	The stack stores the remanent data **Requirements** The protocol stack requires access to non-volatile memory. **Firmware configuration** In the tag list “Remanent Data Responsibility” the tag “Remanent Data stored by Host” has to be set to disabled in the firmware file (`.nxf` or `.nxi`). This is the default setting in a firmware.

Remanent data is stored by	Description
Application	The application stores the remanent data If the host application stores remanent data, the protocol stack no longer accesses the Flash memory, but provides the complete remanent data block towards the host application per indication. The host application has to store the provided data with each indication and has to set this data back to the stack in the (re)configuration process. **Requirement** The application has to use the *Channel Component Information* service (GENAP_GET_COMPONENT_IDS_REQ) to get the information on the required size for remanent data of each protocol stack component. The application has to use the *Set Remanent Data* service [sec-appintf-set-remanent-data> and to support the *Store Remanent Data* service <<sec-appintf-store-remanent-data]. **Firmware configuration** In the tag list “Remanent Data Responsibility” the tag “Remanent Data stored by Host” has to be set to enabled in the firmware file (`.nxf or .nxi).` `*Configuration` The application has to use the *Set Remanent Data* service HIL_SET_REMANENT_DATA_REQ to provide the remanent data to each protocol stack component any time the host application starts up for the first time (e.g. after power-on) and before the application sends the EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ. For a state diagram, see section Host application behavior. **During runtime** The stack component indicates to the application the *Store Remanent Data* service HIL_STORE_REMANENT_DATA_IND each time remanent data has been changed. The stack component provides the remanent data as a block to the application. The application has to store the remanent data with each indication. **Note** For a detailed description of the *Channel Component Information* service, the *Set Remanent Data* service, and the *Store Remanent Data* service, see reference [9].

Table 66. Protocol stack or host application stores remanent data

3.5.3 Remanent data state

The remanent data is either available/undeleted or unavailable/deleted. This state is not explicitly observable, but maintained by the protocol stack. This state is stored in the remanent data BLOB itself. If there is no such valid BLOB, the remanent data counts as unavailable/deleted. Figure Remanent data state transitions illustrates these two states and the events that trigger transitions between these states.

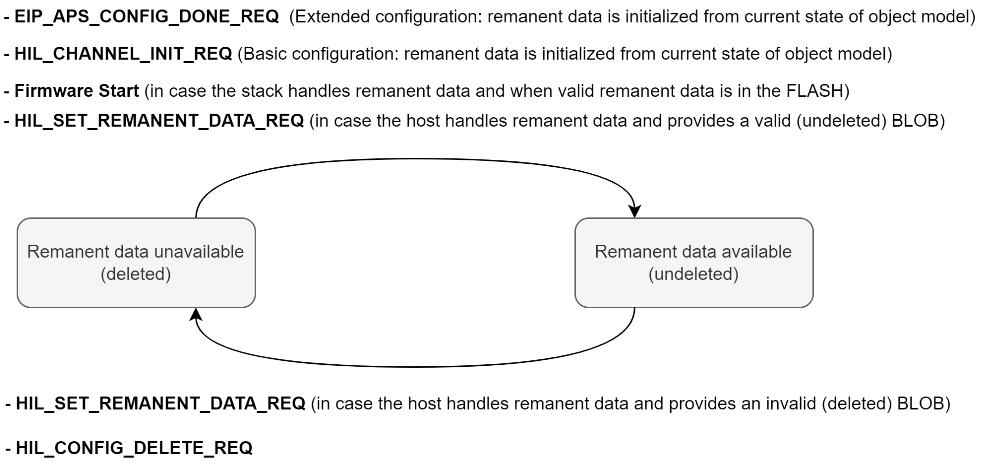

- **EIP_APS_CONFIG_DONE_REQ** (Extended configuration: remanent data is initialized from current state of object model)
- **HIL_CHANNEL_INIT_REQ** (Basic configuration: remanent data is initialized from current state of object model)
- **Firmware Start** (in case the stack handles remanent data and when valid remanent data is in the FLASH)
- **HIL_SET_REMANENT_DATA_REQ** (in case the host handles remanent data and provides a valid (undeleted) BLOB)
Remanent data unavailable (deleted)
Remanent data available (undeleted)
- **HIL_SET_REMANENT_DATA_REQ** (in case the host handles remanent data and provides an invalid (deleted) BLOB)
- **HIL_CONFIG_DELETE_REQ**

Figure 10. Remanent data state transitions

3.5.4 Remanent data flow

When the basic configuration packet set is used, the protocol stack is initialized primarily with the packets EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ and HIL_CHANNEL_INIT_REQ. This leads to a data flow of configuration data and remanent data as illustrated in Figure Remanent data flow with basic configuration packets.

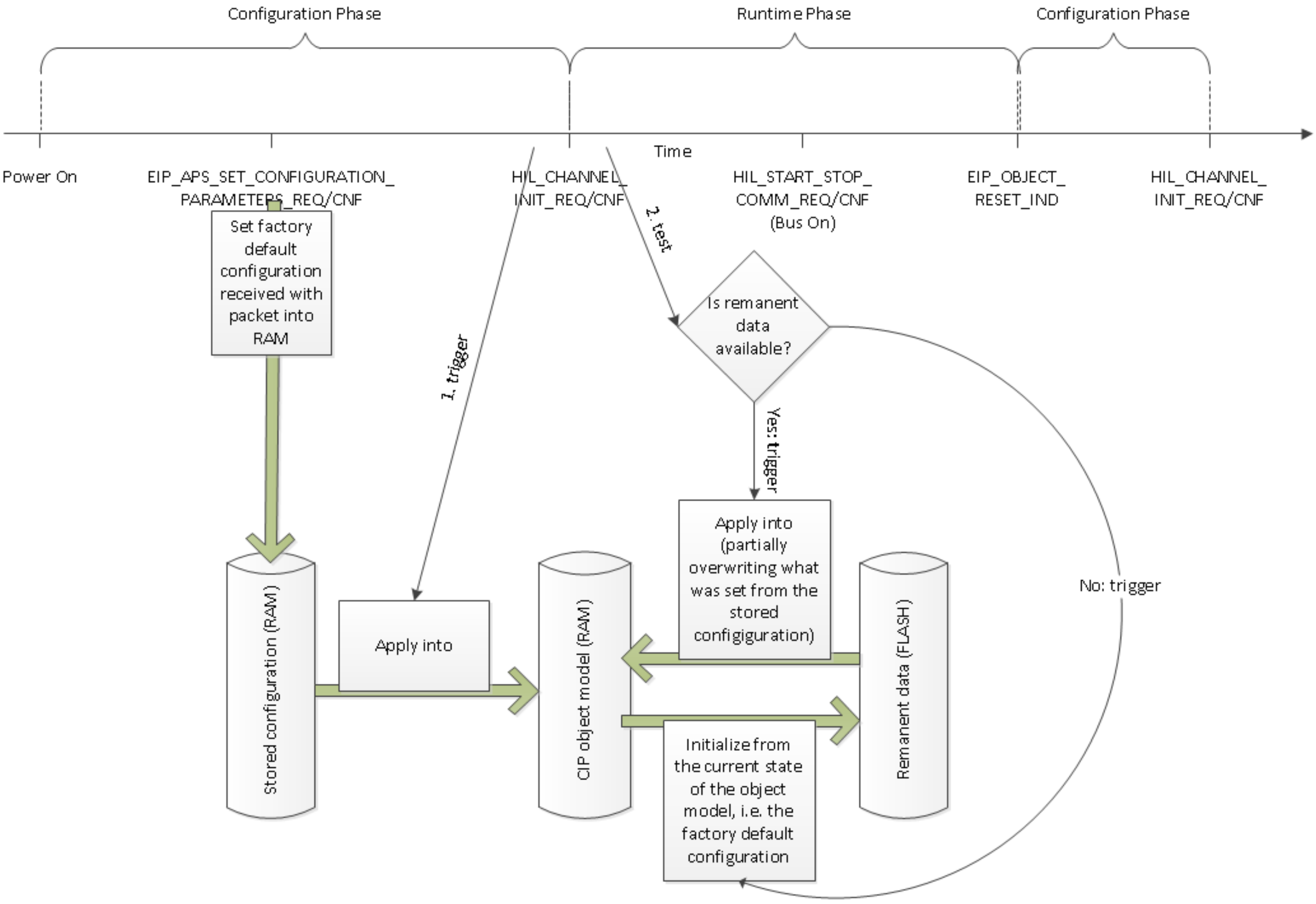

Configuration Phase
Runtime Phase
Configuration Phase
Time
Power On
EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ/CNF
HIL_CHANNEL_INIT_REQ/CNF
HIL_START_STOP_COMM_REQ/CNF (Bus On)
EIP_OBJECT_RESET_IND
HIL_CHANNEL_INIT_REQ/CNF
Set factory default configuration received with packet into RAM
1. trigger
2. test
Is remanent data available?
Yes: trigger
No: trigger
Apply into (partially overwriting what was set from the stored configiguration)
Stored configuration (RAM)
Apply into
CIP object model (RAM)
Remanent data (FLASH)
Initialize from the current state of the object model, i.e. the factory default configuration

Figure 11. Remanent data flow with basic configuration packets

With the extended configuration packet, EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ is not used and thus the stored configuration is pointless in this scenario. Instead, the factory default configuration is set via EIP_OBJECT_CIP_SERVICE_REQ, Set_Attribute_Single directly into the object model. The EIP_APS_CONFIG_DONE_REQ is used instead of the HIL_CHANNEL_INIT_REQ to trigger the application of remanent data in addition to these factory defaults.

The HIL_CHANNEL_INIT_REQ has a different purpose if no stored configuration is available: It resets the protocol stack to its initial state so that a fresh (factory default) configuration can be set as illustrated in Figure Remanent data flow with extended configuration packets.

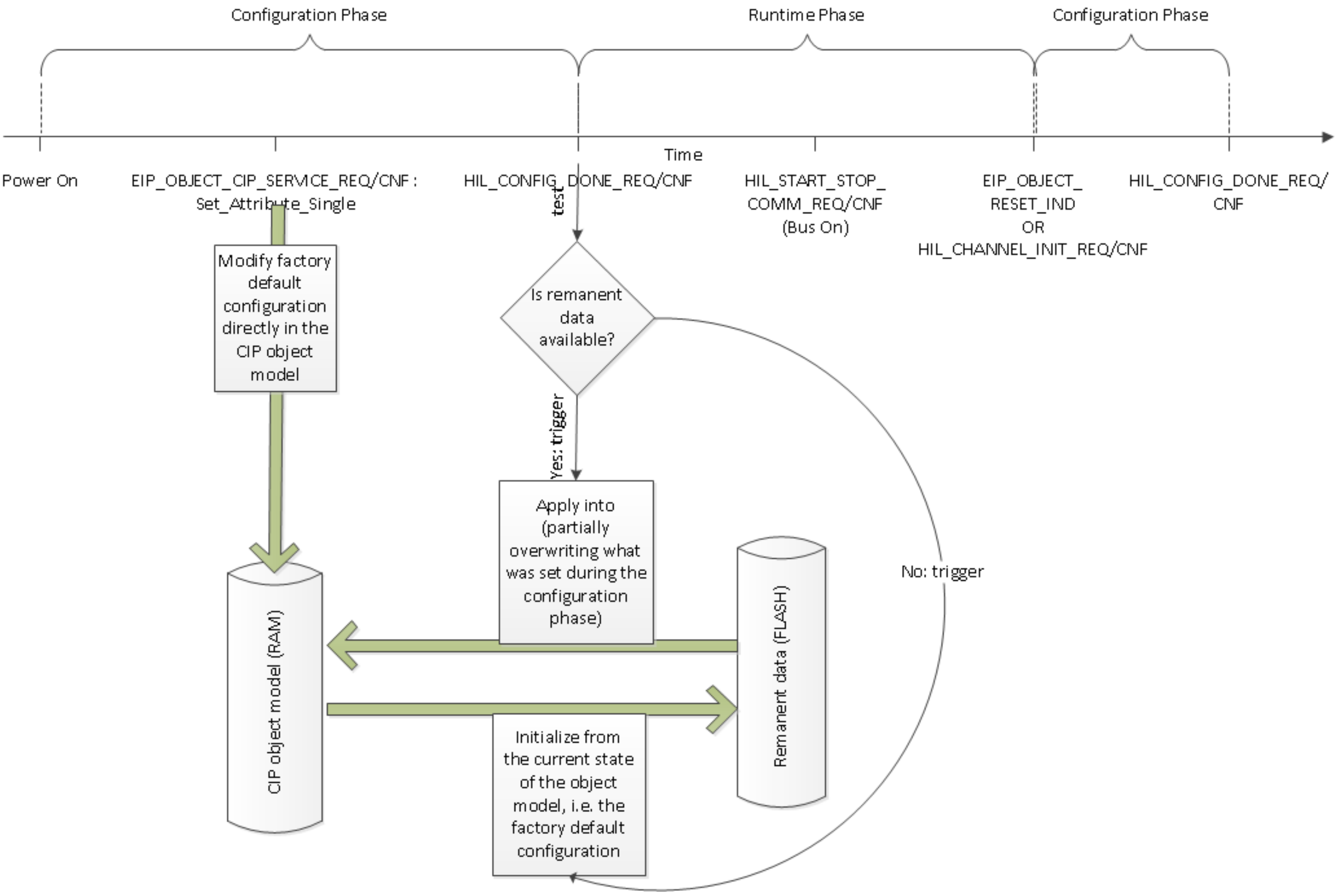

Configuration Phase
Runtime Phase
Configuration Phase
Time
Power On
EIP_OBJECT_CIP_SERVICE_REQ/CNF : Set_Attribute_Single
HIL_CONFIG_DONE_REQ/CNF
HIL_START_STOP_COMM_REQ/CNF (Bus On)
EIP_OBJECT_RESET_IND OR HIL_CHANNEL_INIT_REQ/CNF
HIL_CONFIG_DONE_REQ/CNF
test
Modify factory default configuration directly in the CIP object model
Is remanent data available?
Yes: trigger
No: trigger
Apply into (partially overwriting what was set during the configuration phase)
CIP object model (RAM)
Remanent data (FLASH)
Initialize from the current state of the object model, i.e. the factory default configuration

Figure 12. Remanent data flow with extended configuration packets

What these figures do not show, but what needs to be emphasized for both configuration package sets, is that there is a difference in behavior of the EIP_OBJECT_CIP_SERVICE_REQ: Set_Attribute_Single depending on whether the protocol stack is in the configuration phase or runtime phase.

Phase	Behavior of EIP_OBJECT_CIP_SERVICE_REQ: Set_Attribute_Single
Configuration phase	The attribute settings are manifested in the object model only. Thus, the attribute settings alter the factory default configuration. With the basic configuration packet set, the attributes may be overwritten later with the stored configuration. With both configuration packet sets, the attributes may be overwritten later with the remanent data.
Runtime phase	The attribute settings are manifested in the object model and in the remanent data. Thus, the attribute settings alter the device runtime configuration.

3.5.5 Remanent data content

All implemented attributes, which are non-volatile according to the CIP specification, and which can dynamically change their value, contribute to the remanent data.

Object	Attribute
Identity (0x1)	Heartbeat Interval (10)
TimeSync (0x43)	PTP Enable (1)
	Port Enable Config (13)
	Port Log Announce Interval Config (14)
	Port Log Sync Interval Config (15)
	Priority1 (16)
	Priority2 (17)
	Domain Number (18)
	Sync Parameters (768)
Quality of Service (0x48)	802.1Q Tag Enable (1)
	DSCP PTP Event (2)
	DSCP PTP General (3)
	DSCP Urgent (4)
	DSCP Scheduled (5)
	DSCP High (6)
	DSCP Low (7)
	DSCP Explicit (8)
TCP/IP Interface (0xF5)	Configuration Control (3)
	Interface Configuration (5)
	Host Name (6)
	TTL Value (8)
	Multicast Configuration (9)
	Select ACD (10)
	Last Conflict Detected (11)
	Quick Connect Enable (12)
	Encapsulation Inactivity Timeout (13)
	DHCP Client Identifier (768)
	DHCP Discover Transmission Rate (769)
EtherNet Link (0xF6)	Interface Control (6)
	Admin State (9)
	MDI Configuration (768)
LLDP Management (0x109)	LLDP Enable (1)
	MsgTxInterval (2)
	MsgTxHold (3)

Table 67. Remanently stored CIP attributes

NOTE This table is for information only. If the host application stores the remanent data as configurable per tag list, the remanent data is provided as an opaque BLOB to the host application. The host application will occasionally provide this data back without modification. It is not intended that the host application modifies the contents of the remanent data block directly. The above table does not describe the internal structure of the BLOB.

3.6 Bus State

3.6.1 Purpose

The `BusOn Signal` is a netX COS Flag, so BusOn is a command on the one hand and an internal state of the firmware on the other.

The host application can set and clear the BusOn signal, and the Firmware will transit to the corresponding state `Bus On` or `Bus Off` in a manner that is specific to the implemented protocol family and the firmware version.

This section specifies the semantics of the BusOn and BusOff states for this EtherNet/IP-Firmware.

> **NOTE** The BusOn Signal may also be modified by the host application through the packet API with HIL_START_STOP_COMM_REQ

3.6.2 BusOn and BusOff States

Basically, BusOn and BusOff Signals/States affect the `ulCommunicationState` field in the DPM Common Status Block. See section Status information for a description of the possible communication states. When successfully configured, BusOn causes the system to transit from communication state `HIL_COMM_STATE_STOP` to `HIL_COMM_STATE_IDLE`. Once a connection has been established, it will further transit to `HIL_COMM_STATE_OPERATE`. It will go back to `HIL_COMM_STATE_STOP` on BusOff.

The first BusOn signal will establish the physical (MAC) and logical (IP) link. In a subsequent BusOff, the physical and logical links will remain active, unless forced to restart (see: TCP/IP Interface Object - Common services). Instead, the protocol stack will drop all CIP connections and ignore all UDP traffic and reject all new TCP connections during such an intermediate BusOff state. Other subsystems at the network remain still active, e.g. the Webserver or the Ethernet Device Configuration Service.

When using the Basic Configuration Packets, depending on configuration parameter `ulSystemFlags` in the SetConfig packet, the BusOn Signal may be set automatically (during HIL_CHANNEL_INIT_REQ). Optionally, the user can select *application controlled start* to suppress automatic transition to BusOn State. Then, the application has to control this programmatically. The same applies for a database configuration which also contains a `ulSystemFlags` attribute. With the Extended Packet Configuration, the BusOn control mode is always application controlled.

During BusOff, no CIP-based network communication from/to the device takes place. Therefore, the CIP object dictionary is not subject to any external changes and will remain in a consistent state. All indication packets that are generated due to external CIP requests will thus not be generated during BusOff, e.g. the Object Change Indication and the Class 3 Service Indication. The system will also cause less load on the netX device’s processor, memory and the network.

3.6.3 BusOn and Producing Assembly Run Status

When the system is properly configured and the BusOn signal is set and eventually the system is in mode OPERATE, all producing assemblies will transit from the IDLE to the RUN state. If the Assembly is used as a producing connection endpoint and that connection has a RUN/IDLE header, the Assembly’s RUN status will be signaled over the connection as well. This situation regularly occurs in the EtherNet/IP Scanner and may also occur in the adapter when RUN/IDLE Information is transmitted in T2O direction, though that is a rather uncommon use case.

In case the RUN/IDLE status is explicitly controlled by the host application (feature MAP_RUNIDLE), the producing Assembly’s RUN/IDLE status will be derived from the application-provided value in the DPM.

The RUN/IDLE Status of consuming Assemblies is not directly affected by the BusOn state, but will eventually be IDLE as well since all connections should be dropped due to the BusOff. Once BusOn is set again, and a connection is reestablished, the RUN/IDLE status from the received I/O frames is reflected. If the connection is modeless, the RUN status will be set with the first valid I/O frame received.

Chapter 4 Application interface

This section defines the application interface of the EtherNet/IP Adapter.

4.1 Packet usage

This section contains general information on using the Packet API, as well as essential details for application developers when programming the API.

4.1.1 Value range

If a value range is defined for a parameter within the packet, the application is obligated to comply with it. Parameters outside the specified range can lead to unpredictable and potentially harmful consequences, such as undefined firmware behavior or error codes being returned. To prevent such issues, it is important for developers to carefully adhere to value ranges for all parameters within packets.

4.2 Configuring the EtherNet/IP Adapter

This section explains the packets used to configure the EtherNet/IP Adapter with the DPM/packet interface. For details on the configuration sequence, see section Configuration using the packet API.

The following packets are available for the configuration:

Packet	Command code (REQ)
EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ	0x00003612
EIP_APS_SET_PARAMETER_REQ	0x0000360A
EIP_APS_CONFIG_DONE_REQ	0x00003614
EIP_OBJECT_MR_REGISTER_REQ	0x00001A02
EIP_OBJECT_AS_REGISTER_REQ	0x00001A0C
EIP_OBJECT_REGISTER_SERVICE_REQ	0x00001A44
EIP_OBJECT_SET_PARAMETER_REQ	0x00001AF2
EIP_OBJECT_CIP_SERVICE_REQ	0x00001AF8
HIL_SET_WATCHDOG_TIME_REQ	0x00002F04
HIL_REGISTER_APP_REQ	0x00002F10
HIL_START_STOP_COMM_REQ	0x00002F30
HIL_CHANNEL_INIT_REQ	0x00002F80

Table 68. Overview: Configuration packets of the EtherNet/IP Adapter

4.2.1 Set Configuration Parameters service

The host application uses this service to configure the device with configuration parameters. This packet is part of the basic configuration set and provides a basic configuration of all built-in CIP objects.

Using this configuration method, the stack automatically creates two assembly instances serving as connection endpoints for implicit/cyclic data exchange. The I/O data of these instances will start at offset 0 in the DPM (relative offset to the base addresses of the input and output areas of the DPM).

NOTE If the application sets the revision information or the product name to zero or to an empty string, the protocol stack will apply default (Hilscher-specific) values.

NOTE If the application sets the serial number to zero, the protocol stack will apply the device-specific information from the Security Memory or FDL, if available.

In case of EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ, the EtherNet/IP Adapter protocol stack will

- test the “configuration locked” condition and reject the request thereafter, see section HIL_LOCK_UNLOCK_CONFIG_REQ
- perform consistency and integrity checks on the received configuration and reject the configuration in case of errors
- in case of success, the buffered configuration will be applied with the next channel initialization (HIL_CHANNEL_INIT_REQ).

This request does not register the application with the stack. The host application has to register itself by means of packet HIL_REGISTER_APP_REQ as described in the netX DPM Manual to receive indication packets from the netX (see section HIL_REGISTER_APP_REQ).

NOTE Parameter set V3 (and newer) is supported only. The stack will reject any older parameter versions or packet lengths.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t		Packet data length in bytes EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ_SIZE + EIP_APS_CONFIGURATION_PARAMETER_SET_V3_SIZE
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x3612	EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ
`tData (EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ_T)`			
`ulParameterVersion`	uint32_t	3 (latest version)	Version of the following parameter structure
`unConfig.tV3`	union		See Table EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T – Configuration Parameter Set V3

Table 69. EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ_T – Set Configuration Parameters request

Structure EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T			
`ulSystemFlags`	uint32_t (Bit field)	0, 1	System flags area The device start can be automatic or application-controlled: Automatic (0): Network connections are opened automatically regardless of the state of the host application. After a device start, the communication with a controller is allowed without the flag `BUS_ON`, but will be interrupted if the flag `BUS_ON` changes state to 0. Application-controlled (1): The channel firmware has to wait until the host application sets the Application Ready flag in the communication change of the state register. Communication with controller is allowed only with the flag `BUS_ON`. For details on this topic, see reference [1].
`ulWdgTime`	uint32_t	0, 20..65535	Watchdog time (in milliseconds). 0 = Watchdog timer has been switched off Default value: 1000
`ulInputLen`	uint32_t	0..504 Default: 16	Length of Input data (O→T direction, data the device receives from a Scanner/PLC)
`ulOutputLen`	uint32_t	0..504 Default: 16	Length of Output data (T→O direction, data the device sends to a Scanner/PLC)
`ulTcpFlag`	uint32_t	Default value: `0x00000410`	The TCP flags configure the TCP stack behavior related the IP Address assignment (STATIC, BOOTP, DHCP) and the Ethernet port settings (such as Auto-Neg, 100/10MBits, Full/Half Duplex). For more information, see Table Available TCP flags in bit field ulTcpFlag of the Basic Configuration Packet. Recommended default value: `0x00000410` (DHCP active and both ports set to Auto-Negotiation) **Note**: For a valid configuration, one of the following bits must be set: 0: STATIC IP 3: BOOTP 4: DHCP If no bit is set, the firmware will use the static IP address 192.168.210.10 as a default.
`ulIPAddr`	uint32_t	All valid IP-addresses Default: 0.0.0.0	IP Address See detailed explanation in the corresponding TCP/IP Manual (reference [2])
`ulNetMask`	uint32_t	All valid masks Default: 0.0.0.0	Network Mask See detailed explanation in the corresponding TCP/IP Manual (reference [2])
`ulGateway`	uint32_t	All valid IP-addresses Default: 0.0.0.0	Gateway Address See detailed explanation in the corresponding TCP/IP Manual (reference [2])

Structure EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T			
`usVendorID`	uint16_t	0..65535	Vendor identification: This is an identification number for the manufacturer of an EtherNet/IP device. Vendor IDs are managed by ODVA (see www.odva.org). The host application is responsible for setting a nonzero value as defined by the CIP specification. The protocol stack does not restrict the value. Default value: 283 (Hilscher)
`usProductType`	uint16_t	0..65535	CIP Device Type (former "Product Type") The list of device types is managed by ODVA (see www.odva.org). It is used to identify the device profile that a particular product is using. Device profiles define minimum requirements a device must implement as well as common options. Publicly defined: 0x00 - 0x63 Vendor-specific: 0x64 - 0xC7 Publicly defined: 0xC8 Reserved by CIP: 0xC9 - 0xFF Publicly defined: 0x100 - 0x2FF Vendor-specific: 0x300 - 0x4FF Reserved by CIP: 0x500 - 0xFFFF Default: 0x0C (Communications Adapter)
`usProductCode`	uint16_t	1..65535	Product code The vendor assigned Product Code identifies a particular product within a device type. Each vendor assigns this code to each of its products. The Product Code typically maps to one or more catalog/model numbers. Products shall have different codes if their configuration and/or runtime options are different. Such devices present a different logical view to the network. On the other hand, for example, two products that are the same except for their color or mounting feet are the same logically and may share the same product code. The value zero is not valid.
`ulSerialNumber`	uint32_t	0	Deprecated. This value has to be set to zero. The firmware will apply the serial number as stored in the Device Data Provider (DDP), which in turn fetches it from either the SecMem or FDL data sources. Refer to section Device serial number for details.
`bMinorRev`	uint8_t	1..255	Minor revision
`bMajorRev`	uint8_t	1..127	Major revision
`abDeviceName[32]`	uint8_t		Device Name This text string should represent a short description of the product/product family represented by the product code. The same product code may have a variety of product name strings. Byte 0 indicates the length of the name. Bytes 1 -30 contain the characters of the device name) Example: "Test Name" `abDeviceName[0]` = 9 `abDeviceName[1..9]` = "Test Name" See [12] for information about restrictions regarding product naming.

Structure EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T

`ulInputAssInstance`	uint32_t	1...0xFFFFFFFE Default: 100	Instance number of input assembly (O→T direction) See Table Assembly instance number ranges. **Note:** The value of `ulInputAssInstance` must differ from the value of `ulOutputAssInstance`. **Note:** The host application is responsible to choose an assembly ID from a proper range as defined in CIP Vol1, table “Assembly instance ID ranges”.
`ulInputAssFlags`	uint32_t	Bit mask	Input assembly (O→T) flags See Table Assembly Types and Option Flags for a description of available Assembly flags. The flag `EIP_AS_TYPE_INPUT` must be set at minimum.
`ulOutputAssInstance`	uint32_t	1 ... 0xFFFFFFFE Default: 101	Instance number of output assembly (T→O direction) See Table Assembly instance number ranges. **Note:** The value of `ulInputAssInstance` must differ from the value of `ulOutputAssInstance`. **Note:** The host application is responsible to choose an assembly ID from a proper range as defined in CIP Vol1, table “Assembly instance ID ranges”.
`ulOutputAssFlags`	uint32_t	Bit mask	Output assembly (T→O) flags See Table Assembly Types and Option Flags for a description of available Assembly flags.
`tQoS_Config`	EIP_DPMINTF_QOS_CONFIG_T	See Table Quality of Service Structure Description (struct EIP_DPMINTF_QOS_CONFIG_T)	Quality of Service configuration This parameter set configures the Quality of Service Object (CIP ID 0x48)
`ulNameServer`	uint32_t		Name Server 1 This parameter configures the NameServer element of attribute 5 of the TCP/IP Interface Object. See section TCP/IP Interface Object (class code: 0xF5) for more information. Default: 0.0.0.0
`ulNameServer_2`	uint32_t		Name Server 2 This parameter configures the NameServer2 element of attribute 5 of the TCP/IP Interface Object. See section TCP/IP Interface Object (class code: 0xF5) for more information. Default: 0.0.0.0
`abDomainName[48 + 2]`	uint8_t		Domain Name This parameter configures the DomainName element of attribute 5 of the TCP/IP Interface Object. See section TCP/IP Interface Object (class code: 0xF5) for more information.
`abHostName[64+2]`	uint8_t		Host Name This parameter configures attribute 6 of the TCP/IP Interface Object. See section TCP/IP Interface Object (class code: 0xF5) for more information.
`bSelectAcd`	uint8_t		Select ACD This parameter configures attribute 10 of the TCP/IP Interface Object. The valid range of values is [0..255], where a value of zero maps to value zero of the corresponding attribute and all values different from zero map to value 1 of the corresponding attribute. See section TCP/IP Interface Object (class code: 0xF5) for more information.

Structure EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T			
tLastConflictDetected	EIP_DPMINTF_TI_ACD_LAST_CONFLICT_T		Last Detected Conflict This parameter configures attribute 11 of the TCP/IP Interface Object. See Table Acd Last Conflict Structure Description (struct EIP_DPMINTF_TI_ACD_LAST_CONFLICT_T) and section TCP/IP Interface Object (class code: 0xF5) for more information.
bQuickConnectFlags	uint8_t	0,1,3 Default: All zero	Quick Connect Flags This parameter enables/ disables the Quick Connect functionality within the stack. This affects the TCP Interface Object (0xF5) attribute 12. See section TCP/IP Interface Object (class code: 0xF5) for more information. Bit 0 (`EIP_OBJECT_QC_FLAGS_ACTIVATE_ATTRIBUTE`): If set (1), the Quick Connect Attribute 12 of the TCP Interface Object (0xF5) is activated (i.e. it is present and accessible via CIP services). You can configure the actual value of Quick Connect Attribute 12 using bit 1. Bit 1 (`EIP_OBJECT_QC_FLAGS_ENABLE_QC`): This bit configures the current value of attribute 12. If set, attribute 12 has the value 1 (Quick Connect enabled). If not set, Quick connect is disabled. This bit will be evaluated only if bit 0 is set (1).
abAdminState[2]	uint8_t	1, 2	Admin State This parameter configures attribute 9 of the Ethernet Link Object. Default: Both entries 0x01 (enabled) See section Ethernet Link Object (class code: 0xF6) for more information.
bTTLValue	uint8_t	1-255 Default: 1	This parameter corresponds to attribute 8 of the TCP/IP Interface Object (CIP Id 0xF5). The TTL value attribute shall use for the IP header Time-to-Live when sending EtherNet/IP packets via multicast. This attribute shall be stored in non-volatile memory.
tMCastConfig	EIP_DPMINTF_TI_MCAST_CONFIG_T		This parameter corresponds to attribute 9 of the TCP/IP Interface Object (CIP Id 0xF5). The MCast Config set the used multicast range for multicast connections. This attribute shall be stored in nonvolatile memory. See Table Multicast Configuration Structure Description (struct EIP_DPMINTF_TI_MCAST_CONFIG_T) and section TCP/IP Interface Object (class code: 0xF5) for more information.
usEncapInactivityTimer	uint16_t	0-3600 Default: 120 seconds	This parameter corresponds to attribute 13 of the TCP/IP Interface Object (CIP Id 0xF5). The Encapsulation Inactivity Timeout closes the sockets when the defined time (specified in seconds) elapsed without Encapsulation activity. This attribute shall be stored in non-volatile memory.

Table 70. EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T – Configuration Parameter Set V3

The bits of the `ulTcpFlag` member have the following semantics:

Bits	Description
31 ... 29	Reserved for future use, set to zero
28	Speed Selection (Ethernet Port 2): Only evaluated if bit 15 is set. Behaves the same as bit 12.

Bits	Description
27	Duplex Operation (Ethernet Port 2): Only evaluated if bit 15 is set. Behaves the same as bit 11.
26	Auto-Negotiation (Ethernet Port 2): Only evaluated if bit 15 is set. Behaves the same as bit 10.
25 ... 16	Reserved for future use, set to zero
15	Extended Flag: Use this flag, if the device has two Ethernet ports in case you intend to configure the two ports separately regarding "Speed Selection", "Duplex Operation" or "Auto-Negotiation". If not set (0), configure both ports with the same parameters using the bits 10 to 12. If set (1), configure port 1 using bits 10 to 12. Configure Port 2 using the bits 26 to 28.
13 .. 14	Reserved for future use, set to zero
12	Speed Selection: (Ethernet Port 1) If set (1), the device will operate at 100 MBit/s, otherwise at 10 MBit/s. The stack will evaluate this parameter only, if auto-negotiation (bit 10) is not set (0).
11	Duplex Operation: (Ethernet Port 1) If set (1), full-duplex operation will be enabled, otherwise the device will operate in half duplex mode The stack will evaluate this parameter only, if auto-negotiation (bit 10) is not set (0).
10	Auto-Negotiation: (Ethernet Port 1) If set (1), the device will negotiate speed and duplex with connected link partner. If set (1), this flag overwrites Bit 11 and Bit 12 .
9 ... 5	Reserved for future use, set to zero
4	Enable DHCP: If set (1), the device tries to obtain its IP configuration from a DHCP server.
3	Enable BOOTP: If set (1), the device tries to obtain its IP configuration from a BOOTP server.
2	Gateway available: If set (1), the stack will evaluate the content of the `ulGateway` parameter. If the flag is not set (0), you must set `ulGateway` to `0.0.0.0`.
1	Netmask available: If set (1), the stack will evaluate the content of the `ulNetMask` parameter. If the flag is not set the device will assume to be an isolated host which is not connected to any network. The `ulGateway` parameter will be ignored in this case.
0	IP address available: If set (1), the stack will evaluate the content of the `ulIpAddr` parameter. In this case, the parameter `ulNetMask` must contain a valid net mask.

Table 71. Available TCP flags in bit field ulTcpFlag of the Basic Configuration Packet

Variable	Type	Value/Range	Description
`ulQoSFlags`	uint32_t	0	Deprecated, set to 0.
`bTag802Enable`	uint8_t	0-1	Enables or disables sending 802.1Q frames on CIP messages. 0: 802.1Q is disabled (default) 1: 802.1Q is enabled **Note**: the EtherNet/IP stack does currently not support attribute 1 of the QoS object. This field only serves as a placeholder for future implementations.
`bDSCP_PTP_Event`	uint8_t	0-63 Default: 59	DSCP value for PTP (IEEE 1588) event messages. Relates to QoS Attribute 2
`bDSCP_PTP_General`	uint8_t	0-63 Default: 47	DSCP value for PTP (IEEE 1588) general messages. Relates to QoS Attribute 3
`bDSCP_Urgent`	uint8_t	0-63 Default: 55	DSCP value for CIP transport class 0/1 Urgent priority messages. Relates to QoS Attribute 4

Variable	Type	Value/Range	Description
bDSCP_Scheduled	uint8_t	0-63 Default: 47	DSCP value for CIP transport class 0/1 Scheduled priority messages. Relates to QoS Attribute 5
bDSCP_High	uint8_t	0-63 Default: 43	DSCP value for CIP transport class 0/1 High priority messages. Relates to QoS Attribute 6
bDSCP_Low	uint8_t	0-63 Default: 31	DSCP value for CIP transport class 0/1 low priority messages. Relates to QoS Attribute 7
bDSCP_Explicit	uint8_t	0-63 Default: 27	DSCP value for CIP explicit messages (messages with transport class 3 and UCMM messages). Relates to QoS Attribute 8

Table 72. Quality of Service Structure Description (struct EIP_DPMINTF_QOS_CONFIG_T)

Variable	Type	Value/Range	Description
bAcdActivity	uint8_t	Default: 0	State of ACD activity when last conflict detected
abRemoteMac[6]	uint8_t	Default: all 0	MAC address of remote node from the ARP PDU in which a conflict was detected.
abArpPdu[28]	uint8_t	Default: all 0	Copy of the raw ARP frame in which a conflict was detected.

Table 73. Acd Last Conflict Structure Description (struct EIP_DPMINTF_TI_ACD_LAST_CONFLICT_T)

Variable	Type	Value/Range	Description
bAllocControl	uint8_t	0-1 Default: 0	0: Multicast addresses shall be generated using the default allocation algorithm. When 0 is specified the values of usNumMcast and ulMcastStartAddr shall also be 0. 1: Multicast addresses shall be allocated according to the values specified in Num Mcast and Mcast Start Addr.
bReserved	uint8_t	0	
usNumMCast	uint16_t	0 (if bAllocControl == 0), Default: 0	Number of IP multicast addresses to allocate for EtherNet/IP.
ulMcastStartAddr	uint32_t	0 (if bAllocControl == 0), 0xE0000000 < addr < 0xF0000000 Default: 0	Mcast Start Addr is the starting multicast address from which Num Mcast addresses are allocated.

Table 74. Multicast Configuration Structure Description (struct EIP_DPMINTF_TI_MCAST_CONFIG_T)

Confirmation packet description

Variable	Type	Value/Range	Description
ulLen	UINT32	0	Packet data length in bytes
ulSta	UINT32		See section Status/error codes
ulCmd	UINT32	0x3613	EIP_APS_SET_CONFIGURATION_PARAMETERS_CNF

Table 75. EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_CNF_T – Set Configuration Parameters confirmation

4.2.2 Set Parameter Flags service

The host application sends the EIP_APS_SET_PARAMETER_REQ packet to activate or deactivate special functionalities or behaviors of the Firmware. The request packet therefore contains a flag field in which each bit stands for a specific functionality.

Bit	Description
0	`Flag EIP_APS_PRM_SIGNAL_MS_NS_CHANGE (0x00000001)` If set (1), the stack will notify the host application whenever the network or module status changes. LEDs at EtherNet/IP devices display the module and the network status (For more information, see section Module and network status). When enabled, the protocol stack generates indication packets EIP_APS_MS_NS_CHANGE_IND on state changes of the module or network status. If not set (0), the stack will not send any notifications.
1..31	Reserved for future use, set to zero

Table 76. EIP_APS_SET_PARAMETER_REQ Flags

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t	4	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x360A	EIP_APS_SET_PARAMETER_REQ
`tData (EIP_APS_SET_PARAMETER_REQ_T)`			
`ulParameterFlags`	uint32_t	See Table EIP_APS_SET_PARAMETER_REQ Flags for possible values	Bit field

Table 77. EIP_APS_PACKET_SET_PARAMETER_REQ_T – Set Parameter Flags request

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x360B	`EIP_APS_SET_PARAMETER_CNF`

Table 78. EIP_APS_PACKET_SET_PARAMETER_CNF_T – Confirmation to Set Parameter Flags request

4.2.3 Finish configuration of CIP objects

The packet EIP_APS_CONFIG_DONE_REQ is part of the Extended Configuration Set. The host application sends this packet to inform the protocol stack that all CIP objects have been registered and configured and thus, that the EtherNet/IP Adapter Stack configuration is finished and it is clear to start its normal operation.

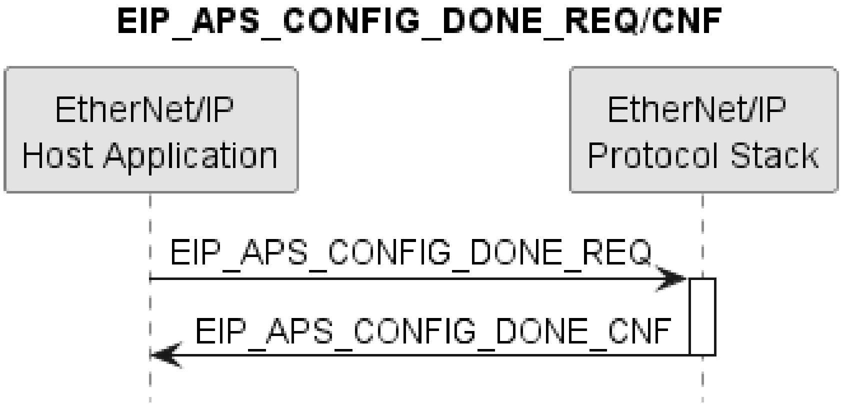

EIP_APS_CONFIG_DONE_REQ/CNF
EtherNet/IP Host Application
EtherNet/IP Protocol Stack
EIP_APS_CONFIG_DONE_REQ
EIP_APS_CONFIG_DONE_CNF

Figure 13. Sequence Diagram for the EIP_APS_CONFIG_DONE_REQ/CNF Packet

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x3614	EIP_APS_CONFIG_DONE_REQ

Table 79. EIP_APS_PACKET_CONFIG_DONE_REQ_T – Signal end of configuration request

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x3615	`EIP_APS_CONFIG_DONE_CNF`

Table 80. EIP_APS_PACKET_CONFIG_DONE_CNF_T – Confirmation of end of configuration request

4.2.4 Register an additional object class

The host application sends the request EIP_OBJECT_MR_REGISTER_REQ to register or activate an additional object class at the message router. Registration/Activation of an additional object class extends the object model of the device by the given object class (see Figure Default Hilscher Device object model for the default object model).

We distinguish between two types of non-default objects:

1. CIP object classes, which the stack already provides, but which remain deactivated in a default configuration, e.g. the Time Sync object. Sending this request for such an object will activate the object. The protocol stack subsequently processes all service requests towards such object types entirely and internally, just as it does for default objects. There is no need for the host application to provided service handlers for objects of this type. These objects can only be activated using the Extended Packet Set configuration (section Extended configuration packet set).
2. CIP objects that are not present in the protocol stack at all. The host application is responsible to implement the provided services and attributes of such an object type at class and instance level. To achieve this, the stack will forward all explicit messages addressing application-registered object classes to the host application via the indication EIP_OBJECT_CL3_SERVICE_IND.

The class code parameter `ulClass` uniquely identifies the object class. According to the CIP specification Vol. 1 section 5, the overall range of class codes splits into certain ranges as shown in Table Address Ranges for the ulClass parameter:

Address Range	Meaning
0x0001 - 0x0063	Open
0x0064 - 0x00C7	Vendor-specific
0x00C8 - 0x00EF	Reserved by ODVA for future use
0x00F0 - 0x02FF	Open
0x0300 - 0x04FF	Vendor-specific
0x0500 - 0xFFFF	Reserved by ODVA for future use

Table 81. Address Ranges for the ulClass parameter

Various volumes of the CIP specification define class code values, which are “open”. Class code values, which are “vendor-specific”, are available to extend your device’s capabilities beyond the available Open options.

> **NOTE** Note that in the vendor specific range 0x300-0x03FF, the EtherNet/IP stack provides a few built-in Hilscher-specific objects. If the host application registers such an object IDa second time, the service will succeed anyway. In such a scenario, the Host-registered object will subsequently be available to explicit services from the network, whereas the Hilscher-specific object will remain available only at the DPM packet interface for explicit service requests. Thus, the object class ID is ambigiously used in the system, and the addressing will be made unique by considering the interface as a secondary key.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination.
`ulLen`	uint32_t	12	`EIP_OBJECT_MR_REGISTER_REQ_SIZE` – Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x1A02	EIP_OBJECT_MR_REGISTER_REQ
`tData (EIP_OBJECT_MR_REGISTER_REQ_T)`			
`ulReserved1`	uint32_t	0	Reserved, set to 0
`ulClass`	uint32_t	0x1..0xFFFF	Class identifier (predefined class code as described in the CIP specification Vol. 1 section 5 (reference [5]) Take care of the address ranges specified above within Table Address Ranges for the ulClass parameter.
`ulOptionFlags`	uint32_t	Bit 0: type selector (see description) Bits 1-31 reserved for future use, set to zero	For type 1, set bit 0 to 1 (flag `EIP_OBJECT_MR_REGISTER_OPTION_FLAGS_USE_OBJECT_PROVIDED_BY_STACK`) For type 2, set bit 0 to 0 Additional CIP object that can be registered with type 1: - Time Sync object (class code 0x43)

Table 82. EIP_OBJECT_MR_PACKET_REGISTER_REQ_T – Request command for register a new class object

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x1A03	`EIP_OBJECT_MR_REGISTER_CNF`

Table 83. EIP_OBJECT_MR_PACKET_REGISTER_CNF_T – Confirmation command of register a new class object

4.2.5 Register a new Assembly instance

The host application sends the packet EIP_OBJECT_AS_REGISTER_REQ to create a new Assembly object class instance. The parameter `ulInstance` of the packet specifies the assembly instance number to create. The `ulFlags` member of the packet, amongst other options, designates the Assembly instance as an Input, Output, Input-Only or Listen-Only Connection Point.

The newly created Assembly instance may occupy a certain range of up to a size of 504 bytes in the Input- or Output-Area of the DPM, respectively. This range is specified by the member's `ulDPMOffset` and `ulSize` of the packet, where `ulDPMOffset` determines the relative memory address starting from the base address of the appropriate DPM I/O area `abPd0Input` or `abPd0Output` (see reference [1]). If multiple assembly instances are registered, you probably want to ensure that the data range of the different instances do not overlap in the DPM I/O area.

Table Assembly instance number ranges lists the assembly instance number ranges specified by the CIP Networks Library (reference [5]).

Assembly instance number range	Device profile usage	Vendor-specific device profile usage
0x0001 – 0x0063	Open (defined in device profile)	Vendor-specific
0x0064 – 0x00C7	Vendor-specific	Vendor-specific
0x00C8 – 0x00D1	Open (defined in device profile)	Vendor-specific
0x00D2 – 0x00EF	Reserved by CIP for future use	Reserved by CIP for future use
0x00F0 – 0x00FF	Vendor-specific	Vendor-specific
0x0100 – 0x02FF	Open (defined in device profile)	Vendor-specific
0x0300 – 0x04FF	Vendor-specific	Vendor-specific
0x0500 – 0xFFFF	Open (defined in device profile)	Vendor-specific
0x00010000 – 0x000FFFFF	Open (defined in device profile)	Vendor-specific
0x00100000 – 0xFFFFFFFF	Reserved by CIP for future use	Reserved by CIP for future use

Table 84. Assembly instance number ranges

NOTE The instance numbers 192 and 193 (0xC0 and 0xC1) are the Hilscher's default assembly instances for **Listen Only** and **Input Only** connection. Do not use these instance numbers for additional assembly instances when configuring the protocol stack with the Basic Configuration Packet Set.

NOTE When using the Basic Configuration Packet Set, the stack creates default assemblies at offsets 0 in the DPM input and output areas.

Further properties of the assembly instance are configurable with the Assembly Flags parameter `ulFlags` of this request packet. For descriptions of the valid assembly flags, see Table Assembly Types and Option Flags.

Per default, as long as no data has ever been set and no connection is established toward the assembly instance, the assigned DPM I/O area holds zeroed data.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination 32 (0x20): Destination is the protocol stack
`ulLen`	uint32_t	16	`EIP_OBJECT_AS_REGISTER_REQ_SIZE` - Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x1A0C	EIP_OBJECT_AS_REGISTER_REQ
`tData (EIP_OBJECT_AS_REGISTER_REQ_T)`			
`ulInstance`	uint32_t	0x0000001... 0xFFFFFFFE (except 0xC0 and 0xC1, see description above)	Assembly instance number See Table Assembly instance number ranges.
`ulDPMOffset`	uint32_t	0..5760	DPM offset of the instance data area **Note**: This offset is not the total DPM offset. It is the relative offset within the beginning of the corresponding input/output data areas `abPd0Input[5760]` and `abPd0Output[5760]` The first instance (for each data direction) that is created usually will have `ulDPMOffset = 0`. If multiple assembly instances are registered, make sure that the data ranges of these instances do not overlap in the DPM.
`ulSize`	uint32_t	1..504 *[..516]*	Size of the data area for the assembly instance data including: *[4 bytes]* when optional Run/Idle header is active *[4 bytes]* when optional Sequence count is active *[4 bytes]* when optional Producing flags is active
`ulFlags`	uint32_t	Bitmap	Property Flags for the assembly instance, see Table Assembly Types and Option Flags.

Table 85. EIP_OBJECT_AS_PACKET_REGISTER_REQ_T – Request command for creating an assembly instance

The following table describes the available bits to configure each assembly’s type and options:

Bits	Name (Bitmask)	Description
31	`EIP_AS_TYPE_LISTENONLY (0x80000000)`	Assembly type: Setting this flag declares an assembly of type "listen only".
30	`EIP_AS_TYPE_INPUTONLY (0x40000000)`	Assembly type: Setting this flag declares an assembly of type "input only".
31...13	Reserved	Reserved for future use, set to zero
12	`EIP_AS_OPTION_MAP_PRODUCING_FLAGS (0x00001000)`	Assembly option: For output (producing) assemblies, this bit decides whether an additional 4-byte flag field is available preceding the assembly’s output data (see Figure DPM Input/Output area layout). This bit must not be set for input assembly instances. The producing flag field offers the host application additional control related to the output data. If this bit is set, the 4-byte flag field will be part of the output data image. See Table Definition of the “Producing Flags” for details on what functionality comes with this flag field.
11	`EIP_AS_OPTION_RXTRIGGER (0x00000800)`	Assembly option: If this flag is set for an input assembly and the DPM handshake mode was set to the EtherNet/IP-specific mode "Receive (RX) Triggered Handshake Mode" (see Input handshake mode / output handshake mode), then each change of the assembly’s data will toggle the DPM handshake bits, promptly presenting the newly received data to the application.
10	`EIP_AS_OPTION_MAP_SEQCOUNT (0x00000400)`	Assembly option: This flag decides whether the 2-byte data sequence count field of the EtherNet/IP PDU will be mapped into the I/O area. Four additional bytes have to be reserved in the assembly’s size and offsets. The lower two bytes will contain the sequence count value consistent to the assembly’s data. The byte order is little endian. The sequence counter wraps-around to zero at value 65536. For input assemblies, thus, the host application has the possibility to detect which assemblies have recently received new data. If the bit is set, the sequence count field will be part of the input data image (see Figure DPM Input/Output area layout). The most recent sequence count field encountered on the network is copied into the DPM and can be read by the host application. **Note**: - The sequence count is incremented only when the connected PLC application updates its production data. - The sequence count is not designed to detect lost packets - The sequence count information remains unchanged when the assembly data is modified over an EtherNet/IP explicit service, whereas the data may has changed. For output assemblies, thus, the host application has to control the value of the sequence counter directly. If the bit is set, the sequence count field will be part of the output data image. The host application will increment the sequence count value with each update of its output I/O data.
9	`EIP_AS_OPTION_INVISIBLE (0x00000200)`	Assembly option: This flag decides whether EtherNet/IP explicit services from the network can access the assembly instance. Flag is set: The assembly instance is not visible at the network. Flag is not set: The assembly instance is visible at the network.

Bits	Name (Bitmask)	Description
8	`EIP_AS_OPTION_MAP_RUNIDLE` (`0x00000100`)	Assembly option: If the bit is set, the 4-byte run/idle header will be part of the I/O data image. An additional 4-byte DPM-Mapping preceeding the assembly data contains the RUN/IDLE header as also contained in the I/O frames. Four additional bytes have to be reserved in the assembly's size and offsets (see Figure DPM Input/Output area layout). Byte order is little endian. For input assemblies that receive the run/idle header, this allows the host application to evaluate the run/idle information on its own. **Note**: - The RUN/IDLE status in the DPM is only updated when the connected PLC application updates its production data, i.e. the received sequence count field increments. - The RUN/IDLE information remains unchanged when the assembly data is modified over an EtherNet/IP explicit service, whereas the data may has changed. For output assemblies, that send the run/idle header[1], this allows the host application to have direct control over the RUN/IDLE status of a connection.
7	`EIP_AS_OPTION_FIXED_SIZE` (`0x00000080`)	Assembly option: This flag decides whether the assembly instance allows to be establish connections with a smaller connection size than specified for the assembly. If it is not set, any connection size up to the specified size will be accepted. This flag is not allowed for assemblies of types input only, listen only and configuration. If the bit is set (1), the connection size in a ForwardOpen must directly correspond to `ulSize`. If the bit is not set (0), the connection size can be smaller or equal to `ulSize`. Example: ■ ulSize = 16 (Bit 7 of ulFlags is 0) A connection to this assembly instance can be opened with a smaller or matching I/O size, e.g. 8. ■ ulSize = 6 (Bit 7 of ulFlags is 1) A connection can only be opened with a matching I/O size, i.e. 6 bytes.
6	`EIP_AS_OPTION_HOLDLASTSTATE` (`0x00000040`)	Assembly option: This flag decides whether the data that is mapped into the corresponding DPM memory area is cleared upon closing of the connection or whether the last sent/received data remains unchanged in the memory. If the bit is set, the data will remain unchanged.
5	`EIP_AS_TYPE_CONFIG` (`0x00000020`)	Assembly type: This flag signifies that the current assembly is a configuration assembly, which can be used to receive configuration data upon connection establishment. **Note**: Compared to input and output assembly instances a configuration instance is set only once via the Forward_Open frame. It is not exchanged cyclically. On connection establishment the configuration data is sent to the host application via the packet EIP_OBJECT_CL3_SERVICE_IND, service `CIP_CMD_SET_ATTR_SINGLE`, addressing attribute 3 of the corresponding assembly object instance.
4	Reserved	Reserved for future use, set to zero
3	`EIP_AS_OPTION_NO_RUNIDLE` (`0x00000008`)	Assembly option: If set, the assembly data is considered as modeless (i.e. it does not contain run/idle information). This parameter has to be consistent with your device's EDS. If not set, the assembly instance's real time format is the 32-Bit Run/Idle header.
2...1	Reserved	Reserved for future use, set to zero

Bits	Name (Bitmask)	Description
0	EIP_AS_TYPE_INPUT (0x00000001)	Assembly type: This flag configures the newly registered assembly instance as an input assembly or an output assembly. Flag is set: Assembly instance is an **input** assembly. An input assembly will only receive data from the network. Flag is not set: Assembly instance is an **output** assembly. An output assembly will transmit data to the network.

[1] This is unusual for adapter devices. In most setups, no RUN/IDLE status is sent in T2O direction.

Table 86. Assembly Types and Option Flags

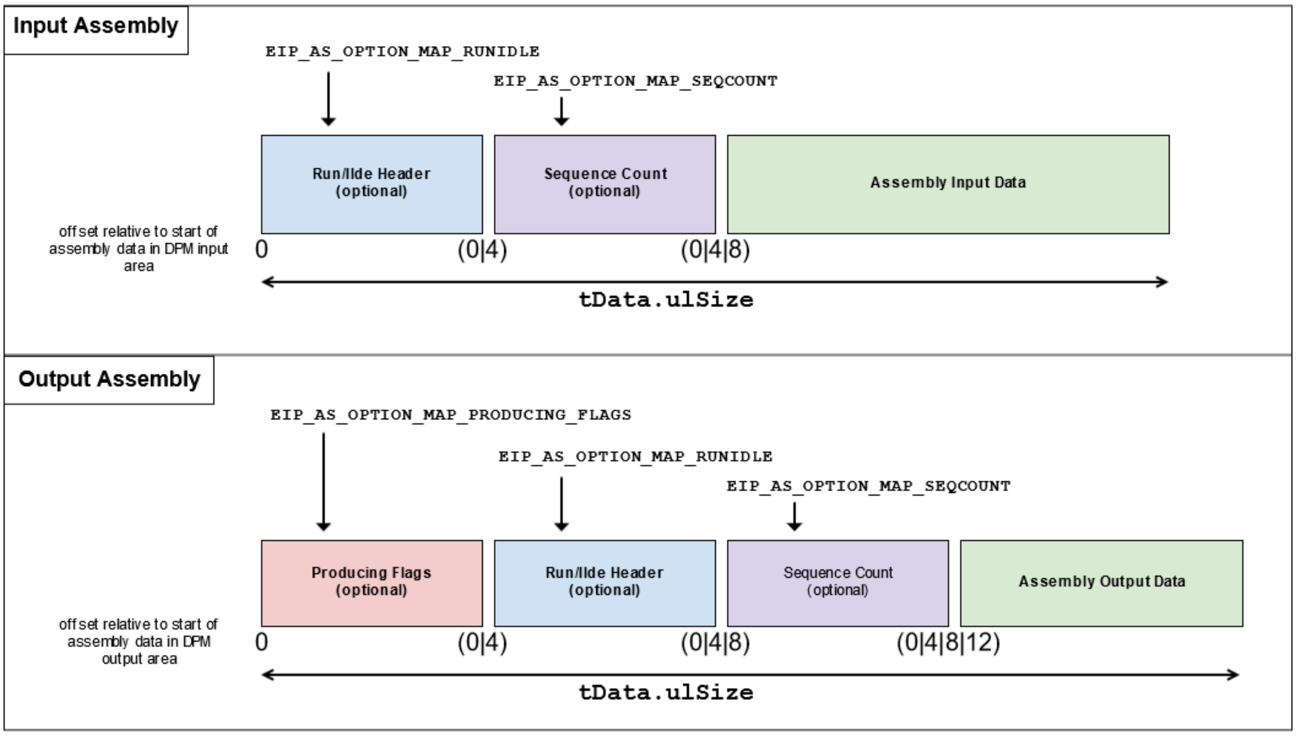

Input Assembly
EIP_AS_OPTION_MAP_RUNIDLE
EIP_AS_OPTION_MAP_SEQCOUNT
Run/Idle Header (optional)
Sequence Count (optional)
Assembly Input Data
offset relative to start of assembly data in DPM input area
0
(0|4)
(0|4|8)
tData.ulSize
Output Assembly
EIP_AS_OPTION_MAP_PRODUCING_FLAGS
EIP_AS_OPTION_MAP_RUNIDLE
EIP_AS_OPTION_MAP_SEQCOUNT
Producing Flags (optional)
Run/Idle Header (optional)
Sequence Count (optional)
Assembly Output Data
offset relative to start of assembly data in DPM output area
0
(0|4)
(0|4|8)
(0|4|8|12)
tData.ulSize

Figure 14. DPM Input/Output area layout according to options EIP_AS_OPTION_MAP_RUNIDLE, EIP_AS_OPTION_MAP_SEQCOUNT, EIP_AS_OPTION_MAP_PRODUCING_FLAGS and the given Assembly size

The following table describes the structure of the “Producing flags” bitfield that may be mapped into the DPM output area for each assembly by means of option `EIP_AS_OPTION_MAP_PRODUCING_FLAGS` (see Figure DPM Input/Output area layout).

Bits	Name (Bitmask)	Description
0	`CIP_AS_PRODUCING_FLAG_TRIGGER_PROCESS_DATA_UPDATE (0x00000001)`	Controls whether or not the process data frame shall be updated with the provided assembly data. **For I/O connections of type "cyclic"**: **Value 0**: the assembly is updated with the new producing data, but the data will not be copied into the process data frame. This means the data will not be sent to the originator of the connection. **Value 1**: the assembly is updated with the new producing data, and the data will be copied into the process data frame. **Note**: typically, for I/O connections of type "cyclic" this bit should be set to value 1 all the time. **For I/O connections of type "application triggered"**: By using this bit, the application can control whether or not the protocol stack shall send a new process data frame on the network. Usually, when running an application controlled connection, each update of the assembly’s process data will trigger a new process data frame on the network. However, in case multiple output assemblies are used, the application may not want to set new data for all assemblies at the same time, but only for selected assembly instances (e.g. CIP Safety). **Value 0**: the assembly is updated with the new producing data, but the data will not be copied into the process data frame. No process data frame will be sent. **Value 1**: the assembly is updated with the new producing data, and the data will be copied into the process data frame. Process data frame will be sent right away.
1-31	Reserved	Reserved for future use, set to zero

Table 87. Definition of the “Producing Flags”

Source code example

The following sample code shows how to fill in the parameter fields of the EIP_OBJECT_AS_REGISTER_REQ packet in order to create two assembly instances, one input and one output instance.

```
/* Fill the EIP_OBJECT_AS_REGISTER_REQ packet to create an input (T->O) assembly instance 100 that
holds 16 bytes of data, has the modeless real-time format and does not allow smaller
connection sizes. */

EIP_OBJECT_AS_PACKET_REGISTER_REQ_T tReq;

  tReq.tHead.ulCmd = EIP_OBJECT_AS_REGISTER_REQ;
  tReq.tHead.ulLen = EIP_OBJECT_AS_REGISTER_REQ_SIZE;

  tReq.tData.ulInstance  = 100;
  tReq.tData.ulSize     = 16;
  tReq.tData.ulFlags    = EIP_AS_TYPE_OUTPUT |EIP_AS_OPTION_NO_RUNIDLE | EIP_AS_OPTION_FIXED_SIZE;
  tReq.tData.ulDPMOffset = 0;

  /* Fill the EIP_OBJECT_AS_REGISTER_REQ packet to create an output (O□T) assembly instance 101
  that holds 8 bytes of data, has the run/idle real-time format and does allow smaller
  connection sizes. */

  EIP_OBJECT_AS_PACKET_REGISTER_REQ_T tReq;

  tReq.tHead.ulCmd = EIP_OBJECT_AS_REGISTER_REQ;
  tReq.tHead.ulLen = EIP_OBJECT_AS_REGISTER_REQ_SIZE;

  tReq.tData.ulInstance  = 101;
  tReq.tData.ulSize     = 8;
  tReq.tData.ulFlags    = EIP_AS_TYPE_INPUT;
  tReq.tData.ulDPMOffset = 0;
```

Confirmation packet description

Variable	Type	Value/Range	Description
ulLen	uint32_t	20	EIP_OBJECT_AS_REGISTER_CNF_SIZE - Packet data length in bytes
ulSta	uint32_t		See section Status/error codes
ulCmd	uint32_t	0x1A0D	EIP_OBJECT_AS_REGISTER_CNF
tData (EIP_OBJECT_AS_REGISTER_CNF_T)			
ulInstance	uint32_t		Instance of the Assembly Object (from the request packet)
ulDPMOffset	uint32_t		Offset of the data in the DPM (from the request packet)
ulSize	uint32_t	<=504 *[..516]*	Size of the assembly instance data (from the request packet)
ulFlags	uint32_t		Property flags of the assembly instance (from the request packet)
hDataBuf	uint32_t		Ignore (deprecated)

Table 88. EIP_OBJECT_AS_PACKET_REGISTER_CNF_T – Confirmation command of register a new class object

4.2.6 Register service

The host application sends EIP_OBJECT_REGISTER_SERVICE_REQ to register a service, which is not directly bound to a CIP object.

Usually, services use the CIP addressing format Class → Instance → Attribute. In contrast, if for example Tags are to be supported which allow addressing the device's data using string identifiers, there may be the requirement to have object-independent/ standalone services using non-standard addressing formats.

Therefore, the host application can register a vendor specific service code (see Table Specified ranges of numeric values of service codes (variable bService)). If the device then receives a corresponding service request (sent from a Scanner or other EtherNet/IP client), it will forward the request to the host application via the indication EIP_OBJECT_CL3_SERVICE_IND.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination. Set to 32 (0x20): Destination is the protocol stack
`ulLen`	uint32_t	1	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x00001A44	EIP_OBJECT_REGISTER_SERVICE_REQ
`tData (EIP_OBJECT_REGISTER_SERVICE_REQ_T)`			
`bService`	uint8_t		Vendor-specific service code (see Table Specified ranges of numeric values of service codes (variable bService))

Table 89. EIP_OBJECT_PACKET_REGISTER_SERVICE_REQ_T - Register Service

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x00001A45	`EIP_OBJECT_REGISTER_SERVICE_CNF`

Table 90. EIP_OBJECT_PACKET_REGISTER_SERVICE_CNF_T – Confirmation command for Register Service confirmation

4.2.7 Set Parameter

The host application sends EIP_OBJECT_SET_PARAMETER_REQ to activate or deactivate certain non-default behavior of the EtherNet/IP protocol stack.

Table EIP_OBJECT_SET_PARAMETER_REQ – Packet Status/Error gives an overview of the bits, which can be set for the member `ulParameterFlags` of the request in order to control the protocol stack's behavior.

Parameter Flags – ulParameterFlags

Bit	Description
0	**EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING** Enables or disables forwarding of Forward_Open and Forward_Close frames to the host application. **Forward_Open frames:** If set (1), all Forward_Open frames will be forwarded to the host application via the packet EIP_OBJECT_LFWD_OPEN_FWD_IND. If not set (0), the Forward_Open will not be forwarded. **Forward_Close frames:** If set (1), all Forward_Close frames will be forwarded via the packet EIP_OBJECT_FWD_CLOSE_FWD_IND. If not set (0), the Forward_Open/Close will not be forwarded.
1	**EIP_OBJECT_PRM_DISABLE_FLASH_LEDS_SERVICE** Enables or disables the Flash_LEDs service (0x4B) of the CIP Identity object. The Flash_LEDs service is enabled by default. If set (1), the Flash_LEDs service is disabled. If not set (0), the Flash_LEDs service is enabled.
2	**EIP_OBJECT_PRM_DISABLE_TRANSMISSION_TRIGGER_TIMER** This flag affects the timing of data production in case of "Application Object" or "Change Of State" triggered data. Setting this flag will turn off the transmission trigger timer for all application-triggered and change-of-state connections. Data production is then only triggered by the EtherNet/IP Application when providing new data to the protocol stack (e.g. each call of xChannelIoWrite() will trigger a new data from on the network). For connections of type "cyclic", the transmission trigger timer will not be disabled.
3	**EIP_OBJECT_PRM_HOST_CONTROLS_IDENTITY_STATE_ATTRIBUTE_8** This flag affects the handling of attribute 8 (State) of the Identity object. Usually, the protocol stack controls this attribute autonomously. However, there are types of host applications that need to control this attribute them self (e.g. CIP Safety). If set, the protocol stack will stop controlling the state attribute. Instead, the host application has to take care of the content of the attribute. The application has to send a CIP set attribute single service (0x10) to attribute 8 (use packet command EIP_OBJECT_CIP_SERVICE_REQ). Note: The designer of the application has to decide whether or not it needs this feature. Activating and, after a while, deactivating the write access must be avoided as this might lead to invalid state attribute values.
4	**EIP_OBJECT_PRM_ENABLE_NULL_FWRD_OPEN** This flag affects whether or not the firmware will be capable of processing NULL-ForwardOpen requests, i.e. ForwardOpen requests which have the transport type NULL for both directions, O2T and T2O. If set, NULL ForwardOpen requests from the network will be accepted. If cleared they will be rejected with an appropriate error code. Per default, NULL ForwardOpen support is disabled. See also section NULL ForwardOpen.
5-31	Reserved for future use, set to 0

Table 91. EIP_OBJECT_SET_PARAMETER_REQ – Packet Status/Error

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t	4	`EIP_OBJECT_SET_PARAMETER_REQ_SIZE` Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x00001AF2	EIP_OBJECT_SET_PARAMETER_REQ
`tData (EIP_OBJECT_SET_PARAMETER_REQ_T)`			
`ulParameterFlags`	uint32_t		See Table EIP_OBJECT_SET_PARAMETER_REQ – Packet Status/Error

Table 92. EIP_OBJECT_PACKET_SET_PARAMETER_REQ_T – Set Parameter request

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t		See Status/error codes
`ulCmd`	uint32_t	0x00001AF3	`EIP_OBJECT_SET_PARAMETER_CNF`

Table 93. EIP_OBJECT_PACKET_SET_PARAMETER_CNF_T – Set Parameter confirmation

4.2.8 CIP Service request

The host application issues CIP service requests toward the EtherNet/IP stack by sending the request packet EIP_OBJECT_CIP_SERVICE_REQ. The service to request is denoted by its service code passed in member `bService` in the request packet. Typically, a service addresses an object class or instance and optionally an attribute of that class or instance by means of the members `ulClass`, `ulInstance`, `ulAttribute` and `ulMember`.

If the requested service requires parameter data to be sent along with the service, this parameter data has to be encoded into member `abData[]` of the packet. In those cases, the number of bytes in `abData[]` must then be added to the `ulLen` field of the packet header.

The result of the service is returned in the fields `ulGRC` (Generic Error Code) and `ulERC` (Additional Error Code) of the confirmation packet. The host application should evaluate the Generic Error Code to determine about success or failure of the service request. In case of successful execution, the variables `ulGRC` and `ulERC` of the confirmation packet will have the value 0. In most cases, the stack will only set the Generic Error Code to a nonzero value on errors, whereas only a small number of services set the Extended Error Code to provide additional diagnostic information. Table CIP Generic Status Codes Definitions (Variable `ulGRC`) shows possible GRC values and their meaning.

If data is received along with the confirmation, it correspondingly can be found in the array `abData[]`. The `ulLen` field of the packet header specifies the overall number of bytes received with the confirmation packet, including the response data array.

Generic Error Codes as denoted by member ulGRC of the Service response

For a comprehensive description of the errors in the following table, see section Status/error codes of this document.

Define	Value	Name
CIP_GSR_SUCCESS	0x00	No error
CIP_GSR_FAILURE	0x01	Connection Failure
CIP_GSR_NO_RESOURCE	0x02	Resource unavailable
CIP_GSR_BAD_DATA	0x03	Invalid parameter value (deprecated, use CIP_GSR_INVALID_PARAMETER)
CIP_GSR_BAD_PATh	0x04	Path segment error
CIP_GSR_BAD_CLASS_INSTANCE	0x05	Path destination unknown
CIP_GSR_PARTIAL_DATA	0x06	Partial Transfer
CIP_GSR_CONN_LOST	0x07	Connection Lost
CIP_GSR_BAD_SERVICE	0x08	Service not supported
CIP_GSR_BAD_ATTR_DATA	0x09	Invalid attribute data detected
CIP_GSR_ATTR_LIST_ERROR	0x0A	Attribute List Error
CIP_GSR_ALREADY_IN_MODE	0x0B	Already in requested mode/state
CIP_GSR_BAD_OBJ_MODE	0x0C	Object state conflict
CIP_GSR_OBJ_ALREADY_EXISTS	0x0D	Object already exists
CIP_GSR_ATTR_NOT_SETTABLE	0x0E	Attribute not settable
CIP_GSR_PERMISSION_DENIED	0x0F	Privilege violation
CIP_GSR_DEV_IN_WRONG_STATE	0x10	Device state conflict
CIP_GSR_REPLY_DATA_TOO_LARGE	0x11	Reply data too large
CIP_GSR_FRAGMENT_PRIMITIVE	0x12	Fragmentation of a primitive value
CIP_GSR_CONFIG_TOO_SMALL	0x13	Not enough data
CIP_GSR_UNDEFINED_ATTR	0x14	Attribute not supported
CIP_GSR_CONFIG_TOO_BIG	0x15	Too much data
CIP_GSR_OBJ_DOES_NOT_EXIST	0x16	Object does not exist
CIP_GSR_NO_FRAGMENTATION	0x17	Service fragmentation sequence not in progress
CIP_GSR_DATA_NOT_SAVED	0x18	No stored attribute data
CIP_GSR_DATA_WRITE_FAILURE	0x19	Store operation failure
CIP_GSR_REQUEST_TOO_LARGE	0x1A	Routing failure, request packet too large
CIP_GSR_RESPONSE_TOO_LARGE	0x1B	Routing failure, response packet too large
CIP_GSR_MISSING_LIST_DATA	0x1C	Missing attribute list entry data
CIP_GSR_INVALID_LIST_STATUS	0x1D	Invalid attribute value list

Define	Value	Name
CIP_GSR_SERVICE_ERROR	0x1E	Embedded Service Error
CIP_GSR_CONN_RELATED_FAILURE	0x1F	Vendor-specific error, currently unused in this protocol stack
CIP_GSR_INVALID_PARAMETER	0x20	Invalid parameter
CIP_GSR_WRITE_ONCE_FAILURE	0x21	Write-once value or medium already written
CIP_GSR_INVALID_REPLY	0x22	Invalid Reply received
CIP_GSR_BAD_KEY_IN_PATH	0x25	Key failure in path
CIP_GSR_BAD_PATH_SIZE	0x26	Path size invalid
CIP_GSR_UNEXPECTED_ATTR	0x27	Unexpected attribute in list
CIP_GSR_INVALID_MEMBER	0x28	Invalid Member ID
CIP_GSR_MEMBER_NOT_SETTABLE	0x29	Member not settable
CIP_GSR_GROUP2_ONLY_S_GENERAL_FAIL	0x2A	Group 2 only server general failure
CIP_GSR_UNKNOWN_MODBUS_ERROR	0x2B	Unknown Modbus Error
CIP_GSR_ATTRIBUTE_NOT_GET	0x2C	Attribute not gettable
CIP_GSR_INSTANCE_NOT_DELETE	0x2D	Instance cannot be deleted
CIP_GSR_SERVICE_NOT_SUPPORT_PATH	0x2E	Service not supported for specified path

Table 94. CIP Generic Status Codes Definitions (Variable `ulGRC`)

Extended Error Codes as denoted by member ulERC of the Service response

The EtherNet/IP protocol stack rarely uses Extended Error Codes and thus this manual does not cover their definitions. Anyway, certain services of the Connection Manager object make use of extended error codes. For definitions and descriptions of the CIP extended error codes, see section 3-5.5 of the CIP specification [5].

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination. Set to 0: Destination is operating system 32 (0x20): Destination is the protocol stack
`ulLen`	uint32_t	20+n	Packet data length in bytes n = Length of service data in bytes (see field `abData[]`)
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x1AF8	EIP_OBJECT_CIP_SERVICE_REQ
`tData (EIP_OBJECT_CIP_SERVICE_REQ_T)`			
`bService`	uint8_t	Valid service code	CIP service code (see also Table Service Codes for the Common Services according to the CIP specification)
`abPad0[3]`	uint8_t	0	Padding. Set to zero.
`ulClass`	uint32_t	Valid Class ID	CIP Class ID (according to *“The CIP Networks Library, volume 1 Common Industrial Protocol Specification, section 5, table 5-1.1”*) For available object classes see section Hilscher EtherNet/IP stack capabilities.
`ulInstance`	uint32_t	Valid Instance number	CIP Object Instance number. For available object classes and instances, see section Hilscher EtherNet/IP stack capabilities.
`ulAttribute`	uint32_t	Valid Attribute number	CIP attribute number (required for get/set attribute only, otherwise set it to 0). For available object classes and attributes, see section Hilscher EtherNet/IP stack capabilities.
`ulMember`	uint32_t	Valid member number	CIP member number Typically, this parameter can be set to 0 as most CIP attributes do not have members.

Variable	Type	Value/Range	Description
`abData[]`	uint8_t		CIP service data The number of bytes n provided in this byte array must be added to the packet header length field `ulLen`. Set the proper packet length as follows: `ptReq→tHead.ulLen = EIP_OBJECT_CIP_SERVICE_REQ_SIZE + n` Range of n: 0 - 1390

Table 95. EIP_OBJECT_PACKET_CIP_SERVICE_REQ_T – CIP Service request

Confirmation packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t		Destination
`ulLen`	uint32_t	28+n	Packet data length in bytes n = Length of service data in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x1AF9	`EIP_OBJECT_CIP_SERVICE_CNF`
`tData (EIP_OBJECT_CIP_SERVICE_CNF_T)`			
`bService`	uint8_t	Valid service code	CIP service code
`abPad0`	uint8_t[3]	0	Padding. Set to zero.
`ulClass`	uint32_t	Valid Class ID	CIP Class ID (according to *“The CIP Networks Library, volume 1 Common Industrial Protocol Specification, section 5, table 5-1.1”*
`ulInstance`	uint32_t	Valid Instance number	CIP instance number
`ulAttribute`	uint32_t	Valid Attribute number	CIP attribute number (for get/set attribute only)
`ulMember`	uint32_t	Valid Member number	CIP member number
`ulGRC`	uint32_t		Generic error code according to *“The CIP Networks Library, volume 1 Common Industrial Protocol Specification, section 5, appendix B-1.)* (see also Table CIP Generic Status Codes Definitions (Variable `ulGRC`))
`ulERC`	uint32_t		Additional error code.
`abData[]`	uint8_t		CIP service data The number of bytes provided in this byte array must be calculated using the packet header length field `ulLen`. Proceed as follows to get the data size: number of bytes provided in abData = `tHead.ulLen - EIP_OBJECT_CIP_SERVICE_REQ_SIZE`

Table 96. EIP_OBJECT_PACKET_CIP_SERVICE_CNF_T – Confirmation to CIP Service request

4.2.9 Set Watchdog Time

The host application sends packet HIL_SET_WATCHDOG_TIME_REQ to enable the netX watchdog timer with the specified timeout value. This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [9].

4.2.10 Register/Unregister Application

The host application sends packets HIL_REGISTER_APP_REQ and HIL_UNREGISTER_APP_REQ, respectively, to register or unregister the host application with the protocol stack. Unless an application has registered, the stack will not generate any indications toward the host application. This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [9].

4.2.11 Start/Stop Communication

The host application sends packet HIL_START_STOP_COMM_REQ to instruct the EtherNet/IP stack to start or stop network communication, i.e. to set or clear the netX's `BUS_ON` signal, according to the contained parameter.

This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [9].

For a description of the BusOff and BusOn behavior, see section Bus State.

4.2.12 Channel Init

The host application sends packet HIL_CHANNEL_INIT_REQ to trigger a channel initialization at the protocol stack. Channel Initialization causes the stack's AP task to perform a reset and to reinitialize.

This is a generic packet, which is not specific to the EtherNet/IP protocol stack. For a comprehensive description, refer to reference [9].

Anyway, the actions performed during a channel initialization are partly specific to the EtherNet/IP stack. At least, the protocol stack will perform the following actions:

- Bring the stack to a defined unconfigured state:
  - Clear READY and RUN bits
  - Set BUS_OFF and stop all communication
  - Call the object-specific reset functions of all CIP objects
  - Unregister all services previously registered with EIP_OBJECT_REGISTER_SERVICE_REQ
  - Remove assembly and connection configuration.
- If applicable: apply configuration from database,
- If applicable: apply configuration from Basic Configuration Packet Set (see section Basic configuration packet set),
- Reply with `HIL_CHANNEL_INIT_CNF`

4.3 Acyclic events indicated by the stack

The protocol stack generates indication packets towards the host application to indicate certain acyclic events. Depending on the stack's configuration/parameters, it may generate the following indications:

Packet	Command code (IND)
EIP_OBJECT_RESET_IND	0x00001A24
EIP_OBJECT_CONNECTION_IND	0x00001A2E
EIP_OBJECT_CL3_SERVICE_IND	0x00001A3E
EIP_OBJECT_CIP_OBJECT_CHANGE_IND	0x00001AFA
HIL_LINK_STATUS_CHANGE_IND	0x00002F8A
EIP_APS_MS_NS_CHANGE_IND	0x0000360C
EIP_OBJECT_LFWD_OPEN_FWD_IND	0x00001A60
EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND	0x00001A4C
EIP_OBJECT_FWD_CLOSE_FWD_IND	0x00001A4E
HIL_STORE_REMANENT_DATA_IND	0x00002F8E

Table 97. Overview: Indications of the EtherNet/IP Adapter

4.3.1 Application compliance

Indications generated by the protocol stack toward the host application, which processes these indications and replies, are critical due to the following reasons:

- A subset of the indications propagate in a synchronous manner, i.e. a particular service request issued by a remote host triggers the indication and that service request cannot be processed any further until the host application has replied to the indication. If the host application would reply with significant delay, it would cause a timeout condition on the remote host.
- The number of packets the stack uses to send indications, especially synchronous indications, are limited. If the host application fails to reply to these indications in time, the protocol stack would run out of packets and consequently, would fail to indicate further events to the host application. This scenario, depending on the particular use case, can be considered a software failure.

To mitigate these effects, the protocol stack implements a three-second timeout on all synchronous indications. If the host would fail to process an indication within this timeout interval, the protocol stack continues to process the causal service. In these cases, the service request is rejected and replied to with an error status 'Embedded service error' (see Table General Error Codes according to CIP Standard).

Thus, a compliant host application has to process indications without significant delay and must respond to all of them. Long-running operations, which would delay or block replies to indication packets may cause blocking of the system and data loss.

4.3.2 Indication of a reset request from the network

The indication EIP_OBJECT_RESET_IND notifies the host application about a reset service request from the network. This means an EtherNet/IP device (could also be a tool) just sent a reset service (CIP service code 0x05) to the device and waits for a response.

As soon as the host application sends the response to this service indication, the EtherNet/IP stack will send the response to the reset service request on the network. Afterwards, the host performs the actual reset, which is described in the sections Host application behavior and Reset).

The reset service indication is assigned an internal timeout of three seconds as described in section Application compliance. If the host fails to reply to the indication within that timeout interval, the protocol stack will respond to the service request on its own behalf, rejecting it with an error code.

The EtherNet/IP stack implements two different reset types that can be requested via the service's parameter: A value of 0 specifies a simple power cycle request and a value of 1 specifies an additional “return to factory defaults” request. Table Allowed Values of ulResetTyp reflects the reset service parameters as defined in the CIP specification.

When the host receives the indication EIP_OBJECT_RESET_IND with reset type 1, then it is also responsible to restore the default configuration by sending the request HIL_DELETE_CONFIG_REQ.

Value	Meaning as defined in the CIP Specification, Volume 1
0	Reset shall be done emulating power cycling of the device.
1	Return as closely as possible to the factory default configuration. Reset is then done emulating power cycling of the device.
2	This type of reset is not supported.
3 - 99	Reserved by CIP
100 - 199	Vendor-specific
200 - 255	Reserved by CIP

Table 98. Allowed Values of ulResetTyp

The host application has the possibility to deny the reset request by setting a non-zero status code such as `ERR_HIL_FAIL` in the `ulSta` member of the response packet `EIP_OBJECT_RESET_RES`. Two error conditions are defined:

- The device does not support the requested CIP Reset type as denoted by member `ulResetTyp` of the indication packet. In this case the host application shall reply with `ERR_HIL_INVALID_PARAMETER` in `tHead.uSta` in order for the device to reply with `CIP_GSR_INVALID_PARAMETER` (0x20) on the network.
- The device temporarily cannot serve the CIP reset due to its internal state, but will be able to handle it at a later point in time. In this case the host application shall reply with any nonzero status code in `tHead.uSta` other than `ERR_HIL_INVALID_PARAMETER` in order for the device to reply with `CIP_GSR_DEV_IN_WRONG_STATE` (0x10) on the network.

Indication packet description

Variable	Type	Value/Range	Description
ulLen	uint32_t	8	Packet data length in bytes
ulSta	uint32_t	0	Status not in use for indication.
ulCmd	uint32_t	0x00001A24	EIP_OBJECT_RESET_IND
tData (EIP_OBJECT_RESET_IND_T)			
ulDataIdx	uint32_t	0	Ignore (Deprecated)
ulResetTyp	uint32_t	0..1, 100-199	Type of the reset 0: Reset is done emulating power cycling of the device(default) 1: Return as closely as possible to the factory default configuration. Reset is then done emulating power cycling of the device. 100-199: Vendor-specific

Table 99. EIP_OBJECT_PACKET_RESET_IND_T – Reset Request from Bus indication

Response packet description

Variable	Type	Value/Range	Description
ulDest	uint32_t		Destination. Use value from indication
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	SUCCESS_HIL_OK, ERR_HIL_FAIL	See section Status/error codes SUCCESS_HIL_OK – reset is accepted ERR_HIL_FAIL – reset is denied
ulCmd	uint32_t	0x00001A25	EIP_OBJECT_RESET_RES

Table 100. EIP_OBJECT_PACKET_RESET_RES_T – Response to Indication to Reset request

4.3.3 Connection State Change indication

The indication EIP_OBJECT_CONNECTION_IND indicates to the host application a change in the state of exactly one connection, i.e. if that connection was established or closed. For the adapter role, this applies to all CIP connections, i.e. CIP class 0,1 and 3 connections.

The indication specifies the new connection state via member `ulConnectionState` and the type of connection `bConnType`, which is affected. Furthermore, all connection parameters are provided to uniquely identify the affected connection and counts of established connections of the different types are provided.

Connection state - `ulConnectionState`

Member `ulConnectionState` indicates whether a connection has been established or closed.

ulConnectionState	Numeric Value	Meaning
EIP_UNCONNECT	0	Connection has been closed. If connection timed out, the value of `ulExtendedState` will be EIP_CONN_STATE_TIMEOUT, otherwise 0.
EIP_CONNECTED	1	Connection has been established

Table 101. Meaning of variable ulConnectionState

Number of exclusive owner connections – `usNumExclusiveowner`

The number of currently established implicit connections (CIP Class 0/1) of type “Exclusive Owner”.

Number of Input only connections – `usNumInputOnly`

The number of currently established implicit connections (CIP Class 0/1) of type “Input Only”.

Number of listen only connections – `usNumListenOnly`

The number of currently established implicit connections (CIP Class 0/1) of type “Listen Only”.

Number of explicit messaging connections – `usNumExplicitMessaging`

The number of currently established explicit messaging (CIP Class 3) connections.

Connection type - `bConnType`

Member `bConnType` specifies the type of the connection affected by the state change:

bConnType	Numeric Value	Meaning
EIP_CONN_TYPE_CLASS_0_1_EXCLUSIVE_OWNER	1	Implicit exclusive owner connection
Reserved	2	Reserved for future use
EIP_CONN_TYPE_CLASS_0_1_LISTEN_ONLY	3	Implicit listen only connection
EIP_CONN_TYPE_CLASS_0_1_INPUT_ONLY	4	Implicit input only connection
EIP_CONN_TYPE_CLASS_3	5	Explicit connection
EIP_CONN_TYPE_ORIGINATOR_CLASS_0_1	16	Implicit originator connection
EIP_CONN_TYPE_ORIGINATOR_CLASS_3	32	Explicit originator connection

Table 102. Meaning of variable bConnType

Class to which the connection was directed - `ulClass`

For implicit connections (class0/1, Exclusive Owner, Input Only), the `ulClass` field is normally 0x04, which is the assembly object class ID.

For explicit connections, the ulClass field is 0x02, which is the Message Router object class ID.

Instance of the connection path - `ulInstance`

For implicit connections, it is the configuration connection point.

For explicit connections, `ulInstance` is always 1.

Input connection point - `ulOTConnPoints`

The assembly instance, i.e. the connection point, to which the connection was directed, in O→T direction.

Output connection point – `ulTOConnPoints`

The assembly instance, i.e. the connection point, to which the connection was directed, in T→O direction.

Connection serial number – `usConnSerialNum`

The Serial Number of the connection affected by the state change, as specified by the Originator of the connection (and accepted by the protocol stack) when the connection was established. This 16-bit value uniquely identifies the connection amongst all connections with the device. For more details, see *"The CIP Networks Library, volume 1"*, section 3-5.5.1.5.

Originator vendor Id – `usVendorId`

The Vendor ID of the connection affected by the state change, as specified by the Originator of the connection (and accepted by the protocol stack) when the connection was established.

Originator serial number – `ulOSerialNum`

The Serial Number of the Originator of the connection affected by the state change, as specified when the connection was established.

Priority/tick time – `bPriority`

The Priority and Tick Time field of the connection affected by the state change, as specified in the Forward Open message with which the connection was opened. The actual Time per Tick is calculated as 2^tick_time [ms]. Refer to *"The CIP Networks Library, volume 1"*, section 3-5.4.1.2.1

Bits 5-7	Bit 4	Bits 3-0
Reserved	Priority 0: Normal 1: reserved	Tick Time

Table 103. Meaning of Variable bPriority

Time Out Ticks Parameter – `bTimeOutTicks`

The Time Out Ticks (Transaction Timeout for Opening the Connection in multiples of Ticks) of the connection affected by the state change, as specified in the Forward Open message which opened the connection.

Timeout multiplier - `bTimeoutMultiple`

The timeout multiplier of the connection affected by the state change, as specified in the Forward Open message with which the connection was opened. The actual connection timeout value (Inactivity Timeout) is calculated by multiplying the connection's RPI value (requested packet interval) with the connection's timeout multiplier. If no messages are received over the connection for this interval, it is closed due to a timeout condition.

The multiplier is numerically encoded according to the following table:

Code	Corresponding multiplier
0	x4
1	x8
2	x16
3	x32
4	x64
5	x128
6	x256
7	x512
8 - 255	Reserved

Table 104. Coding of timeout multiplier values

Transport/trigger – `bTriggerType`

The trigger type of the connection affected by the state change, as specified in the Forward Open message with which the connection as opened. It encodes the trigger condition for transmissions in T→O connection and the connection's transport class.

Bit 7	Bits 4-6	Bits 3-0
Direction 1 – server 0 – client	Trigger 0 – cyclic 1 – change of state 2 – application triggered	Connection class 0 – class 0 1 – class 1 2 – class 2 3 – class 3

Table 105. Meaning of variable bTriggerType

OT connection ID – `ulOTConnID`

The connection ID in O→T direction as selected by the originator of the connection.

TO connection ID – `ulTOConnID`

The connection ID in T→O direction as selected by the protocol stack when processing the Forward Open request of the connection whose state changed.

OT requested packet Interval- `ulOTRpi`

The requested packet interval (RPI) of the connection affected by the state change in O→T direction, as specified in the Forward Open message with which the connection was opened, in units of microseconds.

OT connection parameter - `usOTConnParam`

The O→T (consumer) connection parameters field of the connection affected by the state change in O→T direction, as specified in the Forward Open message with which the connection was opened. Table Meaning of variable usOTConnParam shows the structure and contents of this bit field.

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bits 8-0
Redundant owner	Connection type		Reserved	Priority		Fixed/variable	Reserved

Table 106. Meaning of variable usOTConnParam

The values have the following meaning

- Fixed/Variable
  This bit indicates whether the connection, in this direction, will accept frames with a variable size or requires all frames to have the fixed connection's size.
  If the bit is set, I/O frames may be smaller than the connection size. Otherwise, the protocol stack will ignore I/O frames with smaller size than the connection size.
- Priority

  Table Priority specifies the connection's priority encoding within bits 11-10 of the connection parameters:

Bit 11	Bit 10	Priority
0	0	Low priority
0	1	High priority
1	0	Scheduled
1	1	Urgent

Table 107. Priority

- Connection type

  Table Connection type specifies the connection type encoding within bits 14-13 of the connection parameters:

Bit 14	Bit 13	Connection type
0	0	Null – connection may be reconfigured
0	1	Multicast
1	0	Point-to-point connection
1	1	Reserved

Table 108. Connection type

NOTE Connection type *„Multicast"* is only supported for connections with CIP transport class 0 and class 1 and in T→O direction
Connection type *„Null"* is not supported. The stack will reject all such connections right anyway and thus, no Connection State Change indication will ever be generated for that connection type. We mention it for convenience and for the future extendibility of the implementation.

- Redundant owner
  The redundant owner bit will be set if more than one owner of the connection is allowed (Bit 15 = 1). If bit 15 is equal to zero, then the connection is an exclusive owner connection.

  NOTE The EtherNet/IP stack does not support redundant owner connections.

OT connection size - `usOTConnSize`

The O→T Connection Size of the connection affected by the state change, as specified (in number of bytes) in the "Forward Open message" which opened the connection.

This size may be smaller or equal to the size of the consuming connection point at which the connection directs.

TO requested packet interval - `ulTORpi`

The requested packet interval (RPI) of the connection affected by the state change in T→O direction, as specified (in units of microseconds) in the "Forward Open message" which opened the connection.

TO connection parameter - `usTOConnParam`

Similarly to `usOTConnParam`, the producer connection parameters for the connection (T→O direction).

TO connection size - `usTOConnSize`

The O→T connection size of the connection affected by the state change, as specified (in number of bytes) in the "Forward Open message" which opened the connection.

This size may be smaller than or equal to the size of the producing connection point at which the connection directs.

Production inhibit time - `ulProInhib`

The production inhibit time of the connection affected by the state change, as specified (in units of milliseconds) in the "Forward Open message" which opened the connection. A value of zero disables the production inhibit timer.

Extended state – `ulExtendedState`

The extended state provides additional information on the cause of the connection state change. This value is significant only if `ulConnectionState` equals `EIP_UNCONNECT`. Table Extended State specifies the possible values of this field.

Value of `ulExtendedState`	Numerical value	Meaning
If (ulConnectionState == EIP_UNCONNECT)		
`EIP_CONN_STATE_UNDEFINED`	0	No extended state available
`EIP_CONN_STATE_TIMEOUT`	1	Connection closed due to timeout condition
If (ulConnectionState == EIP_CONNECT)		
`EIP_CONN_STATE_UNDEFINED`	0	No extended state available

Table 109. Extended State

Indication packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	124	`EIP_OBJECT_CONNECTION_IND` – Packet data length in bytes
`ulSta`	uint32_t	0	Status not in used for indication.
`ulCmd`	uint32_t	0x1A2E	EIP_OBJECT_CONNECTION_IND
`tData (EIP_OBJECT_CONNECTION_IND_T)`			
`ulConnectionState`	uint32_t	0, 1	Reason of changing the connection state Connection established (1) Connection disconnected (0)

Variable	Type	Value/Range	Description
`usNumExclusiveOwner`	uint16_t		Number of established exclusive owner connections (adapter role)
`usNumInputOnly`	uint16_t		Number of established input only connections (adapter role)
`usNumListenOnly`	uint16_t		Number of established listen only connections (adapter role)
`usNumExplicitMessaging`	uint16_t		Number of established explicit connections (adapter role)
`usNumImplicitMessagingOriginator`	uint16_t		Number of class 0/1 connections currently opened by us (scanner role)
`usNumExplicitMessagingOriginator`	uint16_t		Number of class 3 connections currently opened by us (scanner role). This field is currently unused and always set to zero.
`bConnType`	uint8_t	1-16	Connection type
`abReserved[3]`	uint8_t	0	Reserved. Always set to 0.
`tConfigPath.ulClass`	uint32_t		Connection configuration path: Class ID If no configuration path is addressed or the addressed configuration path is to be ignored because no configuration data was given, this field is set to zero.
`tConfigPath.ulInstance`	uint32_t		Connection configuration path: Instance ID If no configuration path is addressed or the addressed configuration path is to be ignored because no configuration data was given, this field is set to zero.
`tConfigPath.ulConnPoint`	uint32_t		Connection configuration path: Connection Point ID If no configuration path is addressed or the addressed configuration path is to be ignored because no configuration data was given, this field is set to zero. For the assembly object, instance IDs and connection points are used synonymously.
`tConfigPath.ulAttribute`	uint32_t		Connection configuration path: Attribute ID If no configuration path is addressed or the addressed configuration path is to be ignored because no configuration data was given, this field is set to zero. For the configuration paths towards the assembly assembly object, always three.
`tConfigPath.ulMember`	uint32_t	0	Connection configuration path: Member ID Always zero
`tConsumptionPath.ulClass`	uint32_t		Connection consumption path: Class ID
`tConsumptionPath.ulInstance`	uint32_t		Connection consumption path: Instance ID
`tConsumptionPath.ulConnPoint`	uint32_t		Connection consumption path: Connection Point ID For the assembly object, instance IDs and connection points are used synonymously.
`tConsumptionPath.ulAttribute`	uint32_t		Connection consumption path: Attribute ID
`tConsumptionPath.ulMember`	uint32_t	0	Connection consumption path: Member ID Always zero
`tProductionPath.ulClass`	uint32_t		Connection production path: Class ID
`tProductionPath.ulInstance`	uint32_t		Connection production path: Instance ID

Variable	Type	Value/Range	Description
tProductionPath.ulConnPoint	uint32_t		Connection production path: Connection Point ID For the assembly object, instance IDs and connection points are used synonymously.
tProductionPath.ulAttribute	uint32_t		Connection production path: Attribute ID
tProductionPath.ulMember	uint32_t	0	Connection production path: Member ID Always zero
usConnSerialNum	uint16_t		Serial number of the connection
usVendorId	uint16_t		Originator vendor id
ulOSerialNum	uint32_t		Originator serial number
bPriority	uint8_t		Priority/Tick Time
bTimeOutTicks	uint8_t		Message timeout
bTimeoutMultiple	uint8_t		Time out multiplier
bTriggerType	uint8_t		Class/Trigger type
ulOTConnID	uint32_t		O→T Connection ID
ulTOConnID	uint32_t		T→O ConnectionID
ulOTRpi	uint32_t		O→T requested packet interval
usOTConnParam	uint16_t		O→T Connection parameter
usOTConnSize	uint16_t		O→T data size
ulTORpi	uint32_t		T→O requested packet interval
usTOConnParam	uint16_t		T→O Connection parameter
usTOConnSize	uint16_t		T→O data size
ulProInhib	uint32_t		Production inhibit time
ulExtendedState	uint32_t		0: No extended status 1: Connection timeout

Table 110. EIP_OBJECT_PACKET_CONNECTION_IND_T – Indication of connection

Response packet description

Variable	Type	Value/Range	Description
ulDest	uint32_t		Destination. Use value from indication
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	0	Status not used for response
ulCmd	uint32_t	0x00001A2F	command / response

Table 111. EIP_OBJECT_PACKET_CONNECTION_RES_T – Response to indication of connection

4.3.4 Configuration Assemblies

Configuration assemblies (`EIP_AS_TYPE_CONFIG`) hold a fixed or variable size configuration data byte sequence of given size. The configuration data structure and content is specific to the application. For instance, it could be used for calibrating sensors or actors prior to actual data transfer.

Therefore, the host application has to set the initial default configuration data into the config assembly after creation. When a new connection is opened towards a configuration assembly and that connection request contains correct length configuration data, then the firmware will compare this new data against the currently active configuration data. It will generate an EIP_OBJECT_CL3_SERVICE_IND, if an actual change in configuration data took place, thus presenting that new configuration data to the host application.

If the host replies to the EIP_OBJECT_CL3_SERVICE_IND with a success status, the firmware will copy-in the new data into the config assembly without any further action required by the host application. The I/O connection eventually will be opened and will bind the particular configuration assembly.

Note that the assembly flag `EIP_AS_OPTION_FIXED_SIZE` can be set on configuration assemblies so that the new data needs to match the exact length of the configuration assembly. Otherwise, also smaller sizes are accepted.

4.3.5 NULL ForwardOpen

A NULL ForwardOpen is a ForwardOpen request received on the network which has the TransportType NULL in both directions, O2T and T2O. A NULL ForwardOpen will thus not open an actual I/O connection for data transport. Instead, it serves one of the three following purposes:

- Ping a Device: A NULL ForwardOpen addressing the Identity Object instance 1, e.g. a request path of {0x20, 0x01, 0x24 0x01}, implements a ping mechanism at the CIP protocol level. The request is replied to with a ForwardOpen response without any further effect in the device. Thus, we will not describe this any further in this manual.
- Set initial configuration data for the application: A NULL ForwardOpen which specifies a connection serial number not matching any existing I/O connection provides initial configuration data for a config assembly, i.e. whilst this configuration assembly is not yet addressed by any other I/O connection. A data segment and a valid configuration application path must be contained. We further refer to this as a non-matching NULL ForwardOpen.
- Set (re)configuration data for the application on-the-fly: A NULL ForwardOpen which specifies a connection serial number matching an existing I/O connection provides configuration data for a config assembly on-the-fly, i.e. whilst this configuration assembly is already addressed by this particular I/O connection. A data segment and a valid configuration application path must be contained. We further refer to this as a matching NULL ForwardOpen.

Per default, NULL ForwardOpen support in the firmware is disabled. It has to be enabled explictly by the host application if there is a demand for the described features. Refer to service EIP_OBJECT_SET_PARAMETER_REQ for details.

NOTE If a firmware is configured for NULL ForwardOpen support, the EDS File and the Conformance Test Configuration File have to be adapted accordingly.

NOTE NULL ForwardOpen requests will also be subject to ForwardOpen forwarding, if enabled, as described in section EIP_OBJECT_LFWD_OPEN_FWD_IND.

4.3.5.1 Non-matching NULL ForwardOpen

A non-matching NULL ForwardOpen allows a CIP client to provide initial configuration data towards the application, i.e. before the addressed configuration assembly is bound to any I/O connection. The configuration data contained in the request, addressing an existing configuration assembly, is presented to the host application by means of the indication EIP_OBJECT_CL3_SERVICE_IND, command code `Set_Attribute_Single`, and, if accepted, will be stored into the configuration assembly. This configuration data then overwrites any configuration data previously set into the configuration assembly. The NULL ForwardOpen request will be replied to with an appropriate status code.

If the addressed configuration assembly is already bound by any I/O connection, the NULL ForwardOpen request will be rejected.

4.3.5.2 Matching NULL ForwardOpen

A matching NULL ForwardOpen allows the originator of a particular existing I/O connection to provide (re)configuration data towards the application, i.e. whilst the addressed configuration assembly may already be bound by that connection. This allows on-the-fly reconfiguration of the application by the originator of that connection. The configuration data contained in the request, addressing an existing configuration assembly, is presented to the host application by means of the indication EIP_OBJECT_CL3_SERVICE_IND, command code `Set_Attribute_Single`, and, if accepted, will be stored into the configuration assembly. This configuration data then overwrites any configuration data previously set into the configuration assembly. The NULL ForwardOpen request will be replied to with an appropriate status code.

If the addressed configuration assembly is already bound in another I/O connection but the matching connection, the NULL ForwardOpen request will be rejected. If the addressed configuration assembly is not bound yet, it will be bound by the matching connection.

NOTE If the configuration data to be set due to a NULL ForwardOpen, no matter if it is matching or non-matching, is not effective, i.e. equals the already contained data of the configuration assembly, no indication will be generated and the request will be replied to with success status right away.

4.3.6 Acyclic Data Transfer indication

The indication EIP_OBJECT_CL3_SERVICE_IND indicates an acyclic service request from the network. Typical situations in which the EtherNet/IP stack generates that indication are:

- An additional object class has been registered using the command EIP_OBJECT_MR_REGISTER_REQ and an Explicit Request is issued toward that object.
- An additional service has been registered for an existing object using EIP_OBJECT_REGISTER_SERVICE_REQ and that service is issued toward that object.
- Configuration data is set into a config assembly due to an opening I/O connection

The following parameters are provided with the indication:

- The CIP service code `bService`
- The CIP object class ID `ulObject`, Instance ID `ulInstance`, Attribute ID `ulAttribute` and Member ID `ulMember` addressed by the service

  NOTE Typically, CIP services received from the network rarely include the member ID just because most of the existing CIP attributes do not have members. In that case, the parameter `ulMember` is set to 0.

- A byte array containing the request data received with the service request, which the host application has to interpret, verify and process.
- The sequence count field of the frame which contained the service request, in case it was received over a class 3 connection (connected explicit)

The parameters Service Code, Class ID, Instance ID, Attribute ID and Member ID corresponds to the normal CIP Addressing. These fields are used for the most common services that use the addressing format “Service → Class → Instance → Attribute → Member”.

In case the service uses another format, the path information is put into the data part (`abData[]`) of this packet.

The data segment `abData[]` is only present for service requests which contained request data (e.g. the `Set_Attribute_Single` service). The `ulLen` field of the packet header can be evaluated to determine the request data size in number of bytes contained in `abData[]`:

```
`service_data_size = tHead.ulLen - EIP_OBJECT_CL3_SERVICE_IND_SIZE`
```

CIP services are divided into different address ranges. The subsequent Table Specified ranges of numeric values of service codes (variable bService) gives an overview. This table is taken from the CIP specification (*“Common Industrial Protocol specification, volume 1, section 4, table 4-9.6”*, see reference [5]).

Range of numeric value of service code (variable `bService`)	**Meaning**
0x00-0x31	Open. The services associated with this range of service codes are referred to as *Common Services*. These are defined in Appendix A of the CIP Networks Library, volume 1 (reference [5]).
0x32-0x4A	Range for service codes for vendor specific services
0x4B-0x63	Range for service codes for object class specific services
0x64-0x7F	Reserved by ODVA for future use
0x80-0xFF	Reserved for use as reply service code (see message router response format in section 2 of reference [6])

Table 112. Specified ranges of numeric values of service codes (variable bService)

NOTE It is specific to each object which services it implements. For object class IDs in the open range, refer to the CIP specification about the requirements to be fulfilled.
If you use object class IDs from the vendor-specific range, it is in your own responsibility to define and implement the set of available services.

Table Service Codes for the Common Services according to the CIP specification lists the service codes for the common services. This table is taken from the CIP specification (“Volume 1 Common Industrial Protocol Specification, section 5, table 5-1.1”, see reference [5]).

Service code (numeric value of bService)	Service to be executed
00	Reserved
01	Get_Attributes_All
02	Set_Attributes_All
03	Get_Attribute_List
04	Set_Attribute_List
05	Reset
06	Start
07	Stop
08	Create
09	Delete
0A	Multiple_Service_Packet
0B	Reserved for future use
0D	Apply_Attributes
0E	Get_Attribute_Single
0F	Reserved for future use
10	Set_Attribute_Single
11	Find_Next_Object_Instance
12	-13 Reserved for future use
14	Error Response (used by DevNet only)
15	Restore
16	Save
17	No Operation (NOP)
18	Get_Member
19	Set_Member
1A	Insert_Member
1B	Remove_Member
1C	GroupSync
1D-31	Reserved for additional Common Services

Table 113. Service Codes for the Common Services according to the CIP specification

After the host application has verified all provided parameters and the request data; it processes the service and finally, sends the response to this indication back to protocol stack.

Therefore, it sets the General Status code `ulGRC` and optionally also the Extended Status Code `ulERC` to an appropriate CIP status code.

The Generic Error Code indicates whether the service was successful in the first place, whereas the Extended Status Code may be set to provide additional diagnostic information. For the stack definitions of generic status codes, see Table CIP Generic Status Codes Definitions (Variable `ulGRC`).

If processing the service succeeded, additional reply data can be sent in the `abData` field of the response message. The response data size in number of bytes must be set in addition to the basic packet length (`EIP_OBJECT_CL3_SERVICE_RES_SIZE`) in the `ulLen` field of the response packet's header.

Figure Sequence Diagram for the EIP_OBJECT_CL3_SERVICE_IND/RES Packet for the Extended Packet Set displays a sequence diagram for the EIP_OBJECT_CL3_SERVICE_IND packet (see Configuration using the packet API).

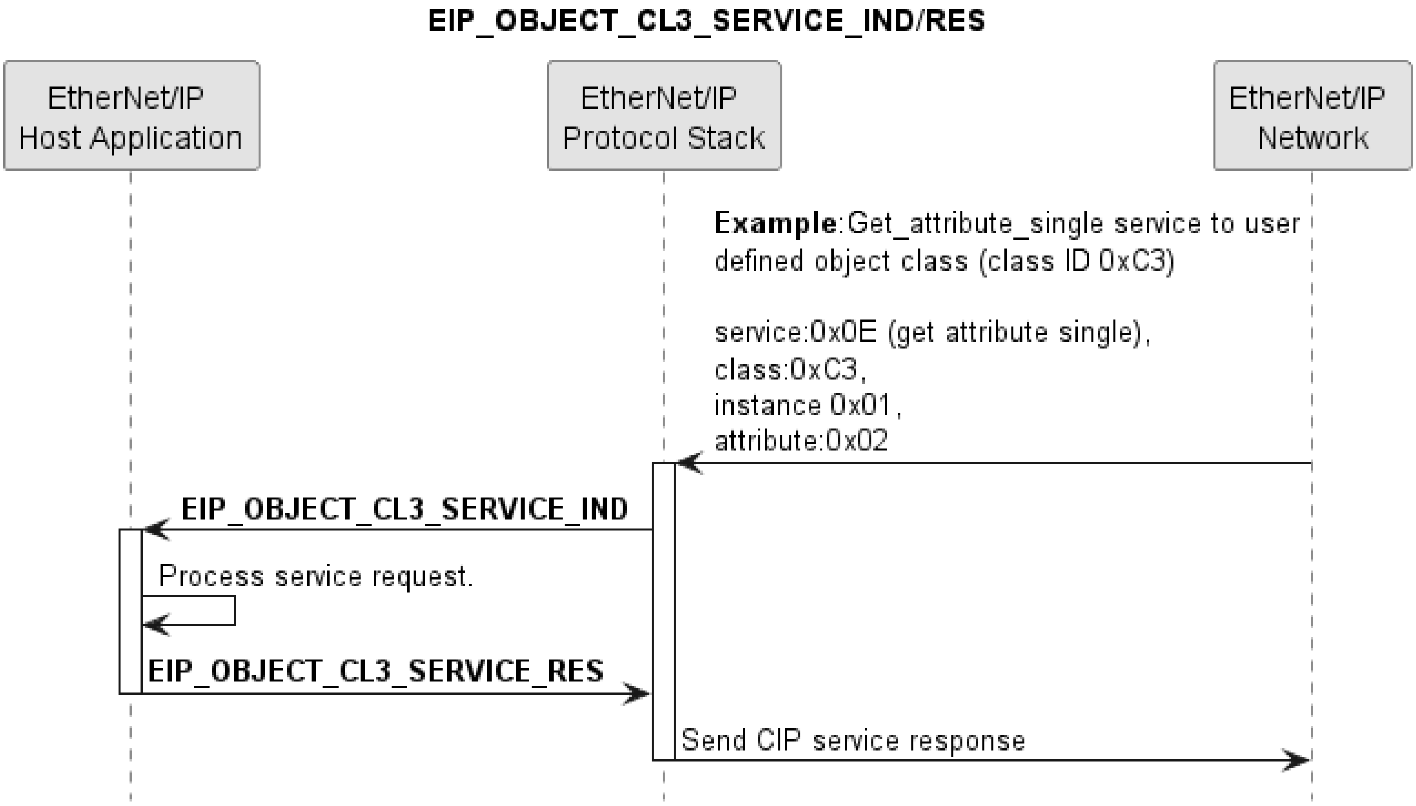

EIP_OBJECT_CL3_SERVICE_IND/RES
EtherNet/IP Host Application
EtherNet/IP Protocol Stack
EtherNet/IP Network
Example: Get_attribute_single service to user defined object class (class ID 0xC3)
service:0x0E (get attribute single),
class:0xC3,
instance 0x01,
attribute:0x02
EIP_OBJECT_CL3_SERVICE_IND
Process service request.
EIP_OBJECT_CL3_SERVICE_RES
Send CIP service response

Figure 15. Sequence Diagram for the EIP_OBJECT_CL3_SERVICE_IND/RES Packet for the Extended Packet Set

Indication packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	24 + n	Packet data length in bytes n = Length of service data area `abData`
`ulSta`	uint32_t	0	Status not in use for indication.
`ulCmd`	uint32_t	0x1A3E	EIP_OBJECT_CL3_SERVICE_IND
tData (EIP_OBJECT_CL3_SERVICE_IND_T)			
`ulConnectionId`	uint32_t	0 … 2^{32}-1	Unique Id (ignore)
`bService`	uint8_t	1-0xFF	CIP service code
`abPad0[3]`	uint8_t	0	Padding. Set to zero.
`ulObject`	uint32_t	1-0xFFFFFFFF	CIP Class ID
`ulInstance`	uint32_t	0-0xFFFFFFFF	CIP Instance Number
`ulAttribute`	uint32_t	0-0xFFFFFFFF	CIP Attribute Number The attribute number is 0, if the service does not address a specific attribute but the whole instance.
`ulMember`	uint32	0-0x7FFFFFFF	CIP member number This parameter contains the member number of the object class instance attribute specified in `ulAttribute`. Note: Typically, CIP services received from the network rarely include the member ID just because most of the existing CIP attributes do not have members. In that case, the parameter `ulMember` is set to 0.
`abData[]`	uint8_t		n bytes of service data (depending on service) This may also contain path information for instance in case that the service does not address an object with the format Class / Instance / Attribute.

Table 114. EIP_OBJECT_PACKET_CL3_SERVICE_IND_T - Indication of acyclic data transfer

Response packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t		Destination. Use value from indication
`ulLen`	uint32_t	32 + n	Packet data length in bytes where n = Length of service data area
`ulSta`	uint32_t	0	Status not used for response. Error code can be set using .tData.ulGRC and .tData.ulERC (see below)
`ulCmd`	uint32_t	0x00001A3F	EIP_OBJECT_CL3_SERVICE_RES
tData (EIP_OBJECT_CL3_SERVICE_RES_T)			
`ulConnectionId`	uint32_t	0 … 2^{32}-1	Unique Id from the indication packet
`bService`	uint8_t	1-0xFF	CIP service code from the indication packet
`abPad0[3]`	uint8_t	0	Padding. Set to zero.
`ulObject`	uint32_t	1-0xFFFFFFFF	CIP Object from the indication packet
`ulInstance`	uint32_t	0-0xFFFFFFFF	CIP Instance from the indication packet
`ulAttribute`	uint32_t	0-0xFFFFFFFF	CIP Attribute from the indication packet
`ulMember`	uint32	0-0x7FFFFFFF	CIP Member from the indication packet
`ulGRC`	uint32_t		Generic Error Code
`ulERC`	uint32_t		Extended Error Code
`abData[]`	uint8_t		n bytes of service data (depending on service)

Table 115. EIP_OBJECT_PACKET_CL3_SERVICE_RES_T – Response to indication of acyclic data transfer

4.3.7 CIP Object Change indication

The indication EIP_OBJECT_CIP_OBJECT_CHANGE_IND indicates a change of an attribute value of a CIP object class or instance of one of the built-in CIP objects. Change indications are generated when the change in the attribute's value happened due to a Set-Attribute Service from the network or another external trigger. Only attributes that have the attribute option flag `CIP_FLG_TREAT_NOTIFY` set, generate change indications (compare to section EIP_OBJECT_ENABLE_ATTRIBUTE_NOTIFICATION_REQ).

The purpose of this indication is to inform the host application about the change in the attribute's value. With Object Change indications, the host application is given the possibility to validate the value, which has been requested as the new attribute's value over the network. If the host application decides to reject the value, it replies to the change indication with a non-zero General Status Code in the `ulSta` field of the response packet's header. This reply will trigger the service response including an appropriate CIP error code on the network. In case the host application accepts the attribute change, it replies to the change indication with `ulSta` set to zero. This will trigger the response service on the network and additionally will take over the new attribute value into the object.

NOTE Accepting a new attribute value might trigger a store remanent data indication being sent to the host application in case the attribute is part of the protocol stack's remanent data. In that scenario the response service on the network will be sent not before the store remanent data indication is replied to.

The timeout restriction for synchronous indications applies to the Object Change indication. For a description of this mechanism, see section Application compliance.

As an example, Figure Exemplary sequence diagram for the EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES packet sequence displays a sequence diagram for the EIP_OBJECT_CIP_OBJECT_CHANGE_IND packet in case the host application stores remanent data.

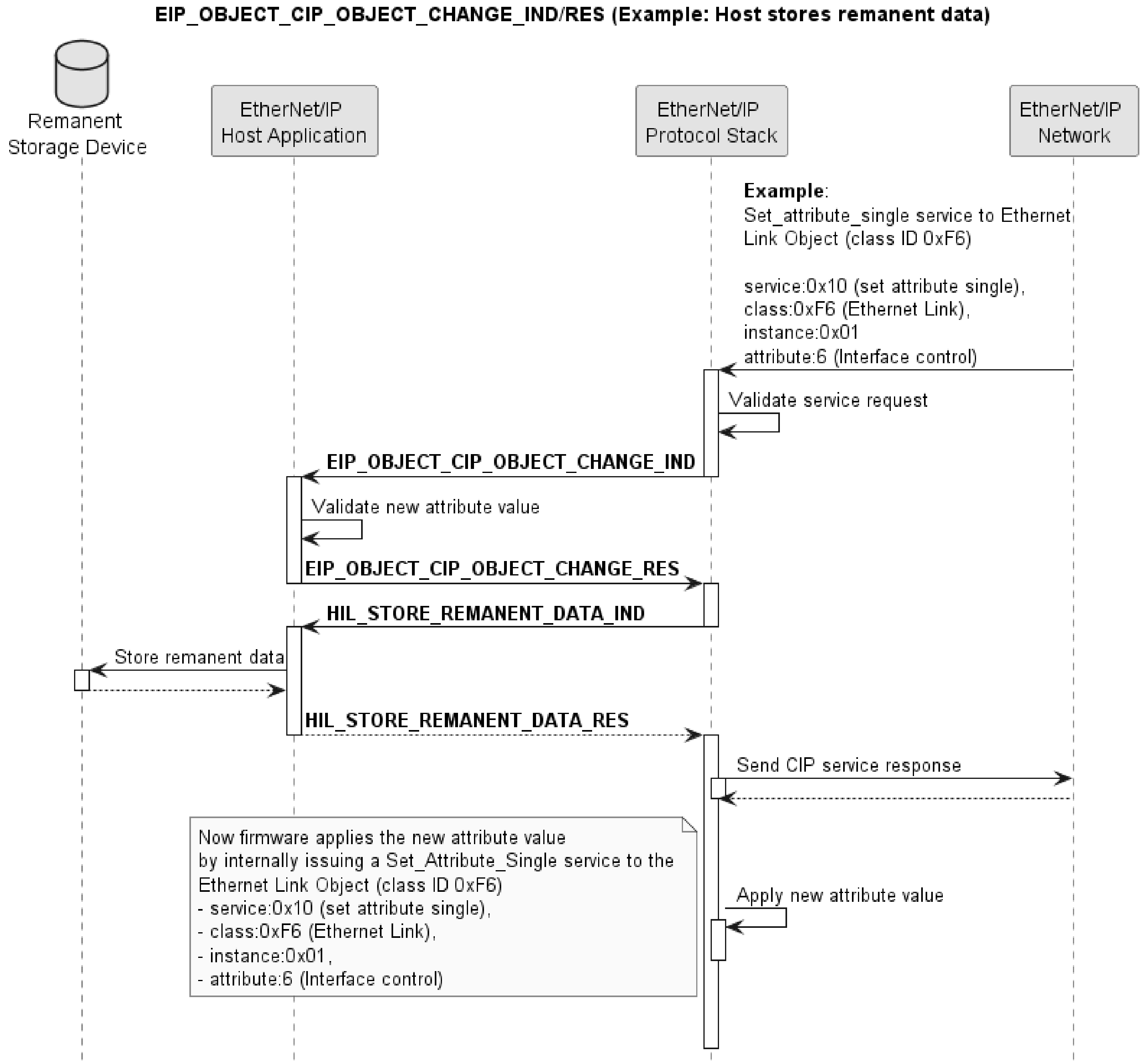

EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES (Example: Host stores remanent data)
Remanent Storage Device
EtherNet/IP Host Application
EtherNet/IP Protocol Stack
EtherNet/IP Network
Example:
Set_attribute_single service to Ethernet Link Object (class ID 0xF6)
service:0x10 (set attribute single),
class:0xF6 (Ethernet Link),
instance:0x01
attribute:6 (Interface control)
Validate service request
EIP_OBJECT_CIP_OBJECT_CHANGE_IND
Validate new attribute value
EIP_OBJECT_CIP_OBJECT_CHANGE_RES
HIL_STORE_REMANENT_DATA_IND
Store remanent data
HIL_STORE_REMANENT_DATA_RES
Send CIP service response
Now firmware applies the new attribute value by internally issuing a Set_Attribute_Single service to the Ethernet Link Object (class ID 0xF6)
- service:0x10 (set attribute single),
- class:0xF6 (Ethernet Link),
- instance:0x01,
- attribute:6 (Interface control)
Apply new attribute value

Figure 16. Exemplary sequence diagram for the EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES packet sequence

Indication packet description

Variable	Type	Value/Range	Description
ulLen	uint32_t	24+n	Packet data length in bytes n = Number of bytes in abData[]
ulSta	uint32_t	0	Status not in use for indication.
ulCmd	uint32_t	0x1AFA	EIP_OBJECT_CIP_OBJECT_CHANGE_IND
tData (EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T)			
ulInfoFlags	uint32_t	0x10 or 0x20	Flags specifying the type of change indication. This, for instance, informs the host whether or not it can reject the attribute change. Refer to the section Definition and purpose of parameter ulInfoFlags below for a comprehensive description.
bService	uint8_t	0x10	CIP service code Currently only the *SetAttributeSingle* service is used in this indication.
abPad0[3]	uint8_t	0	Padding. Set to zero
ulClass	uint32_t		CIP class ID
ulInstance	uint32_t		CIP instance number
ulAttribute	uint32_t		CIP attribute number
ulMember	uint32_t		CIP member number
abData[]	uint8_t		Attribute Data The number of bytes n provided in abData = tHead.ulLen - EIP_OBJECT_CIP_OBJECT_CHANGE_IND_SIZE

Table 116. EIP_OBJECT_PACKET_CIP_OBJECT_CHANGE_IND_T – CIP Object Change indication

Response packet description

Variable	Type	Value/Range	Description
ulDest	uint32_t		Destination. Use value from indication
ulLen	uint32_t	EIP_OBJECT_CIP_OBJECT_CHANGE_RES_SIZE plus the size of the payload data	Packet data length in bytes. Typically, the host application will not change this value, but pass it back as received with the object change indication.
ulSta	uint32_t		Currently, for this response packet, the Hilscher status codes in tHead.ulSta are discretely mapped to CIP general status codes. with the restriction that this mapping applies only for changes of the type EIP_OBJECT_CIP_OBJECT_CHANGE_IND_PROPOSE: 1. SUCCESS_HIL_OK (0x0) maps to CIP_GSR_SUCESSS (0x0) 'Success'. 2. ERR_HIL_OPERATION_NOT_POSSIBLE_IN_CURRENT_STATE (0xC000012DL) maps to CIP_GSR_DEV_IN_WRONG_STATE (0x10) 'Device State Conflict'. 3. Any other nonzero status code maps to CIP_GSR_SERVICE_ERROR (0x1E) 'Embedded Service Error'.
ulCmd	uint32_t	0x1AFB	EIP_OBJECT_CIP_OBJECT_CHANGE_RES
tData (EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T)			
ulInfoFlags	uint32_t	0x10 or 0x20	Flags specifying the type of change indication (from the indication packet)
bService	uint8_t	0x10	CIP service code (from the indication packet)
abPad0[3]	uint8_t	0	Padding (from the indication packet)
ulClass	uint32_t		CIP class ID (from the indication packet)
ulInstance	uint32_t		CIP instance number (from the indication packet)
ulAttribute	uint32_t		CIP attribute number (from the indication packet)
ulMember	uint32_t		CIP member number (from the indication packet)

Variable	Type	Value/Range	Description
abData[]	uint8_t		Attribute Data (from the indication packet)

Table 117. EIP_OBJECT_PACKET_CIP_OBJECT_CHANGE_RES_T – Response to CIP Object Change indication

4.3.7.1 Definition and purpose of parameter ulInfoFlags

Three values for the field ulInfoFlags in `EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T` are defined, which may be binary OR'd together:

Definition	Value	Description
EIP_OBJECT_CIP_OBJECT_CHANGE_IND_PROPOSE	0x10	The conditional change is proposed towards the host application, giving it the chance to reject the change.
EIP_OBJECT_CIP_OBJECT_CHANGE_IND_INFORM	0x20	The host application is informed that an unconditional change of the attribute value took place or is about to take place. The host application does not have any means to reject the change.
EIP_OBJECT_CIP_OBJECT_CHANGE_NV_STORING_BYPASSED	0x40	The changed attribute value normally is subject to remanent data storing, but the particular change has been omitted from the remanent storing. The prime example for this is the case where the device yields it's IP address due to a fresh DHCP cycle being started or the DHCP lease expiring. The IP address attribute (class 0xF5, instance 1, attribute 5) will then be set to zero until a new valid IP address has been obtained from the server. Since it is undesirable to have an (intermediate) IP of 0.0.0.0 in the remanent data, the storing is bypassed in such cases. See section DHCP/BOOTP Client for further information. Other situations where the IP configuration (temporarily) is set to 0.0.0.0, so that such an indication may be generated, are: 1. Link loss: The network cable is unplugged or the Ethernet Link is disabled 2. Network Interface reconfiguration due to changes in attribute 3 of the TCP/IP Interface object 3. A CIP Identity reset, when DHCP/ACD is freshly performed 4. A passive change by an external source, e.g. the Hilscher Ethernet Device Configuration Tool

Table 118. EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T - ulInfoFlags

4.3.8 Link Status Change

The indication HIL_LINK_STATUS_CHANGE_IND indicates a change in the Ethernet Link Status. This is informative for the application and has only to be evaluated if the host application implements a certain behavior on, e.g., link losses.

NOTE This indication is also sent directly after the host application has registered at the EtherNet/IP Stack (HIL_REGISTER_APP_REQ).

Indication packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	32	Packet data length in bytes
`ulSta`	uint32_t	0	Status not in use for indication.
`ulCmd`	uint32_t	0x2F8A	HIL_LINK_STATUS_CHANGE_IND
`tData (HIL_LINK_STATUS_CHANGE_IND_DATA_T)`			
`atLinkData[2]`	HIL_LINK_STATUS_T		Link status information for two ports. If only one port is available, ignore second entry.

Table 119. HIL_LINK_STATUS_CHANGE_IND_T - Link Status Change indication

`HIL_LINK_STATUS_T` is structured like this:

Variable	Type	Value/Range	Description
`ulPort`	uint32_t	0, 1	The port-number this information belongs to.
`fIsFullDuplex`	uint32_t	FALSE (0) TRUE	Is the established link full Duplex? Only valid if fIsLinkUp is TRUE.
`fIsLinkUp`	uint32_t	FALSE (0) TRUE	Is the link up for this port?
`ulSpeed`	uint32_t	0, 10 or 100	If the link is up, this field contains the speed of the established link. Possible values are 10 (10 MBit/s), 100 (100MBit/s) and 0 (no link).

Table 120. Structure HIL_LINK_STATUS_T

Response packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t		Destination. Use value from indication
`ulLen`	uint32_t	0	Packet data length in bytes. Depends on number of parameters
`ulSta`	uint32_t	0	Status not used for response.
`ulCmd`	uint32_t	0x2F8B	HIL_LINK_STATUS_CHANGE_RES

Table 121. HIL_LINK_STATUS_CHANGE_RES_T - Link Status Change response

4.3.9 Module Network Status Change

This packet indicates a change in either the module or network status. The LEDs of the device display the module status and the network status.

NOTE The change indication can be enabled by setting the flag `EIP_APS_PRM_SIGNAL_MS_NS_CHANGE` using the packet EIP_APS_SET_PARAMETER_REQ.

Indication packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	8	Packet data length in bytes
`ulSta`	uint32_t	0	Status not in use for indication.
`ulCmd`	uint32_t	0x360C	EIP_APS_MS_NS_CHANGE_IND
Data (EIP_APS_MS_NS_CHANGE_IND_T)			
`ulModuleStatus`	uint32_t	0 - 5	The module status describes the current state of the corresponding MS-LED (if it is connected). For details, see Module status.
`ulNetworkStatus`	uint32_t	0 - 5	The network status describes the current state of the corresponding NS-LED (if it is connected). For details, see Network status.

Table 122. EIP_APS_PACKET_MS_NS_CHANGE_IND_T – Module/Network Status Change indication

Response packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t	0	Status not in use for response.
`ulCmd`	uint32_t	0x360D	EIP_APS_MS_NS_CHANGE_RES

Table 123. EIP_APS_PACKET_MS_NS_CHANGE_RES_T - Link Status Change response

4.3.10 Forward_Open indication

NOTE The Forward Open Forwarding functionality can be enabled by setting the Parameter flag `EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING` using command EIP_OBJECT_SET_PARAMETER_REQ.

The indication EIP_OBJECT_LFWD_OPEN_FWD_IND indicates reception of a Forward_Open request on the EtherNet/IP network. A host application will only use the Forward Open Forwarding feature when it requires full control over those frames, i.e. adding application-specific behavior that the protocol stack does not implement. In the vast majority of EtherNet/IP applications, there is no need for the host application to implement costly direct handling of Forward Open frames. You are encouraged to consider carefully, whether you have a demand for this feature.

When Forward Open Forwarding is enabled, it is mandatory for the host application to correctly handle and reply to the indications:

- EIP_OBJECT_LFWD_OPEN_FWD_IND (Section EIP_APS_SET_PARAMETER_REQ)
- EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND
- EIP_OBJECT_FWD_CLOSE_FWD_IND

When Forward Open Forwarding is used, the protocol stack will pass every Forward Open Frame it receives on to the application without any previous processing. The host application has the possibility to modify the received Forward Open frame, return it in the response to the indication, `EIP_OBJECT_FWD_OPEN_FWD_RES`, and let the protocol stack continue its regular Forward Open processing on that modified frame just as if was received over the network. In their most basic variant, the Forward Open handlers would just return the received packet data for the protocol stack to process.

Typically, the host application would validate and/or modify the forward open request and let the stack continue processing that validated or modified request.

Furthermore, the application can attach “Application Reply Data” to the `EIP_OBJECT_LFWD_OPEN_FWD_RES` response message, which will be sent back to the originator on success. By setting a nonzero CIP status code in the response packet, the host application can effectively reject the opening or closing of a connection. As well, a non-zero 16-Bit extended status code can be set in the lower bits of `ulERC` in the response packet to provide further diagnostic information to the originator.

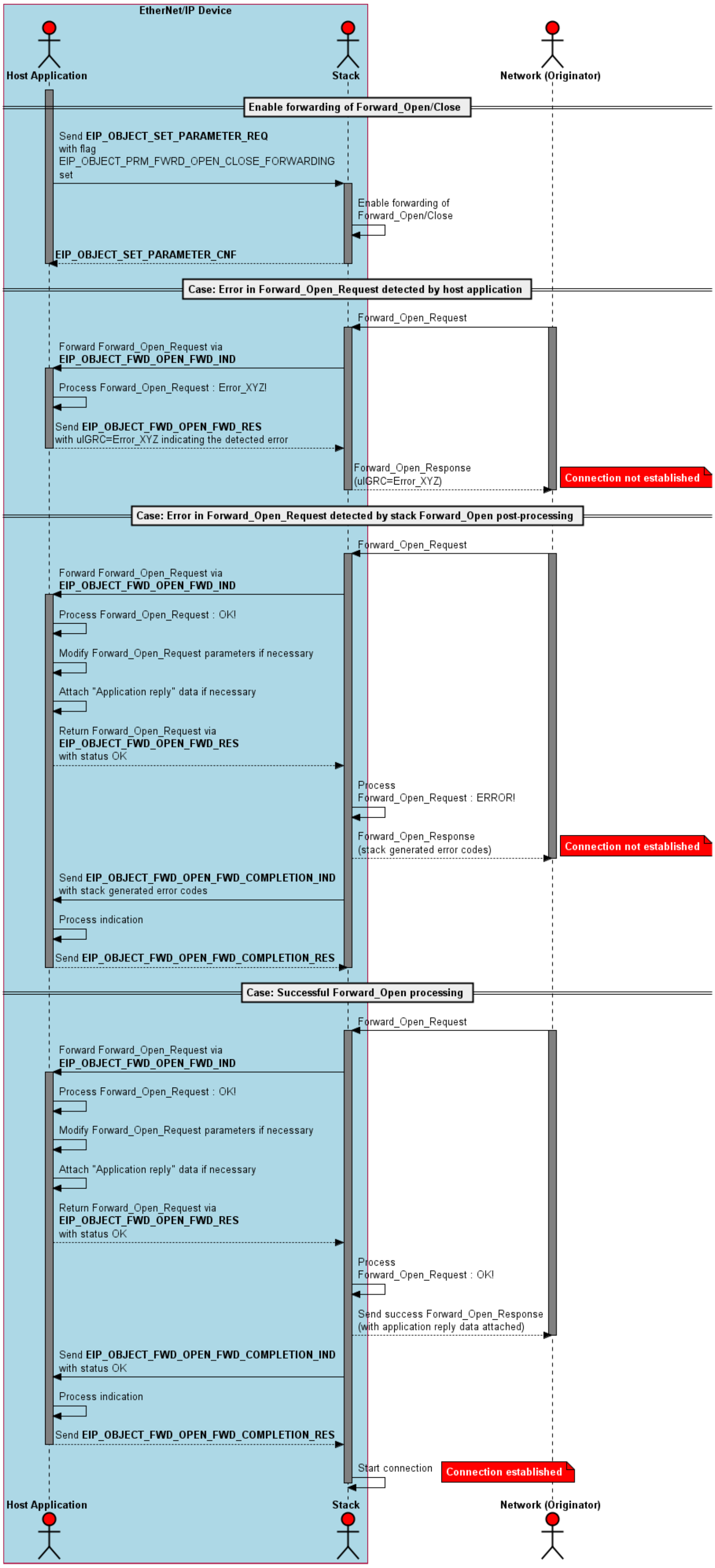
There are no restrictions regarding modification of the forward open, except for the maximum packet size and maximum path length.

When the protocol stack receives the host application’s response, `EIP_OBJECT_LFWD_OPEN_FWD_RES`, it will validate and process the Forward Open and then send the indication EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND to indicate the result of the Forward Open to the host application.

For an overview of the possible packet sequences, see following figure.

To attach “Application Reply Data”, add this data at the end of the connection path (`abConnPath`) field of the indication’s response and set `ulAppReplySize` and `ulAppReplyOffset` accordingly, as well as the packet’s data length `tHead.ulLen`. `ulAppReplySize` specifies the size of the application reply data in bytes and `ulAppReplyOffset` specifies the byte-offset of the application reply data within the data part of the `EIP_OBJECT_LFWD_OPEN_FWD_RES` packet.

EIP_OBJECT_FWD_OPEN_FWD_IND/RES
and
EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND/RES


EtherNet/IP Device
Host Application
Stack
Network (Originator)
Enable forwarding of Forward_Open/Close
Send EIP_OBJECT_SET_PARAMETER_REQ with flag EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING set
Enable forwarding of Forward_Open/Close
EIP_OBJECT_SET_PARAMETER_CNF
Case: Error in Forward_Open_Request detected by host application
Forward_Open_Request
Forward Forward_Open_Request via EIP_OBJECT_FWD_OPEN_FWD_IND
Process Forward_Open_Request : Error_XYZ!
Send EIP_OBJECT_FWD_OPEN_FWD_RES with ulGRC=Error_XYZ indicating the detected error
Forward_Open_Response (ulGRC=Error_XYZ)
Connection not established
Case: Error in Forward_Open_Request detected by stack Forward_Open post-processing
Forward_Open_Request
Forward Forward_Open_Request via EIP_OBJECT_FWD_OPEN_FWD_IND
Process Forward_Open_Request : OK!
Modify Forward_Open_Request parameters if necessary
Attach "Application reply" data if necessary
Return Forward_Open_Request via EIP_OBJECT_FWD_OPEN_FWD_RES with status OK
Process Forward_Open_Request : ERROR!
Forward_Open_Response (stack generated error codes)
Connection not established
Send EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND with stack generated error codes
Process indication
Send EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES
Case: Successful Forward_Open processing
Forward_Open_Request
Forward Forward_Open_Request via EIP_OBJECT_FWD_OPEN_FWD_IND
Process Forward_Open_Request : OK!
Modify Forward_Open_Request parameters if necessary
Attach "Application reply" data if necessary
Return Forward_Open_Request via EIP_OBJECT_FWD_OPEN_FWD_RES with status OK
Process Forward_Open_Request : OK!
Send success Forward_Open_Response (with application reply data attached)
Send EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND with status OK
Process indication
Send EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES
Start connection
Connection established
Host Application
Stack
Network (Originator)

Indication packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	60 + n	`EIP_OBJECT_LFWD_OPEN_FWD_IND_SIZE + n` - Packet data length in bytes n: Length of connection path (`abConnPath`) in bytes
`ulSta`	uint32_t	0	Status not in use for indication.
`ulCmd`	uint32_t	`0x1A60`	EIP_OBJECT_LFWD_OPEN_FWD_IND
`Data (EIP_OBJECT_LFWD_OPEN_FWD_IND_T)`			
`pRouteMsg`	uint32_t		Pointer to remember the underlying encapsulation request (must not be modified by app)
`aulReserved[4]`	uint32_t		Placeholder to be filled by response parameters, see `EIP_OBJECT_LFWD_OPEN_FWD_RES_T`
`tFwdOpenData`	EIP_LFWD_OPEN_DATA_T		Forward Open data (See Table EIP_LFWD_OPEN_DATA_T - Forward_Open request data)

Table 124. EIP_OBJECT_PACKET_LFWD_OPEN_FWD_IND_T – Forward_Open indication

The following Table EIP_LFWD_OPEN_DATA_T - Forward_Open request data explains the structure `EIP_LFWD_OPEN_DATA_T`:

Variable	Type	Description
`Structure EIP_LFWD_OPEN_DATA_T`		
`bPriority`	`uint8_t`	Used to calculate request timeout information
`bTimeOutTicks`	`uint8_t`	Used to calculate request timeout information
`ulOTConnID`	`uint32_t`	Network connection ID originator to target
`ulTOConnID`	`uint32_t`	Network connection ID target to originator
`usConnSerialNum`	`uint16_t`	Connection serial number
`usVendorId`	`uint16_t`	Originator Vendor ID
`ulOSerialNum`	`uint32_t`	Originator serial number
`bTimeoutMultiple`	`uint8_t`	Connection timeout multiplier
`abReserved1[3]`	`uint8_t`	Reserved
`ulOTRpi`	`uint32_t`	Originator to target requested packet rate in microseconds
`ulOTConnParam`	`uint32_t`	Originator to target connection parameter
`ulTORpi`	`uint32_t`	Target to originator requested packet rate in microseconds
`ulTOConnParam`	`uint32_t`	Target to originator connection parameter
`bTriggerType`	`uint8_t`	Transport type/trigger
`bConnPathSize`	`uint8_t`	Connection path size in 16 bit words
`abConnPath[]`	`uint8_t`	Connection path

Table 125. EIP_LFWD_OPEN_DATA_T - Forward_Open request data

For a detailed description on these parameters, see section EIP_OBJECT_CONNECTION_IND.

Response packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t		Destination. Use value from indication
`ulLen`	uint32_t	60 + n	`EIP_OBJECT_FWD_OPEN_FWD_RES_SIZE` + n - Packet data length in bytes n: Length of connection path (`abConnPath`) in bytes + Length of “Application Reply” data in `abConnPath`
`ulSta`	uint32_t	0	Status not in use for response. Error code can be set using .tData.ulGRC and .tData.ulERC (see below)
`ulCmd`	uint32_t	`0x1A61`	`EIP_OBJECT_LFWD_OPEN_FWD_RES`
`tData (EIP_OBJECT_LFWD_OPEN_FWD_RES_T)`			
`pRouteMsg`	void*		Pointer to underlying Encapsulation request
`ulGRC`	uint32_t		General Error Code, see Table CIP Generic Status Codes Definitions (Variable `ulGRC`)
`ulERC`	uint32_t		Extended Error Code
`ulAppReplyOffset`	uint32_t		Offset of “Application Reply” data.
`ulAppReplySize`	uint32_t		Length of “Application Reply” data in bytes. The “Application Reply” data can be attached by copying it right behind the connection path in `tFwdOpenData.abConnPath[]`
`tFwdOpenData`	EIP_LFWD_OPEN_DATA_T		Forward Open data (See Table EIP_LFWD_OPEN_DATA_T - Forward_Open request data)

Table 126. EIP_OBJECT_PACKET_LFWD_OPEN_FWD_RES_T – Response of Forward_Open indication

4.3.11 Forward_Open_Completion indication

NOTE This functionality must be enabled by setting the Parameter flag `EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING` using command EIP_OBJECT_SET_PARAMETER_REQ.

The indication EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND indicates to the host application the completion of a Forward Open request.

As stated in the preceding section, after reception of `EIP_OBJECT_FWD_OPEN_FWD_RES` and checking parameters and initializing corresponding resources, the protocol stack sends the indication EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND to give feedback to the host application whether the connection could be established or not.

For an overview of the possible packet sequences, see this figure.

Indication packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	18	Packet data length in bytes
`ulSta`	uint32_t	0	Status not in use for indication.
`ulCmd`	uint32_t	`0x1A4C`	EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND
`tData (EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_T)`			
`usReserved`	uint16_t	0	Deprecated and reserved for future use.
`usConnSerialNum`	uint16_t	0 - 255	Connection serial number
`usVendorId`	uint16_t		Originator Vendor ID
`ulOSerialNum`	uint32_t		Originator serial number
`ulGRC`	uint32_t		General Error Code, see Table CIP Generic Status Codes Definitions (Variable `ulGRC`)
`ulERC`	uint32_t		Extended Error Code

Table 127. EIP_OBJECT_PACKET_FWD_OPEN_FWD_COMPLETION_IND_T – Forward_Open Completion indication

For more information on the parameters `usConnSerialNum`, `usVendorId` and `ulOSerialNum`, see section EIP_OBJECT_CONNECTION_IND.

Response packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t		Destination. Use value from indication
`ulLen`	uint32_t	0	`EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES_SIZE` - Packet data length in bytes
`ulSta`	uint32_t	0	Status not in use for response.
`ulCmd`	uint32_t	`0x1A4D`	`EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES`

Table 128. EIP_OBJECT_PACKET_FWD_OPEN_FWD_COMPLETION_RES_T – Response of Forward_Open Completion indication

4.3.12 Forward_Close indication

NOTE To enable this functionality, set the Parameter flag `EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING` using command EIP_OBJECT_SET_PARAMETER_REQ.

The indication EIP_OBJECT_FWD_CLOSE_FWD_IND indicates reception of a Forward_Close request on the network. The protocol stack forwards the Forward_Close request without doing any processing on it. Only the parameters “Connection Serial Number”, “Originator Vendor ID” and “Originator Serial number” will be checked in advance. The host application now has the possibility to check/modify parameters within the Forward_Close request data. The host application also has the possibility to reject the Forward_Close request right away by setting the corresponding status field in the `EIP_OBJECT_FWD_CLOSE_FWD_RES` packet.

When the protocol stack receives the host application’s response, `EIP_OBJECT_FWD_CLOSE_FWD_RES`, it will validate and process the Forward Close just as if it came directly from the network. For an overview of the possible packet sequences, see following figure.

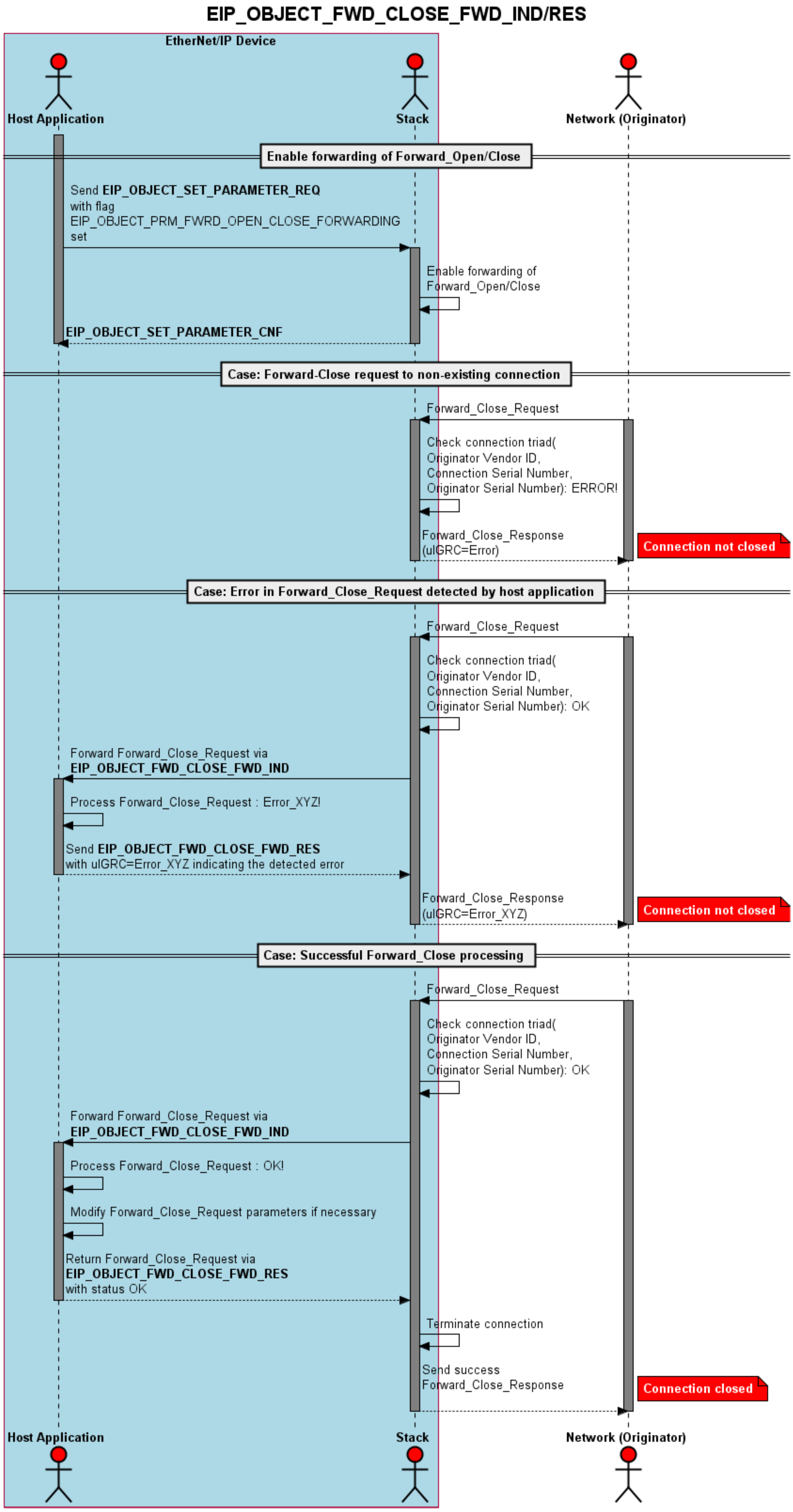

EIP_OBJECT_FWD_CLOSE_FWD_IND/RES
EtherNet/IP Device
Host Application
Stack
Network (Originator)
Enable forwarding of Forward_Open/Close
Send EIP_OBJECT_SET_PARAMETER_REQ with flag EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING set
Enable forwarding of Forward_Open/Close
EIP_OBJECT_SET_PARAMETER_CNF
Case: Forward-Close request to non-existing connection
Forward_Close_Request
Check connection triad(Originator Vendor ID, Connection Serial Number, Originator Serial Number): ERROR!
Forward_Close_Response (ulGRC=Error)
Connection not closed
Case: Error in Forward_Close_Request detected by host application
Forward_Close_Request
Check connection triad(Originator Vendor ID, Connection Serial Number, Originator Serial Number): OK
Forward Forward_Close_Request via EIP_OBJECT_FWD_CLOSE_FWD_IND
Process Forward_Close_Request : Error_XYZ!
Send EIP_OBJECT_FWD_CLOSE_FWD_RES with ulGRC=Error_XYZ indicating the detected error
Forward_Close_Response (ulGRC=Error_XYZ)
Connection not closed
Case: Successful Forward_Close processing
Forward_Close_Request
Check connection triad(Originator Vendor ID, Connection Serial Number, Originator Serial Number): OK
Forward Forward_Close_Request via EIP_OBJECT_FWD_CLOSE_FWD_IND
Process Forward_Close_Request : OK!
Modify Forward_Close_Request parameters if necessary
Return Forward_Close_Request via EIP_OBJECT_FWD_CLOSE_FWD_RES with status OK
Terminate connection
Send success Forward_Close_Response
Connection closed
Host Application
Stack
Network (Originator)

Figure 17. Packet sequence for Forward_Close forwarding functionality

Indication packet description

Variable	Type	Value/Range	Description
ulDest	uint32_t	0x20	Destination
ulLen	uint32_t	24 + n	EIP_OBJECT_FWD_CLOSE_FWD_IND_SIZE + n - Packet data length in bytes n: Length of connection path (abConnPath) in bytes
ulSta	uint32_t	0	Status not in use for indication.
ulCmd	uint32_t	0x1A4E	EIP_OBJECT_FWD_CLOSE_FWD_IND
tData (EIP_OBJECT_FWD_CLOSE_FWD_IND_T)			
ulRouteMsg	uint32_t		Pointer to remember underlying Encapsulation request (must not be modified by app)
aulReserved[2]	uint32_t		Place holder to be filled by response parameters, see EIP_OBJECT_FWD_CLOSE_FWD_RES_T
tFwdCloseData	EIP_CM_APP_FWCLOSE_IND_T		Forward Close data (See Table EIP_CM_APP_FWCLOSE_IND_T - Forward_Close request data)

Table 129. EIP_OBJECT_PACKET_FWD_CLOSE_FWD_IND_T – Forward_Close request indication

Variable	Type	Description
bPriority	uint8_t	Used to calculate request timeout information
bTimeOutTicks	uint8_t	Used to calculate request timeout information
usConnSerialNum	uint16_t	Connection serial number
usVendorId	uint16_t	Originator Vendor ID
ulOSerialNum	uint32_t	Originator serial number
bConnPathSize	uint8_t	Connection path size in 16 bit words
bReserved1	uint8_t	Reserved
abConnPath[]	uint8_t	Connection path

Table 130. EIP_CM_APP_FWCLOSE_IND_T - Forward_Close request data

Response packet description

Variable	Type	Value/Range	Description
ulDest	uint32_t		Destination. Use value from indication
ulLen	uint32_t	24 + n	EIP_OBJECT_FWD_CLOSE_FWD_RES_SIZE + n - Packet data length in bytes n: Length of connection path (abConnPath) in bytes
ulSta	uint32_t	0	Status not used for response
ulCmd	uint32_t	0x1A4F	EIP_OBJECT_FWD_CLOSE_FWD_RES
Data (EIP_OBJECT_FWD_CLOSE_FWD_RES_T)			
ulRouteMsg	uint32_t		Pointer to underlying Encapsulation request
ulGRC	uint32_t		General Error Code, see Table CIP Generic Status Codes Definitions (Variable ulGRC).
ulERC	uint32_t		Extended Error Code
tFwdCloseData	EIP_CM_APP_FWCLOSE_IND_T		Forward Close data (See Table EIP_CM_APP_FWCLOSE_IND_T - Forward_Close request data)

Table 131. EIP_OBJECT_PACKET_FWD_CLOSE_FWD_RES_T – Response of Forward_Close indication

4.3.13 Store Remanent Data indication

In case the application is responsible to store remanent data (section Remanent data), the application must handle this service. For a description of this service and the indication and response packet, see reference [9].

Value for ulComponentID

```
#define HIL_COMPONENT_ID_EIP_APS                     ((uint32_t)0x00590000L)
```

4.4 Additional services requested by the application

In this section, we describe the following set of additional services that the host application can use:

Packet	Command code (REQ/CNF or IND/RES)
EIP_APS_GET_MS_NS_REQ	0x0000360E
HIL_SET_WATCHDOG_TIME_REQ	0x00002F04
HIL_GET_WATCHDOG_TIME_REQ	0x00002F02
HIL_GET_DPM_IO_INFO_REQ	0x00002F0C
HIL_UNREGISTER_APP_REQ	0x00002F12
HIL_DELETE_CONFIG_REQ	0x00002F14
HIL_LOCK_UNLOCK_CONFIG_REQ	0x00002F32
HIL_FIRMWARE_IDENTIFY_REQ	0x00001EB6
GENAP_GET_COMPONENT_IDS_REQ	0x0000AD00
HIL_SET_REMANENT_DATA_REQ	0x00002F8C
HIL_SET_TRIGGER_TYPE_REQ	0x00002F90
HIL_GET_TRIGGER_TYPE_REQ	0x00002F92
EIP_OBJECT_FORCE_LED_STATE_REQ	0x00001A40
EIP_OBJECT_ENABLE_ATTRIBUTE_REQ	0x00001A10
EIP_OBJECT_SET_ATTRIBUTE_PERMISSION_REQ	0x00001A12
EIP_OBJECT_ENABLE_ATTRIBUTE_NOTIFICATION_REQ	0x00001A14
EIP_OBJECT_ENABLE_DISABLE_ATTRIBUTE_PROTECTION_REQ	0x00001A16

Table 132. Overview: Additional services of the EtherNet/IP Adapter

4.4.1 Get Module Status/ Network Status

The host application sends EIP_APS_GET_MS_NS_REQ to retrieve the current Module and Network Status of the netX device.

Table Possible values of the module status lists all possible values of the Module Status (Parameter `ulModuleStatus` of the confirmation packet) and their meaning.

Table Possible values of the network status lists all possible values of the Network Status (Parameter `ulNetworkStatus` of the confirmation packet) and their meaning.

Request packet description

Variable	Type	Value/Range	Description
ulDest	uint32_t	0x20	Destination
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t	0	-
ulCmd	uint32_t	0x360E	EIP_APS_GET_MS_NS_REQ

Table 133. EIP_APS_PACKET_GET_MS_NS_REQ_T – Get Module Status/ Network Status Request

Confirmation packet description

Variable	Type	Value/Range	Description
ulLen	uint32_t	EIP_APS_GET_MS_NS_CNF_SIZE	Packet data length in bytes
ulSta	uint32_t		See section Status/error codes
ulCmd	uint32_t	0x360F	EIP_APS_GET_MS_NS_CNF
tData (EIP_APS_GET_MS_NS_CNF_T)			
ulModuleStatus	uint32_t	0..5	Module Status The module status describes the current state of the corresponding MS-LED (if it is connected). See Table Possible values of the module status for more information.
ulNetworkStatus	uint32_t	0..5	Network Status The network status describes the current state of the corresponding NS-LED (if it is connected). See Table Possible values of the network status for more information.

Table 134. EIP_APS_PACKET_GET_MS_NS_CNF_T – Confirmation of Get Module Status / Network Status request

4.4.2 Set Watchdog Time

The host application sends HIL_SET_WATCHDOG_TIME_REQ to set the interval of the watchdog timer, in units of milliseconds.

This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [9].

4.4.3 Get Watchdog Time

The host application sends HIL_GET_WATCHDOG_TIME_REQ to retrieve the currently configured interval of the watchdog timer, in units of milliseconds.

This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [10].

4.4.4 Get DPM I/O Information

The host application sends HIL_GET_DPM_IO_INFO_REQ to obtain the offsets and lengths of the areas used within the DPM I/O blocks.

This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [9].

4.4.5 Delete Configuration

The host application sends HIL_DELETE_CONFIG_REQ to delete the internally stored configuration from RAM or FLASH. For the EtherNet/IP stack, this will remove all stored remanent data. Database files on the filesystem are not deleted.

This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [9].

> **NOTE** In case the host application stores remanent data, the sending of HIL_DELETE_CONFIG_REQ generates the indication packet HIL_STORE_REMANENT_DATA_IND. In this case, the HIL_DELETE_CONFIG_REQ will not be confirmed until that indication is replied to by the host application.

4.4.6 Lock/Unlock Configuration

The host application sends HIL_LOCK_UNLOCK_CONFIG_REQ to lock or unlock configuration data, respectively. A locked configuration cannot be altered.

This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [9].

4.4.7 Get Firmware Identification

The host application sends HIL_FIRMWARE_IDENTIFY_REQ to retrieve version information of the protocol stack firmware running on the netX, i.e. its name, version and date.

This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [9].

4.4.8 Get Component Information

The host application sends GENAP_GET_COMPONENT_IDS_REQ to retrieve information of the EtherNet/IP protocol stack, i.e. the component id, the remanent data size and version.

This is a generic packet, which is not specific to the EtherNet/IP protocol stack. Therefore, this document does not cover this packet. For details, refer to reference [9].

4.4.9 Set Remanent Data request

In case the application is responsible to store remanent data (section Remanent data), the application must use this service during startup to provide the remenent data to the firmware. For a description of this service and the indication and respone packet, see reference [9]. For a state diagram, see section Host application behavior.

Value for ulComponentID

```
#define HIL_COMPONENT_ID_EIP_APS                    ((uint32_t)0x00590000L)
```

4.4.10 Set Trigger Type

The host application sends packet HIL_SET_TRIGGER_TYPE_REQ to the stack to configure the data exchange trigger mode for the IO handshakes and Sync handshake.

The trigger mode defines the network-specific event for the protocol stack to finish the synchronization of the provider/consumer data update cycle or a pending synchronization request.

Consumer data (DPM input)

The protocol stack finishes the consumer data update cycle:

- Instantly (best-effort) in free-run mode: `HIL_TRIGGER_TYPE_*_NONE`
- In case eligible new input data is received: `HIL_TRIGGER_TYPE_*_RX_DATA_RECEIVED`

Provider data (DPM output)

The protocol stack finishes the provider data update cycle:

- Instantly (best-effort) in free-run mode: `HIL_TRIGGER_TYPE_*_NONE`

Synchronization

The protocol stack finishes the synchronization cycle:

- When a certain point in time is reached: `HIL_TRIGGER_TYPE_*_TIMED_ACTIVATION`

All trigger modes are functionally independent and can be used individually or combined. We recommend using a time-triggered or an event-triggered interface design, but not a combination of both.

The default trigger modes is free-run mode for consumer and provider data and disabled for the synchronization trigger mode.

Note that `HIL_TRIGGER_TYPE_*_RX_DATA_RECEIVED` is not meant to implement bus-cycle synchronous operation, but to provide input data with lower latency and application overhead (due to true event-based operation instead of polling).

For a more specific description of the handshake modes supported by the EtherNet/IP stack, see section Handshake modes.

Notes

- In case the protocol stack is configured with a trigger mode unequal to free-run, it is protocol-stack-specific at which point of time the synchronization or provider/consumer data update is finished. E.g. the protocol stack will wait for a network connection to be established.
- If supported, the protocol stack accepts the service in bus off mode. It is protocol-stack-specific if the service is accepted in bus on mode.
- On channel initialization, the protocol stack keeps the previously configured trigger mode until active change or device reset.
- The protocol stack monitors (for the configured data exchange mode) whether the host application handles the handshake as expected. Every time an error symptom occurs, the respective handshake error counter is incremented. The error counter counts up to the maximal possible value and saturates.
- In case the trigger mode is configured in default mode, the handshake error counters are set to 0 and do not count.
- The protocol stack resets the handshake error counter to the initial value (zero) after each channel init.

The application uses this request packet to modify the trigger mode of the protocol stack.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
`ulLen`	uint32_t	6	Packet data length in bytes
`ulCmd`	uint32_t	0x00002F90	HIL_SET_TRIGGER_TYPE_REQ
`tData (HIL_SET_TRIGGER_TYPE_REQ_DATA_T)`			
`usPdInHskTriggerType`	uint16_t	0x0010, 0x0011	The Input Handshake Trigger mode to be used.
`usPdOutHskTriggerType`	uint16_t	0x0010	The Output Handshake Trigger mode to be used.
`usSyncHskTriggerType`	uint16_t	0x0010, 0x0014	The Sync Handshake Trigger mode to be used.

Table 135. HIL_SET_TRIGGER_TYPE_REQ_T – Set Trigger Type request

The protocol stack will respond to the request with the following confirmation.

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x00002F91	HIL_SET_TRIGGER_TYPE_CNF

Table 136. HIL_SET_TRIGGER_TYPE_CNF_T – Set Trigger Type confirmation

4.4.11 Get Trigger type

The application can use this service to read the handshake trigger type currently configured in the protocol stack.

To do so, the host application sends the HIL_GET_TRIGGER_TYPE_REQ packet to retrieve

- the trigger mode (handshake behavior) for IO handshake and Sync handshake
- the fastest allowed DPM update time

of the protocol stack related to a specific DPM Communication Channel.

The protocol stack will respond to the request with the `HIL_GET_TRIGGER_TYPE_CNF` confirmation. The following table explains the variables returned within the confirmation packet.

Variable	Remarks
usPdInHskTriggerType	Input process data trigger type. Value is a type of `HIL_TRIGGER_TYPE_PDIN_*`. `HIL_TRIGGER_TYPE_PDIN_NONE` (0x0010) means no input data synchronization (free-run) `HIL_TRIGGER_TYPE_PDIN_RX_DATA_RECEIVED` (0x0011) means input data will be updated when new data is received. (bus cycle synchronous)
usPdOutHskTriggerType	Output process data trigger type. Value is a type of `HIL_TRIGGER_TYPE_PDOUT_*`. `HIL_TRIGGER_TYPE_PDIN_NONE` (0x0010) means no output data synchronization (free-run)
usSyncHskTriggerType	Synchronization trigger type Value is a type of `HIL_TRIGGER_TYPE_SYNC_*`. `HIL_TRIGGER_TYPE_PDIN_NONE` (0x0010) means no sync signal generation (free-run) `HIL_TRIGGER_TYPE_SYNC_TIMED_ACTIVATION` (0x0014) means generate Sync event when data shall be applied
usMinFreeRunUpdateInterval	Minimal provide/consumer data update interval in free-run mode. In free-run mode, the application has to ensure to request provider/consumer data updates not faster (i.e. more frequently) than this interval. The value is specified in units of microseconds, the default value is 1000 µs, values between 0 and 31 are not valid.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x00002F92	HIL_GET_TRIGGER_TYPE_REQ

Table 137. HIL_GET_TRIGGER_TYPE_REQ_T – Get Trigger Type request

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	8	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x00002F93	HIL_GET_TRIGGER_TYPE_CNF
`tData (HIL_GET_TRIGGER_TYPE_CNF_DATA_T)`			
`usPdInHskTriggerType`	uint16_t	0x0010, 0x0011	The Input Handshake Trigger mode currently used.
`usPdOutHskTriggerType`	uint16_t	0x0010	The Output Handshake Trigger mode currently used.
`usSyncHskTriggerType`	uint16_t	0x0010, 0x0014	The Sync Handshake Trigger mode currently used.
`usMinFreeRunUpdateInterval`	uint16_t	>=32	The fastest possible update time in case FreeRun mode is active (in microseconds).

Table 138. HIL_GET_TRIGGER_TYPE_CNF_T – Get Trigger Type confirmation

4.4.12 Force LED State service

The host application can send the EIP_OBJECT_FORCE_LED_STATE_REQ packet to force the COM0 (CIP Module Status) and/or COM1 (CIP Network Status) LEDs to a specific state.

> **NOTE** If this service is used the protocol stack no longer can control the Module and Network Status LEDs. Thus, the host application additionally must disable the "Flash LED" service of the Identity object which will not be functional while the LEDs are forced.
> This host application can enable/disable this service by using the packet EIP_OBJECT_SET_PARAMETER_REQ with flag `EIP_OBJECT_PRM_DISABLE_FLASH_LEDS_SERVICE` (see EIP_OBJECT_SET_PARAMETER_REQ).

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t	8	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x1A40	EIP_OBJECT_FORCE_LED_STATE_REQ
`tData (EIP_OBJECT_FORCE_LED_STATE_REQ_T)`			
`ulLedType`	uint32_t	0 - 1	Defines the LED that is affected 0: `CIP_LED_TYPE_MODULE_STATUS`, Module Status LED (COM0) 1: `CIP_LED_TYPE_NETWORK_STATUS`, Network Status LED (COM1)
`ulLedState`	uint32_t	0 - 7	0: `CIP_LED_STATE_NO_FORCING` stack will derive the LED state from the current device state (disable previous forcing of selected LED) 1: `CIP_LED_STATE_FORCE_OFF`, force selected LED off 2: `CIP_LED_STATE_FORCE_RED`, force selected LED red 3: `CIP_LED_STATE_FORCE_GREEN`, force selected LED green 4: `CIP_LED_STATE_FORCE_RED_FLASH_1HZ`, force selected LED red flashing with 1Hz interval 5: `CIP_LED_STATE_FORCE_GREEN_FLASH_1HZ`, force selected LED green flashing 1Hz interval 6: `CIP_LED_STATE_FORCE_RED_GREEN_FLASH_1HZ`, force selected LED red-green flashing with 1Hz (new for CIPSafety) 7: `CIP_LED_STATE_FORCE_RED_GREEN_FLASH_2HZ`, force selected LED red-green flashing with 2Hz (new for CIPSafety)

Table 139. EIP_OBJECT_PACKET_FORCE_LED_STATE_REQ_T – Force LED State request

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x1A41	`EIP_OBJECT_FORCE_LED_STATE_CNF`

Table 140. EIP_OBJECT_PACKET_FORCE_LED_STATE_CNF_T – Confirmation to Force LED State request

4.4.13 Enable Attribute service

Per default, certain attributes of particular built-in CIP objects are disabled and appear as if they are not implemented to both external CIP clients and the host application. However, the host application can enable these attributes at any time, typically during configuration of the EtherNet/IP stack. After a reset or power cycle, the attributes return to their default, i.e. disabled state. Explicitly disabling an attribute is not possible.

See section Available object classes to find out which attributes can be enabled.

The host application sends the following command to enable one of these attributes.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t	12	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x1A10	EIP_OBJECT_ENABLE_ATTRIBUTE_REQ
`tData (EIP_OBJECT_ENABLE_ATTRIBUTE_REQ_T)`			
`ulClass`	uint32_t		Class ID of the disabled attribute which shall be enabled
`ulInstance`	uint32_t		Instance ID of the disabled attribute which shall be enabled
`ulAttribute`	uint32_t		Attribute ID of the disabled attribute which shall be enabled

Table 141. EIP_OBJECT_PACKET_ENABLE_ATTRIBUTE_REQ_T – Enable attribute

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	1	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x1A11	`EIP_OBJECT_ENABLE_ATTRIBUTE_CNF`
`tData (EIP_OBJECT_ENABLE_ATTRIBUTE_CNF_T)`			
`bGrc`	uint8_t		CIP status code

Table 142. EIP_OBJECT_PACKET_ENABLE_ATTRIBUTE_CNF_T – Enable attribute confirmation

4.4.14 Set Attribute Permission service

Per default, some attributes of built-in CIP objects can be fully accessed only from the host application, but accessing them from the CIP network may be restricted to either get-access, i.e. read-only access, or to no access permission at all.

The host application can however grant the CIP network additional access rights or revoke (default) access rights for particular attributes by means of the following service.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t	14	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x1A12	EIP_OBJECT_SET_ATTRIBUTE_PERMISSION_REQ
tData (EIP_OBJECT_SET_ATTRIBUTE_PERMISSION_REQ_T)			
`ulClass`	uint32_t		Class ID of the attribute whose permission shall be modified
`ulInstance`	uint32_t		Instance ID of the attribute whose permission shall be modified
`ulAttribute`	uint32_t		Attribute ID of the attribute whose permission shall be modified
`fAllowBusSetAccess`	uint8_t		Grant the CIP network set access for the addressed attribute
`fAllowBusGetAccess`	uint8_t		Grant the CIP network get access for the addressed attribute

Table 143. EIP_OBJECT_PACKET_SET_ATTRIBUTE_PERMISSION_REQ_T – Set attribute permission

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	1	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x1A13	EIP_OBJECT_SET_ATTRIBUTE_PERMISSION_CNF
tData (EIP_OBJECT_SET_ATTRIBUTE_PERMISSION_CNF_T)			
`bGrc`	uint8_t		CIP status code

Table 144. EIP_OBJECT_PACKET_SET_ATTRIBUTE_PERMISSION_CNF_T – Set attribute permission confirmation

4.4.15 Enable Attribute Notification service

The attribute notification mechanism will cause the protocol stack to generate an indication packet EIP_OBJECT_CIP_OBJECT_CHANGE_IND to the host application with each attribute change for a defined set of CIP object attributes. If a particular attribute is flagged for notification, each modification of its value due to CIP network access will present such an indication packet to the host application, so that it can either verify and reject the new attribute value or at least take notice of the changed value, depending on the semantics of the particular attribute (inform vs. propose semantics).

Currently, the following attributes are subject to notification:

1. All attributes that are remanently stored as listed in section Remanent data content.
2. All attributes which have been enabled for notification by the host application by means of this service.

There is no means to explicitly disable attribute notifications once enabled except for resetting the protocol stack. Subsequent to this command, changes of the addressed attribute will be notified by means of EIP_OBJECT_CIP_OBJECT_CHANGE_IND.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t	12	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x1A14	EIP_OBJECT_ENABLE_ATTRIBUTE_NOTIFICATION_REQ
`tData (EIP_OBJECT_ENABLE_ATTRIBUTE_NOTIFICATION_REQ_T)`			
`ulClass`	uint32_t		Class ID of the attribute for which to enable notification
`ulInstance`	uint32_t		Instance ID of the attribute for which to enable notification
`ulAttribute`	uint32_t		Attribute ID of the attribute for which to enable notification

Table 145. EIP_OBJECT_PACKET_ENABLE_ATTRIBUTE_NOTIFICATION_REQ_T – Enable attribute notification

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	1	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x1A15	EIP_OBJECT_ENABLE_ATTRIBUTE_NOTIFICATION_CNF
`tData (EIP_OBJECT_ENABLE_ATTRIBUTE_NOTIFICATION_CNF_T)`			
`bGrc`	uint8_t		CIP status code

Table 146. EIP_OBJECT_PACKET_ENABLE_ATTRIBUTE_NOTIFICATION_CNF_T– Enable attribute notification confirmation

4.4.16 Enable/Disable Attribute Protection service

As described in section CIP device protection, the default protection policy comprises a certain set of attributes. This set can be modified by enabling attribute protection for further attributes. Also, attributes can dynamically be removed from the protection policy. Therefore, the host application sends the following service.

> **NOTE** Certain attributes possess specific semantics in relation to the protection policy. Take, for example, attribute 3 of a configuration assembly object. If this attribute is included in the protection policy, it will no longer be accessible for set access via CIP explicit messaging. However, it can still be modified through implicit access, such as when configuration data is set through a valid ForwardOpen. Given that the host application is responsible for configuration data processing, it can also adhere to the device protection for this scenario.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t	13	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x1A16	EIP_OBJECT_ENABLE_DISABLE_ATTRIBUTE_PROTECTION_REQ
`tData (EIP_OBJECT_ENABLE_DISABLE_ATTRIBUTE_PROTECTION_REQ_T)`			
`ulClass`	uint32_t		Class ID of the attribute for which to enable/disable protection
`ulInstance`	uint32_t		Instance ID of the attribute for which to enable/disable protection
`ulAttribute`	uint32_t		Attribute ID of the attribute for which to enable/disable protection
`fProtected`	uint8_t		Set attribute protected/unprotected

Table 147. EIP_OBJECT_PACKET_ENABLE_DISABLE_ATTRIBUTE_PROTECTION_REQ_T – Enable/Disable attribute protection

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	1	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x1A17	EIP_OBJECT_ENABLE_DISABLE_ATTRIBUTE_PROTECTION_CNF
`tData (EIP_OBJECT_ENABLE_DISABLE_ATTRIBUTE_PROTECTION_CNF_T)`			
`bGrc`	uint8_t		CIP status code

Table 148. EIP_OBJECT_PACKET_ENABLE_DISABLE_ATTRIBUTE_PROTECTION_CNF_T – Enable/Disable attribute protection confirmation

4.4.17 Terminate CIP Connection service

The host application sends EIP_OBJECT_TERMINATE_CONNECTION_REQ to terminate a CIP connection immediately. The service is intended for CIP-Safety use cases, and non-CIP-Safety applications should have no need to use this service, since it should have no demand to shutdown a connection actively, which would be unexpected for any originator or observer in non-safety applications.

The service will terminate the CIP connection uniquely addressed by the given connection triad, which is the connection serial number, the originator vendor ID, and the originator serial number. The connection triad used for addressing corresponds to the connection triad that is presented in the EIP_OBJECT_CONNECTION_IND_T packet on each connection state change.

The service will abruptly abort the CIP connection, and return the confirmation as soon as the connection is fully closed and all associated resources are available again.

Termination of an Exclusive Owner connection will also close all ListenOnly connections that use the terminated exclusive owner connection as the controlling connection.

Active connection termination through the service will still send one or multiple EIP_OBJECT_CONNECTION_IND to inform the host application about the connection state change(s). The application will likely implement a bookkeeping of the connections and states.

The originating TCP connection, if still present, will remain open if a connection is terminated.

With this service, we specifically address CIP-Safety use cases. For instance, a specific connection may need to be terminated if the time coordinated messaging of a safety connection fails. In other scenarios, all connections must be terminated, e.g. the CIP class 1 Safety Open.

Since EtherNet/IP provides no active closing from the target towards the originator, an observer will only recognize the stopped data consumption and production, thus may not be able to distinguish the terminate connection situation from an ordinary timeout condition. The application-side CIP Safety layers are aware of this process though, by adding additional context to the affected CIP connections.

Request packet description

Variable	Type	Value/Range	Description
`ulDest`	uint32_t	0x20	Destination
`ulLen`	uint32_t	8	Packet data length in bytes
`ulSta`	uint32_t	0	See section Status/error codes
`ulCmd`	uint32_t	0x1A18	EIP_OBJECT_TERMINATE_CONNECTION_REQ
`tData (EIP_OBJECT_TERMINATE_CONNECTION_REQ_T)`			
`usConnectionSerialNumber`	uint32_t		Connection serial number.
`usVendorId`	uint32_t		Originator vendor ID.
`ulOriginatorSerialNum`	uint32_t		Originator serial number.

Table 149. EIP_OBJECT_PACKET_TERMINATE_CONNECTION_REQ_T – Terminate connection request

Confirmation packet description

Variable	Type	Value/Range	Description
`ulLen`	uint32_t	0	Packet data length in bytes
`ulSta`	uint32_t		See section Status/error codes
`ulCmd`	uint32_t	0x1A19	`EIP_OBJECT_TERMINATE_CONNECTION_CNF_T`

Table 150. EIP_OBJECT_PACKET_TERMINATE_CONNECTION_CNF_T – Terminate connection confirmation

Noteworthy failure cases of the terminate connection service are:

- ERR_HIL_NO_MORE_RESOURCES: Resource temporarily unavailable.
- ERR_EIP_SERVICE_RUNNING: Connection is currently being closed by other means.
- ERR_EIP_OBJECT_UNKNOWN_CONNECTION: The specified connection does not exist.

Chapter 5 Resource and feature configuration via tag list

Modification of the firmware’s taglist allows controlling of certain behavior, features and resource limits. The Hilscher Tag List Editor software should be used for modifiying the firmware’s taglist.

Please find the supported firmware tags in the corresponding firmware datasheet [11].

Chapter 6 Status/error codes

6.1 Stack-specific error codes

Hexadecimal Value	Definition Description
0x00000000	SUCCESS_HIL_OK Status ok
0xC01F0001	ERR_EIP_OBJECT_COMMAND_INVALID Invalid command received.
0xC01F0002	ERR_EIP_OBJECT_OUT_OF_MEMORY System is out of memory
0xC01F0003	ERR_EIP_OBJECT_OUT_OF_PACKETS Task runs out of empty packets at the local packet pool
0xC01F0004	ERR_EIP_OBJECT_SEND_PACKET Sending a packet failed
0xC01F0010	ERR_EIP_OBJECT_AS_ALLREADY_EXIST Assembly instance already exists
0xC01F0011	ERR_EIP_OBJECT_AS_INVALID_INST Invalid assembly instance
0xC01F0012	ERR_EIP_OBJECT_AS_INVALID_LEN Invalid assembly length
0xC01F0020	ERR_EIP_OBJECT_CONN_OVERRUN No free connection buffer available
0xC01F0021	ERR_EIP_OBJECT_INVALID_CLASS Object class is invalid
0xC01F0022	ERR_EIP_OBJECT_SEGMENT_FAULT Segment of the path is invalid
0xC01F0023	ERR_EIP_OBJECT_CLASS_ALLREADY_EXIST Object class is already used
0xC01F0024	ERR_EIP_OBJECT_CONNECTION_FAIL Connection failed.
0xC01F0025	ERR_EIP_OBJECT_CONNECTION_PARAM Unknown format of connection parameter
0xC01F0026	ERR_EIP_OBJECT_UNKNOWN_CONNECTION Invalid connection ID
0xC01F0027	ERR_EIP_OBJECT_NO_OBJ_RESSOURCE No resource for creating a new class object available
0xC01F0028	ERR_EIP_OBJECT_ID_INVALID_PARAMETER Invalid request parameter
0xC01F0029	ERR_EIP_OBJECT_CONNECTION_FAILED General connection failure. For details, see General Error Code and Extended Error Code.
0xC01F0031	ERR_EIP_OBJECT_READONLY_INST Access denied. Instance is read only
0xC01F0032	ERR_EIP_OBJECT_DPM_USED DPM address is already used by another instance.
0xC01F0033	ERR_EIP_OBJECT_SET_OUTPUT_RUNNING Set Output command is already running
0xC01F0034	ERR_EIP_OBJECT_TASK_RESETING EtherNet/IP Object Task is running a reset.
0xC01F0035	ERR_EIP_OBJECT_SERVICE_ALLREADY_EXIST Object Service already exists

Hexadecimal Value	Definition Description
0xC01F0036	ERR_EIP_OBJECT_DUPLICATE_SERVICE The service is rejected by the application due to a duplicate sequence count.
0xC01F0037	ERR_EIP_TIMER_INVALID_HANDLE Timer function is called with invalid timer handle.
0xC01F0038	ERR_EIP_INVALID_STACK_MODE Setting the operation mode is called with an undefined mode value.
0xC01F0039	ERR_EIP_OUT_OF_ASSEMBLIES No assembly instances free to open a connection.
0xC01F003A	ERR_EIP_CALLBACK_REQUIERED Function needs callback to provide result data.
0xC01F003B	ERR_EIP_SERVICE_NOT_SUPPORTED This service is at the actual configuration not supported.
0xC01F003C	ERR_EIP_SERVICE_RUNNING This service is running and cannot be started twice.
0xC01F003D	EIP_ERR_CC_DATA_IMAGE_ERROR The address of the data is not at the range of the data image.
0xC01F003E	EIP_ERR_CC_UNKNOWN_FORMAT The format of the data mapping is unknown.
0xC01F003F	ERR_EIP_CONNECTION_POINT_CREATE Creating the connection point failed.

Table 151. Status/Error Codes of EtherNet/IP objects

Hexadecimal Value	Definition Description
0xC0590001	ERR_EIP_APS_COMMAND_INVALID Invalid command received.
0xC0590002	ERR_EIP_APS_PACKET_LENGTH_INVALID Invalid packet length.
0xC0590003	ERR_EIP_APS_PACKET_PARAMETER_INVALID Parameter of the packet are invalid.
0xC0590004	ERR_EIP_APS_TCP_CONFIG_FAIL Configuration of TCP/IP failed.
0xC0590005	ERR_EIP_APS_CONNECTION_CLOSED Existing connection is closed.
0xC0590006	ERR_EIP_APS_ALREADY_REGISTERED An application is already registered.
0xC0590007	ERR_EIP_APS_ACCESS_FAIL Command is not allowed.
0xC0590008	ERR_EIP_APS_STATE_FAIL Command not allowed at this state.
0xC0590009	ERR_EIP_APS_IO_OFFSET_INVALID Invalid offset for I/O data.
0xC059000A	ERR_EIP_APS_FOLDER_NOT_FOUND Folder for database not found.
0xC059000B	ERR_EIP_APS_CONFIG_DBM_INVALID Configuration database invalid.
0xC059000C	ERR_EIP_APS_NO_CONFIG_DBM Configuration database not found.
0xC059000D	ERR_EIP_APS_NWID_DBM_INVALID Network database invalid.
0xC059000E	ERR_EIP_APS_NO_NWID_DBM Network database not found.
0xC059000F	ERR_EIP_APS_NO_DBM No database found.
0xC0590010	ERR_EIP_APS_NO_MAC_ADDRESS_AVAILABLE No MAC address available.
0xC0590011	ERR_EIP_APS_INVALID_FILESYSTEM Access to file system failed.
0xC0590012	ERR_EIP_APS_NUM_AS_INSTANCE_EXCEEDS Maximum number of assembly instances exceeds.
0xC0590013	ERR_EIP_APS_CONFIGBYDATABASE Stack is already configured via database.

Table 152. Status/Error Codes of EtherNet/IP application task

Hexadecimal Value	Definition Description
0xC0950001	ERR_EIP_DLR_COMMAND_INVALID Invalid command received.
0xC0950002	ERR_EIP_DLR_NOT_INITIALIZED DLR task is not initialized.
0xC0950003	ERR_EIP_DLR_FNC_API_INVALID_HANDLE Invalid DLR handle at API function call.
0xC0950004	ERR_EIP_DLR_INVALID_ATTRIBUTE Invalid DLR object attribute.
0xC0950005	ERR_EIP_DLR_INVALID_PORT Invalid port.
0xC0950006	ERR_EIP_DLR_LINK_DOWN Port link is down.
0xC0950007	ERR_EIP_DLR_MAX_NUM_OF_TASK_INST_EXCEEDED Maximum number of EthernetIP task instances exceeded.
0xC0950008	ERR_EIP_DLR_INVALID_TASK_INST Invalid task instance.
0xC0950009	ERR_EIP_DLR_CALLBACK_NOT_REGISTERED Callback function is not registered.
0xC095000A	ERR_EIP_DLR_WRONG_DLR_STATE Wrong DLR state.
0xC095000B	ERR_EIP_DLR_NOT_CONFIGURED_AS_SUPERVISOR Not configured as supervisor.
0xC095000C	ERR_EIP_DLR_INVALID_CONFIG_PARAM Configuration parameter is invalid.
0xC095000D	ERR_EIP_DLR_NO_STARTUP_PARAM_AVAIL No startup parameters available.

Table 153. Status/Error Codes of EtherNet/IP DLR task

6.2 General EtherNet/IP error codes

The following table contains the possible General Error Codes defined within the CIP specification [1], Appendix B.

General Status Code (specified hexadecimally)	Status Name	Description
00	Success	The service has successfully been performed by the specified object.
01	Connection failure	A connection-elated service failed. This happened at any location along the connection path.
02	Resource unavailable	Some resources which were required for the object to perform the requested service were not available.
03	Invalid parameter value	See status code 0x20, which is usually applied in this situation.
04	Path segment error	A path segment error has been encountered. Evaluation of the supplied path information failed.
05	Path destination unknown	The path references an unknown object class, instance or structure element causing the abort of path processing.
06	Partial transfer	Only a part of the expected data could be transferred.
07	Connection lost	The connection for messaging has been lost.
08	Service not supported	The requested service has not been implemented or has not been defined for this object class or instance.
09	Invalid attribute value	Detection of invalid attribute data
0A	Attribute list error	An attribute in the Get_Attribute_List or Set_Attribute_List response has a status not equal to 0.
0B	Already in requested mode/state	The object is already in the mode or state which has been requested by the service
0C	Object state conflict	The object is not able to perform the requested service in the current mode or state
0D	Object already exists	It has been tried to create an instance of an object which already exists.
0E	Attribute not settable	It has been tried to change a non-modifiable attribute.
0F	Privilege violation	A check of permissions or privileges failed.
10	Device state conflict	The current mode or state of the device prevents the execution of the requested service.
11	Reply data too large	The data to be transmitted in the response buffer requires more space than the size of the allocated response buffer
12	Fragmentation of a primitive value	The service specified an operation that is going to fragment a primitive data value, i.e. half a REAL data type.
13	Not enough data	The service did not supply all required data to perform the specified operation.
14	Attribute not supported	An unsupported attribute has been specified in the request
15	Too much data	More data than was expected were supplied by the service.
16	Object does not exist	The specified object does not exist in the device.
17	Service fragmentation sequence not in progress	Fragmentation sequence for this service is not currently active for this data.
18	No stored attribute data	The attribute data of this object has not been saved prior to the requested service.
19	Store operation failure	The attribute data of this object could not be saved due to a failure during the storage attempt.
1A	Routing failure, request packet too large	The service request packet was too large for transmission on a network in the path to the destination. The routing device was forced to abort the service.
1B	Routing failure, response packet too large	The service response packet was too large for transmission on a network in the path from the destination. The routing device was forced to abort the service.
1C	Missing attribute list entry data	The service did not supply an attribute in a list of attributes that was needed by the service to perform the requested behavior.

General Status Code (specified hexadecimally)	Status Name	Description
1D	Invalid attribute value list	The service returns the list of attributes containing status information for invalid attributes.
1E	Embedded service error	An embedded service caused an error.
1F	Vendor-specific error	A vendor specific error has occurred. This error should only occur when none of the other general error codes can correctly be applied.
20	Invalid parameter	A parameter which was associated with the request was invalid. The parameter does not meet the requirements of the CIP specification and/or the requirements defined in the specification of an application object.
21	Write-once value or medium already written	An attempt was made to write to a write-once medium for the second time, or to modify a value that cannot be changed after being established once.
22	Invalid reply received	An invalid reply is received. Possible causes can for instance be among others a reply service code not matching the request service code or a reply message shorter than the expectable minimum size.
23-24	Reserved	Reserved for future extension of CIP standard
25	Key failure in path	The key segment (i.e. the first segment in the path) does not match the destination module. More information about which part of the key check failed can be derived from the object specific status.
26	Path size Invalid	Path cannot be routed to an object due to lacking information or too much routing data have been included.
27	Unexpected attribute in list	It has been attempted to set an attribute which may not be set in the current situation.
28	Invalid member ID	The Member ID specified in the request is not available within the specified class/ instance or attribute
29	Member cannot be set	A request to modify a member which cannot be modified has occurred
2A	Group 2 only server general failure	This DeviceNet-specific error cannot occur in EtherNet/IP
2B-CF	Reserved	Reserved for future extension of CIP standard
D0-FF	Reserved for object class and service errors	An object class specific error has occurred.

Table 154. General Error Codes according to CIP Standard

Chapter 7 Appendix

7.1 Legal notes

Copyright

Important notes

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

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

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

Liability disclaimer

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

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

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

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

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

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

Warranty

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

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

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

Costs of support, maintenance, customization and product care

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

Additional guarantees

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

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

Confidentiality

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

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

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

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

Export provisions

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

7.2 Third party software license

lwIP IP stack

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

Copyright (c) 2001-2004 Swedish Institute of Computer Science.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

4. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
5. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
6. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

7.3 Contacts

Headquarters

Germany
Hilscher Gesellschaft für Systemautomation mbH
Rheinstrasse 15
65795 Hattersheim
Phone: +49 (0) 6190 9907-0
Fax: +49 (0) 6190 9907-50
E-Mail: info@hilscher.com

Support
Phone: +49 (0) 6190 9907-99
E-Mail: de.support@hilscher.com

Subsidiaries

China
Hilscher Systemautomation (Shanghai) Co. Ltd.
200010 Shanghai
Phone: +86 (0) 21-6355-5161
E-Mail: info@hilscher.cn
Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

France
Hilscher France S.a.r.l.
69800 Saint Priest
Phone: +33 (0) 4 72 37 98 40
E-Mail: info@hilscher.fr
Support
Phone: +33 (0) 4 72 37 98 40
E-Mail: fr.support@hilscher.com

India
Hilscher India Pvt. Ltd.
Pune, Delhi, Mumbai
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