Protocol API

EtherNet/IP Adapter

V3.6.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC150401API07EN | Revision 7 | English | 2021-02 | Released | Public

Table of Contents

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 through the dual-port memory 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 List of Revisions

Rev	Date	Name	Revisions
7	2021-02-22	MBO	Initial for firmware Version V3.6.0.0: merged changes from EISV 5.2 manual
		KMI	Removed description of HIL_GET/SET_FW_PARAMETER_REQ

Table 1: List of Revisions

1.3 System Requirements

This software package has following system requirements to its environment:

- netX-Chip as CPU hardware platform

1.4 Intended Audience

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

- Knowledge of the netX DPM Interface manual
- Knowledge of the Common Industrial Protocol (CIP™) Specification Volume 1
- Knowledge of the Common Industrial Protocol (CIP™) Specification Volume 2

1.5 Specifications

The data below applies to the EtherNet/IP Adapter firmware and stack version V3.6.0.

This firmware/stack meets a subset of the requirements outlined in the CIP specification Volumes 1 and 2 as referenced by [7] and [8].

Technical Data

Feature	Value
Maximum number of input data	504 bytes per assembly instance
Maximum number of output data	504 bytes per assembly instance
Maximum number of assembly instances	10
IO Connection Types (implicit)	Exclusive Owner Listen Only Input only
IO Connection Trigger Types	Cyclic, minimum 1 ms* Application Triggered, minimum 1 ms* Change Of State, minimum 1 ms* * Depending on the number of parallel connections and sizes of input and output data.
Explicit Messages	Connected and unconnected
Unconnected Message Manager (UCMM)	Supported
Max. number of connections	Class 1: 5 Class 3: 8 UCMM: 8
Predefined standard objects	Identity Object (0x01) Message Router Object (0x02) Assembly Object (0x04) Connection Manager (0x06) DLR Object (0x47) QoS Object (0x48) TCP/IP Interface Object (0xF5) Ethernet Link Object (0xF6)
Maximum number of user-specific objects	20
Supported features	TCP/IP, UDP/IP DHCP BOOTP Device Level Ring (DLR) - Media Redundancy Address Conflict Detection (ACD) Quality of Service CIP Reset services - Identity Object Reset Service of Type 0 and 1
Ethernet interface	10 and 100 MBit/s Integrated switch
Duplex modes	Half Duplex, Full Duplex, Auto-Negotiation
MDI modes	MDI, MDI-X, Auto-MDIX
Data transport layer	Ethernet II, IEEE 802.3

Table 2: Technical data

Firmware/stack available for netX

netX 50	yes
netX 51	yes
netX 52	yes
netX 100, netX 500	yes
netX 90	no
netX 4000	no

Configuration

- Configuration by packets
- Configuration by tool e.g. Communication Studio (Download or exported configuration of two files named `config.nxd` and `nwid.nxd`)
- Configuration by packets

Diagnostic

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

Limitations

- Tags are not supported
  Note: “Tags” is a common mechanism to address typed PLC data using string identifiers.
- Connection type “Null Forward Open” is not supported
- CIP Motion is not supported
- CIP Safety is not supported

1.6 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
AS	Assembly Object
BOOTP	Boot Protocol
CIP	Common Industrial Protocol
CM	Connection Manager
DHCP	Dynamic Host Configuration Protocol
DiffServ	Differentiated Services
DLR	Device Level Ring (i.e. ring topology on device level)
DPM	Dual Port Memory
EIM	Ethernet/IP Scanner (= **M**aster)
EIP	Ethernet/IP
EIS	Ethernet/IP Adapter (= **S**lave)
ENCAP	Encapsulation Layer
ERC	Extended Error Code
GRC	Generic Error Code
IANA	Internet Assigned Numbers Authority
ID	Identity Object
IP	Internet Protocol
LSB	Least Significant Byte
MR	Message Router Object
MSB	Most Significant Byte
ODVA	Open DeviceNet Vendors Association
OSI	Open Systems Interconnection (according to ISO 7498)
QoS	Quality of Service
RPI	Requested Packet Interval
TCP	Transmission Control Protocol
UCMM	Unconnected Message Manager
VLAN	Virtual Local Area Network

Table 3: Terms, Abbreviations and Definitions

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

1.7 Input and Output Data Conventions

EtherNet/IP and the netX use different naming schemes in certain cases. To avoid problems with that this section shall clarify 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).

1.8 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: Packet API, netX Dual-Port Memory, Packet-based services, Revision 4, English, 2020.

[3] Hilscher Gesellschaft für Systemautomation mbH: Protocol API, Socket Interface, Packet Interface, Revision 5, English, 2019.

[4] Hilscher Gesellschaft für Systemautomation mbH: Protocol API, Ethernet Interface, Packet Interface, Revision 11, English, 2020.

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

[6] Hilscher Gesellschaft für Systemautomation mbH: Application Note: Functions of the Integrated WebServer, Revision 6, English, 2017.

[7] ODVA: The CIP Networks Library, Volume 1, “Common Industrial Protocol (CIP™)”, Edition 3.27, November 2019.

[8] ODVA: The CIP Networks Library, Volume 2, “EtherNet/IP Adaptation of CIP”, Edition 1.25, November 2019.

[9] The Common Industrial Protocol (CIP™) and the Family of CIP Networks, Publication Number: PUB00123R0, downloadable from ODVA website (http://www.odva.org/).

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

2 Hilscher EtherNet/IP Stack Capabilities

This chapter describes the interfaces of the Hilscher EtherNet/IP Stack, 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 Loadable Firmware, the netX chip is used as a dedicated communication processor while the host application executes on its own processor.

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

In the LFW scenario, the firmware satisfies two interfaces as shown in Figure 1.

1. The **DPM/Packet Interface** toward the host application for exchange of commands, event notification and process data via the Hilscher Dual-Port Memory Interface.
2. The **EtherNet/IP Interface** according to the CIP specification. Based on TCP/IP and UDP/IP, external devices communicate with the Protocol Stack over one of the three supported connection types: Unconnected Explicit Messaging (UCMM), Connected Explicit Messaging (Class 3) or Implicit Messaging (Class 1).

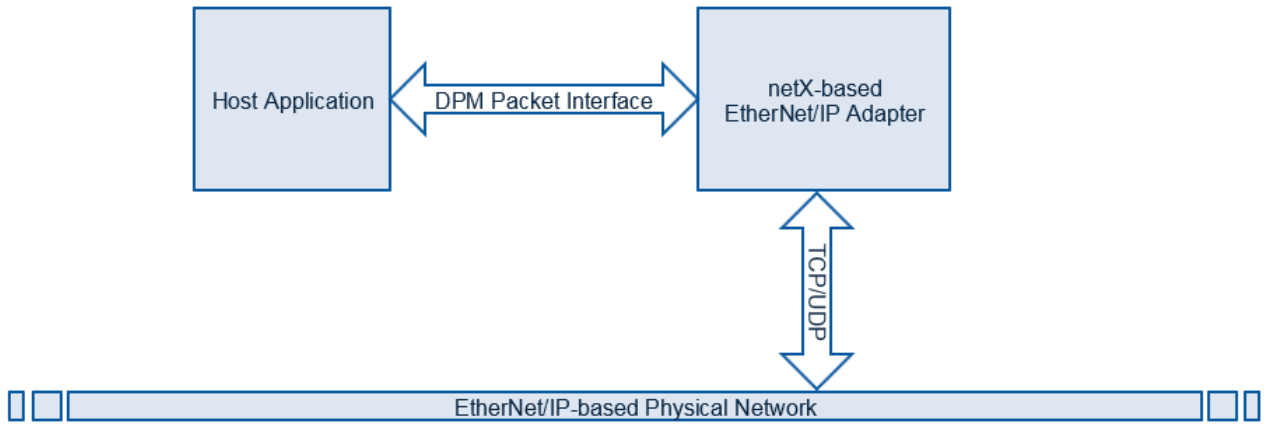


Figure 1: Interfaces of the EtherNet/IP Stack (LFW)

2.2 Structure of the EtherNet/IP Adapter Stack

The figure below shows the internal structure of the EtherNet/IP adapter stack and its components.

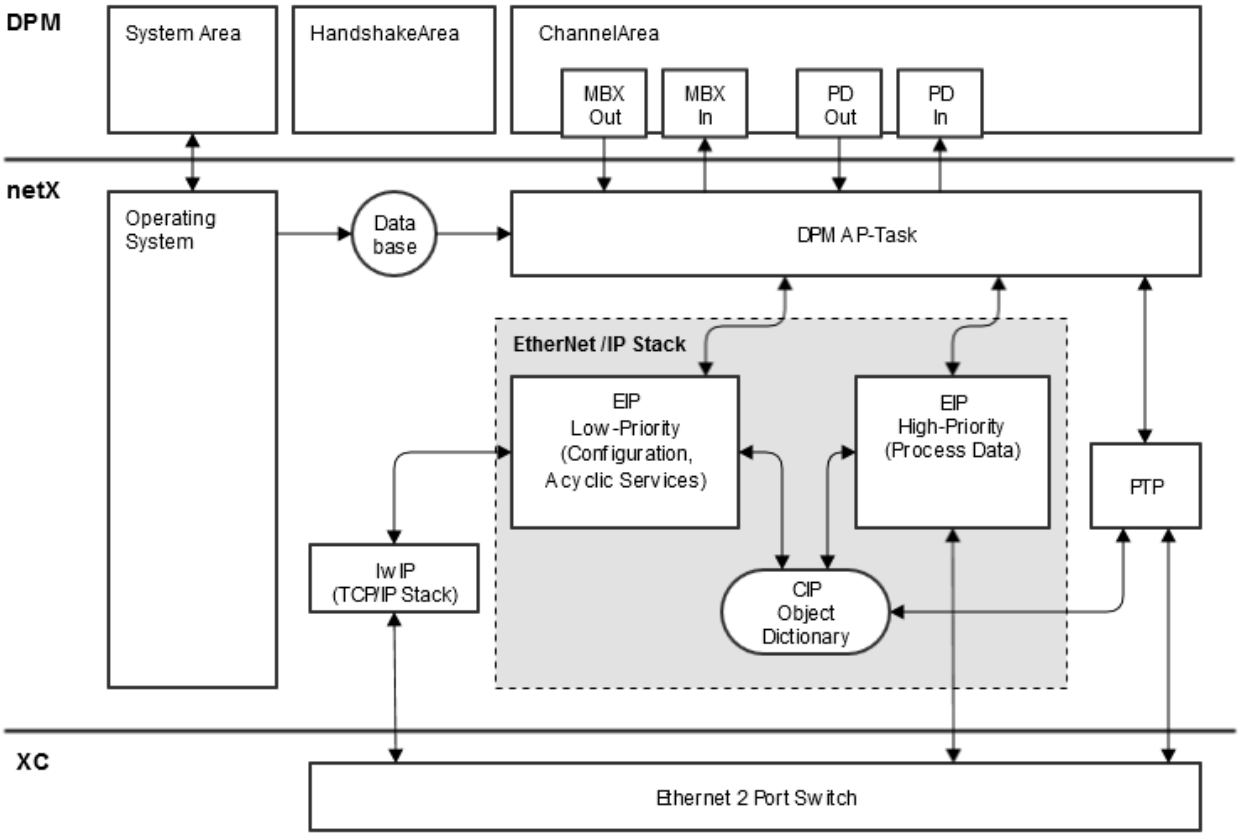


Figure 2: EtherNet/IP firmware structure

2.3 Available Object Classes

The following subsections describe all default CIP object classes that are available within the Hilscher EtherNet/IP stack. Figure 3 gives an overview of the available CIP objects and their instances in the Protocol Stack's default configuration.

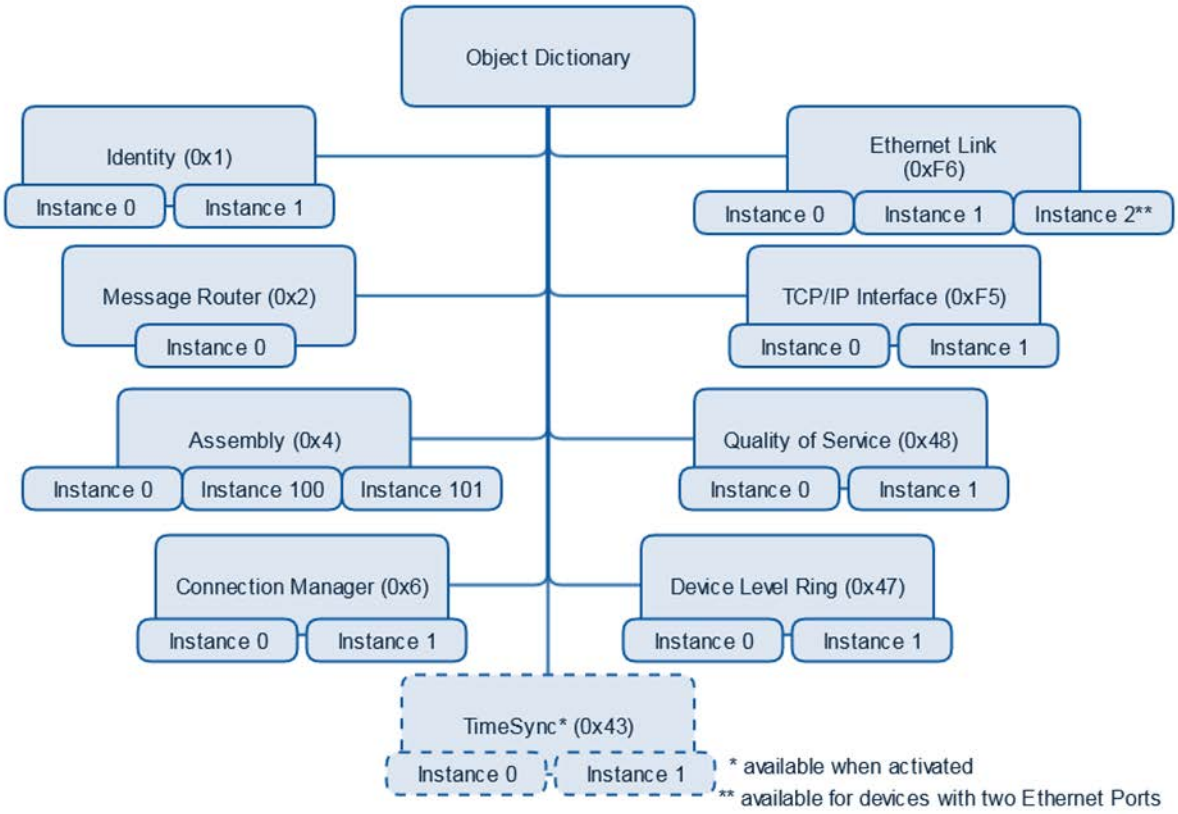


Figure 3: Default Hilscher Device Object Model

2.3.1 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. Naturally, the attributes each object class provides at class and instance levels are different 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 uses 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.

2.3.2 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 4: 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's EtherNet/IP Interface – see 2.1). 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 2.1) of the stack (see description of packet *CIP Service Request* on page 121).

   The definitions for access rules 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).
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 this attribute is supported by the stack 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 "Set Attribute Option". See section *Attribute Option Flags* on page 51 and *Set Attribute Option (0xFF34)* on page 52).

   ➔ The attribute is supported and activated per default.

   ➔ The attribute is supported and deactivated per default. The host can activate it.

   ➔ The attribute is not supported. The host cannot activate it.

2.3.3 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 5: Introduction of Instance Attribute Description

2.3.4 Services

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

Correspondingly, we will present services supported by each object class in two tables:

The first table shows common services that can be issued toward the protocol stack over the network, as well as by the host application. The second table shows Hilscher-specific services that are available to the host application only.

Both tables have the same format:

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
1	2	3	4	5

Table 6: Introduction of Service Description

1. **Service Code** is a hexadecimal value to uniquely identify the CIP service. Service codes below the value 255 are defined within the EtherNet/IP specification. Larger numbers are reserved for Hilscher-specific services. Hilscher-specific services are described separately in chapter *Hilscher-specific CIP services* on page 51.
2. **Name** specifies the name of the service.
3. Addressing the object's class level

   ➔ 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 object's instance level

   ➔ 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.

2.3.5 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.

2.3.5.1 Class Attributes

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

Table 7: Identity Object - Class Attributes

2.3.5.2 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	(1)	✓
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 2.6.1	✓
7	Product Name	Get	Get/Set	Human readable identification	“netX”	✓
8	State	Get	Get	Present state of the device		✓
9	Conf. Consist. Value	Get	Get	Configuration Consistency Value	0	✓
10	Heart Beat Interval	Get	Get	The nominal interval between heartbeat messages in seconds.	0	⚠
19	Protection Mode	Get	Get/Set	Current protection mode of the device (see 2.12 “CIP Device protection” for more information)	0	✓

Table 8: Identity Object - Instance Attributes

2.3.5.3 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 2.3.12.2), the reset service will not be support for any instance. In that case the service will be reject with general status code 0x08 "Service not supported".				

Table 9: Identity Object - Common Services

2.3.5.4 Hilscher-specific services

These services are available to the host application only. All other service codes are reserved for stack-internal use only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0xFF32[1)]	Reserved	✓	✓	Reserved for stack-internal use only.
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options
0x0101	Enable/Disable "Flash LEDs" service	✓	✓	Enables or disables the service 0x4B "Flash LEDs" Service data size = 1Byte If service shall be disabled: [00] If service shall be enabled: [01] Note: the Flash_LEDs service is enabled by default.
1) The host application shall not send this service, as this is an internal stack service.				

Table 10: Identity Object – Hilscher-specific Services

2.3.6 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.

2.3.6.1 Class Attributes

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

Table 11: Message Router Object - Class Attributes

2.3.6.2 Instance Attributes

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

2.3.6.3 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 12: Message Router Object - Common Services

2.3.6.4 Hilscher-specific services

These services are available to the host application only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0xFF33	Get Attribute Option	✓	n.a.	Retrieve attribute options
0xFF34	Set Attribute Option	✓	n.a.	Modify attribute options

Table 13: Message Router Object – Hilscher-specific Services

2.3.7 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.

2.3.7.1 Class Attributes

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

Table 14: Assembly Object - Class Attributes

2.3.7.2 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.	✓
2	Member	Get	Get	Member list	n.a.	✓
3	Data	Get/Set	Get/Set	Current process data snapshot	n.a.	✓
4	Size	Get	Get/Set	Process data size in number of bytes	n.a.	✓
768	Member data list	None	None	Data of assembly members	n.a.	⚠
769	Parameter	None	Get	Assembly parameter	n.a.	⚠
770	Status	None	Get	Status of the assembly	n.a.	⚠

Table 15: Assembly Object - Instance Attributes

2.3.7.3 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 16: Assembly Object - Common Services

2.3.7.4 Hilscher-specific services

These services are available to the host application only. All other service codes are reserved for stack-internal use only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x0401[1)]	Reserved	✓	✗	Reserved for stack-internal use only.
0x0402[1)]		✗	✓	
0x0403[1)]		✗	✓	
0x0404[1)]		✗	✓	
0x0405[1)]		✗	✓	
0x0406[1)]		✗	✓	
0x0407[1)]		✗	✓	
0x0408[1)]		✗	✓	
0x0409[1)]		✗	✓	
0x040A[1)]		✓	✓	
0x040D[1)]		✓	✓	
0x040E[1)]		✓	✓	
0x040F[1)]		✓	✓	
0xFF32[1)]		✓	✓	
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options
1) The host application shall not send this service, as this is an internal stack service.				

Table 17: Assembly Object - Hilscher-specific Services

2.3.8 Connection Manager Object (Class Code: 0x06)

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

2.3.8.1 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 18: Connection Manager Object - Class attributes

2.3.8.2 Instance attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Open Requests	Get	Get	Number of Forward Open service requests received.	(0)	⚠

Table 19: Connection Manager Object - Instance Attributes

2.3.8.3 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	✓	✓	Open new connection
0x4E	Forward Close	✓	✓	Close connection

Table 20: Connection Manager Object - Common Services

2.3.8.4 Hilscher-specific services

These services are available to the host application only. All other service codes are reserved for stack-internal use only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0xFF32[1)]	Reserved	✓	✓	Reserved for stack-internal use only.
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options
1) The host application shall not send this service, as this is an internal stack service.				

Table 21: Connection Manager Object – Hilscher-specific Services

2.3.9 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 (0x1A02).

Note: The TimeSync object has to be registered during the stack configuration sequence, before the EIP_APS_CONFIG_DONE_REQ or HIL_CHANNEL_INIT_REQ packets. Registration during runtime leads to undefined behavior.

For further information regarding CIP Sync and its use with the EtherNet/IP protocol stack and your host application, please refer to the corresponding Application Note [5].

2.3.9.1 Class Attributes

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

Table 22: Time Sync Object - Class Attributes

2.3.9.2 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	✓

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
5	OffsetFromMaster	Get	Get	Offset between local clock and master clock	0	✓
6	MaxOffsetFromMaster	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	all 0	✓
9	ParentClockInfo	Get	Get	Parent Clock Info	all 0	✓
10	LocalClockIno	Get	Get	Local Clock Info	all 0	✓
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	0	✓
15	PortLogSyncIntervalCfg	Get/Set	Get/Set	Port log sync interval cfg	0	✓
16	Priority1			Priority 1	n.a.	✗
17	Priority2			Priority 2	n.a.	✗
18	DomainNumber	Get/Set	Get/Set	Domain number	0	✓
19	ClockType	Get	Get	Clock type	0	✓
20	ManufactureIdentity	Get	Get	Manufacture identity	all 0	✓
21	ProductDescription	Get	Get	Product description	“”	✓
22	RevisionData	Get	Get	Revision data	“”	✓
23	UserDescription	Get	Get	User description	“”	✓
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	all 0	✓
29	AssociatedInterfaceObjects	Get	Get	Objects associated with PTP ports	CIP path to Ethernet Link object	✓
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 23: Time Sync Object - Instance Attributes

2.3.9.3 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 24: Time Sync Object - Common Services

2.3.9.4 Hilscher-specific services

These services are available to the host application only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0xFF32[1)]	Reset Object	✓	✓	Perform a reset of the object
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options
1) The host application shall not send this service, as this is an internal stack service. The service is documented for the sake of completeness only.				

Table 25: Time Sync Object – Hilscher-specific services

2.3.9.5 Instance Attributes

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 that the host application will receive in order 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 Dual Port Memory interface (for further information see CIP Sync Application Note [5]).

Note: Currently, only Sync 0 can be used.

Time Sync Object- Attribute 768 (0x300)			
Variable	**Type**	**Value/Range**	**Description**
`ulSync0Interval`	UINT32	0, 10000 … 999999999 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).
`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, 10000 … 999999999 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).
`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 26: Time Sync Object – Attribute 768 (0x300)

2.3.10 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.

2.3.10.1 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 27: DLR Object - Class Attributes

2.3.10.2 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 28: DLR Object - Instance Attributes

2.3.10.3 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 29: DLR Object - Common Services

2.3.10.4 Hilscher-specific services

These services are available to the host application only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0xFF32[1)]	Reset Object	✓	✓	Perform a reset of the object
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options

1) The host application shall not send this service, as this is an internal stack service. The service is documented for the sake of completeness only.

Table 30: DLR Object – Hilscher-specific Services

2.3.11 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.

2.3.11.1 Class Attributes

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

Table 31: QoS Object - Class Attributes

2.3.11.2 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	**(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 32: QoS Object - Instance Attributes

2.3.11.3 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 33: Quality of Service Object - Common Services

2.3.11.4 Hilscher-specific services

These services are available to the host application only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options

Table 34: Quality of Service Object – Hilscher-specific Service

2.3.12 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.

2.3.12.1 Class Attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	Revision	Get	Get/Set	Revision of this object	(4)	✓
2	Max. Instance	Get	Get/Set	Maximum instance number of an object currently created in this class level of the device	(1)	✓
3	Number of Instances	Get	Get/Set	The number of Instances currently created in this class	(1)	⚠
6	Maximum ID Number Class Attributes	Get	Get/Set	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/Set	The attribute ID number of the last instance attribute of the class definition implemented in the device.	(13)	⚠

Table 35: TCP/IP Interface Object - Class Attributes

2.3.12.2 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	Set	Get/Set	Interface control flags	(0)	✓
4	Physical Link Object	Get	Get/Set	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.)	(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.	("")	✓

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
7	Safety Network Number[1)]	Get	Get/Set	See CIP Safety Specification, Volume 5, Chapter 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)	✓
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 36: TCP/IP Interface Object - Instance Attributes

2.3.12.3 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 37: TCP/IP Interface Object - Common Services

2.3.12.4 Hilscher-specific services

These services are available to the host application only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0xF501[1)]	Get Multicast	✓	✓	Generate and return new multicast address
0xFF32[1)]	Reset Object	✓	✓	Perform a reset of the object
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options

1) The host application shall not send this service, as this is an internal stack service. The service is documented for the sake of completeness only.

Table 38: TCP/IP Interface Object – Hilscher-specific Services

2.3.13 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.

2.3.13.1 Class Attributes

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

Table 39: Ethernet Link Object - Class Attributes

2.3.13.2 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)	✓

<table>
<tr><th rowspan="2">Attr ID</th><th rowspan="2">Name</th><th colspan="2">Access</th><th rowspan="2">Description</th><th rowspan="2">Default Value</th><th rowspan="2">Supported by default</th></tr>
<tr><th>from Network</th><th>from Host</th></tr>
<tr><td>9</td><td>Admin State</td><td>Get/Set</td><td>Get/Set</td><td>Administrative state:<table><tr><td>1</td><td>EIP_EN_INTF_STATE_ENABLE</td><td>Enable interface</td></tr><tr><td>2</td><td>EIP_EN_INTF_STATE_DISABLE</td><td>Disable Interface</td></tr></table></td><td>(disable)</td><td>✓</td></tr>
<tr><td>10</td><td>Interface Label</td><td>Get</td><td>Get/Set</td><td>Human readable identification</td><td>(“port1”,”port2”)</td><td>✓</td></tr>
<tr><td>11</td><td>Interface Capability</td><td>Get</td><td>Get/Set</td><td>Indication of capabilities of the interface</td><td>10 / HD,
10 / FD,
100 / HD
100 / FD</td><td>✓</td></tr>
<tr><td>768</td><td>MDIX</td><td>Set</td><td>Get/Set</td><td>MDIX configuration
Format: uint8_t, range [1 .. 3]<table><tr><td>1</td><td>EIP_EN_INTF_MDIX_AUTO</td><td>Auto detect</td></tr><tr><td>2</td><td>EIP_EN_INTF_MDIX_MDI</td><td>Explicit MDI</td></tr><tr><td>3</td><td>EIP_EN_INTF_MDIX_MDIX</td><td>Explicit MDIX</td></tr></table></td><td>1</td><td>✓</td></tr>
</table>

Table 40: Ethernet Link Object - Instance Attributes

2.3.13.3 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 41: Ethernet Link Object - Common Services

2.3.13.4 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 42: Ethernet Link Object – Class-Specific Services

2.3.13.5 Hilscher-specific services

These services are available to the host application only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options

Table 43: Ethernet Link Object – Hilscher-specific Services

2.3.14 Predefined Connection Object (Class Code: 0x401)

The Predefined Connection Object maintains implicit (class 1) connections. It is a Hilscher-specific CIP object, which is not covered by the CIP specification.

2.3.14.1 Class Attributes

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

Table 44: Predefined Connection Object - Class Attributes

2.3.14.2 Instance Attributes

Attr ID	Name	Access		Description	Default Value	Supported by default
		from Network	from Host			
1	State	Get	Get	State of the connection		✓
2	Count	Get	Get	Number of connections		✓
3	Configuration	Get	Get/Set	Connection configuration		✓

Table 45: Predefined Connection Object - Instance Attributes

2.3.14.3 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	
0x0E	Get Attribute Single	✓	✓	Retrieve attribute value
0x10	Set Attribute Single	✗	✓	Modify attribute value

Table 46: Predefined Connection Object - Common Services

2.3.14.4 Hilscher-specific services

These services are available to the host application only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0x08	Create	✓	✗	Create new predefined connection instance
0x09	Delete	✗	✓	Delete predefined connection instance
0xFF01[1)]	Open Connection	✓	✗	Verifies connection parameters and eventually sets connection state, effectively Opening a connection and binding the involved instances of the Assembly object
0xFF02[1)]	Close Connection	✓	✗	Closes an open connection by setting the connection state and unbinding the involved assembly objects
0xFF32[1)]	Reset Object	✓	✓	Perform a reset of the object
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options
1) The host application shall not send this service, as this is an internal stack service. The service is documented for the sake of completeness only.				

Table 47: Predefined Connection Object – Hilscher-specific Services

2.3.14.5 Create (0x08)

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

Request Service Data Field Parameters:

Name	Byte Size	Description
Consumer Connection Point	4	Connection point addressing the O2T (Originator to Target) direction. Typically, this is an assembly instance number.
Producer Connection Point	4	Connection point addressing the T2O (Target to Originator) direction. Typically, this is an assembly instance number.
Configuration Connection Point	4	Connection point addressing the configuration assembly instance.
Minimum O2T RPI	4	Minimum Requested Packet Interval (RPI) of the consuming direction in microseconds
Maximum O2T RPI	4	Maximum Requested Packet Interval (RPI) of the consuming direction in microseconds
Minimum T2O RPI	4	Minimum Requested Packet Interval (RPI) of the producing direction in microseconds
Maximum T2O RPI	4	Maximum Requested Packet Interval (RPI) of the producing direction in microseconds

Name	Byte Size	Description

Supported Trigger Types	1	This field specifies the supported trigger types of the connection. There can be up to 3 trigger types supported.
		Use the following flags
		`#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.
		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).

Name	Byte Size	Description

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 [7]

Table 48: Hilscher Service – Create – Request Data Parameters

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.

2.3.14.6 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.

2.3.15 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, which is not covered by the CIP specification.

2.3.15.1 Class attributes

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

Table 49: Diagnosis Object - Class attributes

2.3.15.2 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 50: Diagnosis Object - Instance attributes

2.3.15.3 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 51: Diagnosis Object - Common services

2.3.15.4 Hilscher-specific services

None

2.3.16 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.

2.3.16.1 Class Attributes

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

Table 52: IO Mapping Object - Class

2.3.16.2 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 53: IO Mapping Object - Instance Attributes

2.3.16.3 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
0x10	Set Attribute Single	✗	✓	Modify attribute value

Table 54: IO Mapping Object - Common Services

2.3.16.4 Hilscher-specific services

These services are available to the host application only.

Service Code	Name	Addressing the object's		Description
		Class Level	Instance Level	
0xFF01[1)]	Create Member	✗	✓	Creates a new I/O Mapping entry, i.e. a new object class instance.
0xFF02[1)]	Delete Member	✗	✓	Deletes I/O Mapping entry, i.e. the addressed object class instance.
0xFF32[1)]	Reset Object	✓	✓	Perform a reset of the object
0xFF33	Get Attribute Option	✓	✓	Retrieve attribute options
0xFF34	Set Attribute Option	✓	✓	Modify attribute options
1) The host application shall not send this service, as this is an internal stack service. The service is documented for the sake of completeness only.				

Table 55: IO Mapping Object – Hilscher-specific Services

2.4 Hilscher-specific CIP services

2.4.1 Common

2.4.1.1 Attribute Option Flags

CIP Attribute Option Flags control the stack on how a certain attribute is to be treated. On the one hand, attribute flags encode access rights and on the other, they control how an attribute is treated by the stack.

EtherNet/IP defines the following four levels of access permission with increasing authority:

- Bus access,
- User access,
- Admin access
- No access.

The higher access rights **imply** the lower ones. This means that if, e.g. bus access is permitted for a particular attribute, then also user access and admin access is granted. Access rights are maintained for both data directions, i.e. Read and Write, or, in CIP-terminology, Get and Set. The host application can modify these attribute flags by means of the services “Set Attribute Option” and “Get Attribute Option”.

Name	Description
CIP_FLG_SET_ACCESS_BUS (0x0010)	The attribute’s value is modifiable over the network.
CIP_FLG_SET_ACCESS_USER (0x0020)	The attribute’s value is modifiable over the host interface.
CIP_FLG_SET_ACCESS_ADMIN (0x0040)	The attribute’s value is modifiable internally by the stack itself.
CIP_FLG_SET_ACCESS_NONE (0x0080)	The attribute’s value is not modifiable.
CIP_FLG_GET_ACCESS_BUS (0x0100)	The attribute’s value is readable over the network.
CIP_FLG_GET_ACCESS_USER (0x0200)	The attribute’s value is readable over the host interface.
CIP_FLG_GET_ACCESS_ADMIN (0x0400)	The attribute’s value is readable internally by the stack itself.
CIP_FLG_GET_ACCESS_NONE (0x0800)	The attribute’s value is not readable.
CIP_FLG_TREAT_FORWARD (0x1000)	Services toward this attribute will be forwarded to the host application (not implemented).
CIP_FLG_TREAT_NOTIFY (0x2000)	The stack will notify the host application about changes in the attribute value by means of an Object Change Indication. Such changes have to be acknowledged by sending the corresponding object change response. See section 4.2.6 for further information.
CIP_FLG_TREAT_DISABLE (0x4000)	The attribute is disabled and will be treated as an unsupported attribute. It still may be exposed in the response to a Get_Attributes_All CIP request.
CIP_FLG_TREAT_PROTECTED (0x8000)	Atrribute is protected (not settable) when protection mode is active (see 2.12 “CIP Device protection” for more information)

Table 56: Attribute Option Flags

2.4.1.2 Get Attribute Option (0xFF33)

The Hilscher-specific “Get Attribute Option” service returns the option flags of the targeted attribute.

Request Service Data Field Parameters:

The service does not accept any parameters.

Success Response Service Data Field Parameters:

Name	Byte Size	Description
Option Flags	2	This is a combination (Boolean OR) of Attribute Flags as described in Table 56.

Table 57: Hilscher Service – Get Attribute Option – Response Data Parameters

Unsuccessful Response Service Data Field Parameters:

The unsuccessful response does not provide any data.

2.4.1.3 Set Attribute Option (0xFF34)

The Hilscher-specific “Set Attribute Option” service writes the option flags of the targeted attribute.

Request Service Data Field Parameters:

Name	Byte size	Description
Option Mask	2	This is a combination (Boolean OR) of Attribute Flags as described in Table 56. All bits set in this bitmask will be affected by the service: ▪ Attribute Flags not set in this mask will not be changed, despite of the value of the corresponding bit index in “Option Flags” ▪ Attribute Flags set in this mask, which are not set in “Option Flags” will be cleared from the Attribute Flags of the targeted attribute ▪ Attribute Flags set in this mask, which are also set in “Option Flags” will be set in the Attribute Flags of the targeted attribute.
Option Flags	2	This is a combination (Boolean OR) of Attribute Flags as described in Table 56. Attribute Flags that are set in this bit field will be set in the Attribute Flags of the targeted attribute, if the corresponding bit index is also set in “Option Mask” Attribute Flags that are not set in this bit field, but are set in “Option Mask”, will be cleared from the Attribute Flags of the targeted attribute.

Table 58: Hilscher Service – Set Attribute Option – Request Data Parameters

Note: The operation technically applying to the Flags of the targeted attribute is, in C:

```
usFlags &= ~usOptionMask;
usFlags |= usOptionFlags & usOptionMask;
```

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.

2.5 Ethernet MAC Address

The protocol stack requires one MAC address to operate. This MAC address, as reflected in the corresponding attribute of the CIP EthernetLink 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 SecMem or FDL data sources. The host application cannot set the MAC address attribute directly. This should be fine for most applications.

Speaking for the firmware, two MAC addresses may be required, one for the protocol stack, and another one for the Raw EtherNet subsystem on DPM Comm channel 1, which one enables statically by means of the taglist (see section *Resource and feature configuration via tag list* on page 172). Table 56 shows the use of the DDP MAC addresses.

Anyway, if the host applicaton 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* on page 172). 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 },   /* only this MAC address is actually used by EtherNet/IP */
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x1 },
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x2 },
  { 0xa, 0xb, 0xc, 0xd, 0xe, 0x3 },   /* This is the second chassis MAC (RawEthernet/NDIS)*/
  { 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	Mapping to Flash Device Label	Required
First DDP MAC address	EtherNet/IP communication (DPM channel 0) Socket API communication (DPM channel 1)	MAC address 1 (communication CPU)	Yes
Fourth DDP MAC address	Raw Ethernet / NDIS (DPM channel 1)	MAC address 4 (communication CPU)	Required if the Raw Ethernet / NDIS is taglist-activated.

Table 59: Ethernet MAC addresses

2.6 Device data

The Flash Device Label contains device data. The device-specific data in the Flash Device Label is set during production of the device. During start of the firmware, the firmware reads this data into the Device Data Provider.

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, Attribtue 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 1: Basic Device Data in the Flash Device Label

The Flash Device Label offers the possibility to store OEM-specific device data. 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	Each flag determine whether the parameter value from the Basic Device Data or from OEM identification is to be used. Bit 0: If 1, `Serial number` is valid. If 1, `OEM Serial number` is valid. Bit 1: If 0, `Device number` is valid. If 1, `OEM order number` is valid. Bit 2: If 0, `Hardware revision number` is valid. If 1, `OEM hardware revision` is valid. Bit 3: If 0, `Production date` is valid. If 1, `OEM production date/time` is valid.	-
OEM serial number	Mapped to Identity Object, Attribtue 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 2: OEM identification in the Flash Device Label

2.6.1 Device Serial Number

The device serial number as reflected in the CIP Identity Object, attribute 6, together with the vendor ID, 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 of its devices.

Per default, the protocol stack applies the serial number from the underlying Device Data Provider (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.

Anyway, if the host applicaton 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 5). Then, it uses the DDP's OEM serial number attribute to set a custom serial number and render this data valid and finally, set the DDP active. The following pseudo code illustrates this approach:

```
  /* optionally when initial DDP state is passive: set additional (OEM) DDP base device parameters:
serial number */
  HIL_DDP_SERVICE_SET_REQ_T *ptReq = (HIL_DDP_SERVICE_SET_REQ_T*)&myPacket;
  char *szSerialNumber = "76543";
  memset(&ptReq->tHead, 0, sizeof(ptReq->tHead));
  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);
  /* also render the OEM serial number "valdid" in the corresponding bitfield */
  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 = 0x1;    /* set bit zero to set OEM serial number valid */
  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);
```

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

2.7 Status information

2.7.1 DPM Communication State

This section describes how the EtherNet/IP Adapter uses the communication state. The communication state is located in the Dual Port Memory as described in [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).

Table 60: Communication States

The following figure illustrates how the communcation state transitions dependent on specific events.

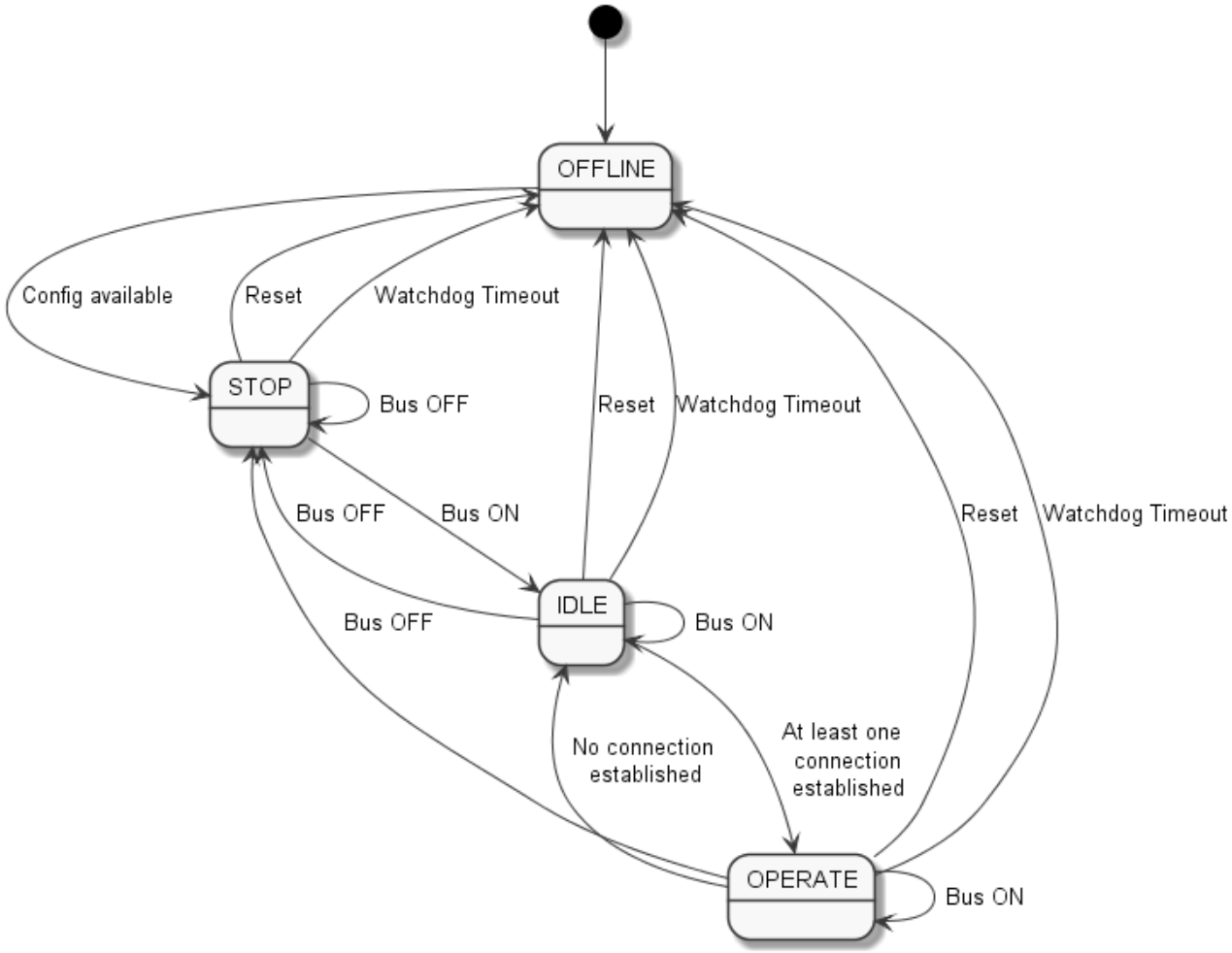


Figure 4: Communication State Transitions

2.7.2 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, which is located in the communication state field (ulCommunicationCOS).
DPM flag „RESTART REQUIRED“	The EtherNet/IP protocol stack does not handle the DPM flag HIL_COMM_COS_RESTART_REQUIRED, which is located in the communication state field (ulCommunicationCOS).
DPM flag „RESTART REQUIRED ENABLE“	The EtherNet/IP protocol stack does not handle the DPM flag HIL_COMM_COS_RESTART_REQUIRED_ENABLE, which is located in the communication state field (ulCommunicationCOS).

Table 61: DPM COS Flags

2.7.3 Other DPM Status Bits

The NCF_COMMUNICATING bit is set for the DPM channel if at least one CIP connection is open, may it be in a target (adapter) or originator role (scanner), and in any CIP connection class.

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

2.8 Module and Network Status

This section describes the LED signaling of EtherNet/IP Adapter devices. Two LEDs display status information namely the Module Status LED denominated as MS and the network Status LED denominated as NS.

2.8.1 Module Status

The following table lists the possible values of the Module Status 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 steady off.
`EIP_MS_SELFTEST`	1	**Self-Test** While the device is performing its power up testing, the module status indicator flashes green/red.
`EIP_MS_STANDBY`	2	**Standby** If the device has not been configured, the module status indicator flashes green.
`EIP_MS_OPERATE`	3	**Device operational** If the device is operating correctly, the module status indicator is steady green.
`EIP_MS_MAJOR_RECOVERABLE_FAULT`	4	**Major recoverable fault** If the device has detected a major recoverable fault, the module status indicator flashes red. **Note:** An incorrect or inconsistent configuration would be 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 steady red.

Table 62: Possible values of the Module Status

2.8.2 Network Status

The following table lists the possible values of the Network Status 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 steady 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 static red.
`EIP_NS_SELFTEST`	5	**Self-Test** The device is performing its power-on self-test (POST). During POST, the network status indicator alternates flashing green and red.

Table 63: Possible values of the Network Status

2.9 Handshake Modes

The handshake mode is get and set by means of the services HIL_GET_TRIGGER_TYPE_REQ (section *Get Trigger Type* on page 170) and HIL_SET_TRIGGER_TYPE_REQ, respectively. 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	Free Running	Disabled
Receive (RX) triggered		Enabled

Table 64: Supported handshake modes

2.9.1 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 or writes I/O data from or toward the protocol stack, control over the input or output areas, respectively, is given to the protocol stack which immediately copies the current process data into (or out of) the DPM and toggles control over the area back 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 reads 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 92 on page 115. In this mode, when the host reads process data from the protocol stack, control over the DPM input area is given to the protocol stack which toggles control back on the next receive of process data on the network if that data is directed toward one of the Trigger Assemblies. If multiple Trigger Assemblies are present, it is undefined which subset of them has possibly been updated and the host application has to take further means to decide over the age of each Assembly's data segment. For further information, please refer to assembly option flag `EIP_AS_OPTION_MAP_SEQCOUNT` in Table 92.

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: We encourage you to decide during the design phase of your project if a time-triggered (Sync Handshake) or event-triggered (Receive-triggered) system suits your needs best and to decide for one type. Although it is technically possible to combine the two modes, settling with one method leads to a clear design and lower complexity implementation.

Note: When in handshake mode “Receive (RX) triggered”, and currently no I/O connection is established towards any Assembly that is eligible for triggering, then control over the input area will not be toggled back to the host until a new connection is established and a first I/O frame is received.

2.9.2 Synchronization Handshake Mode

Independently 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.

Please refer to Application Note [5] for a detailed description on CIP Sync.

2.9.3 Configuration

The handshake mode can be configured using the *Set Trigger Type Request* `HIL_SET_TRIGGER_TYPE_REQ` (0x2F90). Refer to section *Set Trigger Type* (on page 167) or reference [2] for further information.

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);
```

2.10 Quality of Service (QoS)

2.10.1 Introduction

Quality of Service, abbreviated as QoS, denotes a mechanism treating data streams according to their delivery characteristics, of which the by far most important one is the priority of the data stream. Therefore, in the context of EtherNet/IP, QoS means 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 (DLR)* on page 65).

In TCP/IP-based protocols, there are two standard mechanisms available for implementing QoS, which are described in more detail below:

- Differentiated Services (abbreviated as DiffServ)
- The 802.1D/Q Protocols

Introducing QoS means providing network infrastructure devices such as switches and hubs with means to differentiate between frames of different priority. Therefore, these devices write priority information into the frames. This technique is called priority tagging.

2.10.2 DiffServ

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

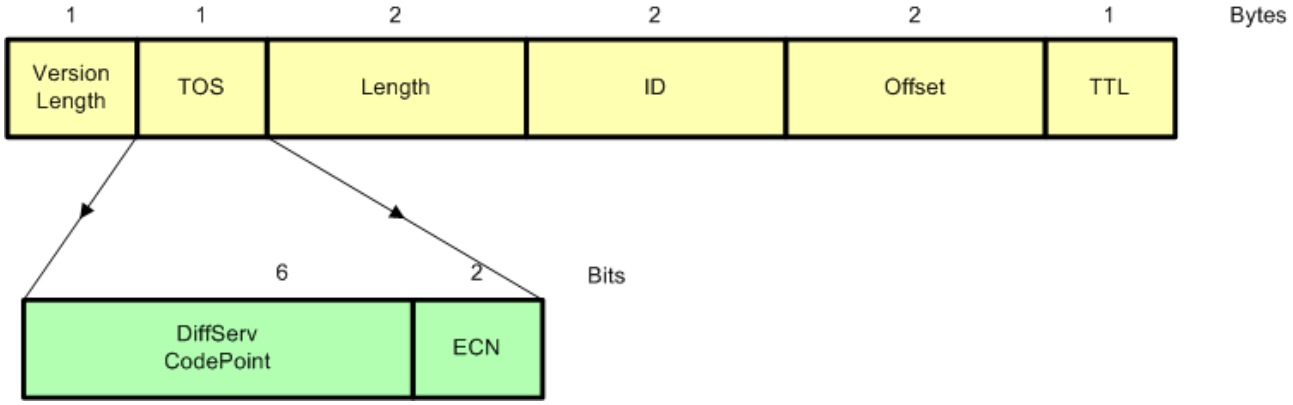


Figure 5: TOS Byte in IP v4 Frame Definition

DiffServ is a schematic model for the priority-based classification of IP frames based on an alternative interpretation of the TOS byte. It has been specified in RFC2474.

The idea of DiffServ consists in redefining 6 bits (i.e. the bits 8 to 13 of the whole IP v4 frame) and to use them as codepoint. Thus, these 6 bits are denominated as DSCP (*Differentiated Services Codepoint*) in the context of DiffServ. These 6 bits allow to address 63 predefined routing behaviors, which can be applied for routing the frame at the next router, and specifies 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 these PHBs. For a list of these DSCPs and the assigned PHBs, see http://www.iana.org/assignments/dscp-registry/dscp-registry.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 which is defined in 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 65: Default Assignment of DSCPs in EtherNet/IP

2.10.3 802.1D/Q Protocol

Another possibility is used by 802.1Q. 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, which is located between Source MAC and Ethertype and Size in the standard Ethernet frame.

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

- The ID of the Virtual LAN (VLAN ID, 12 bits wide)
- And the priority (defined in 802.1D)

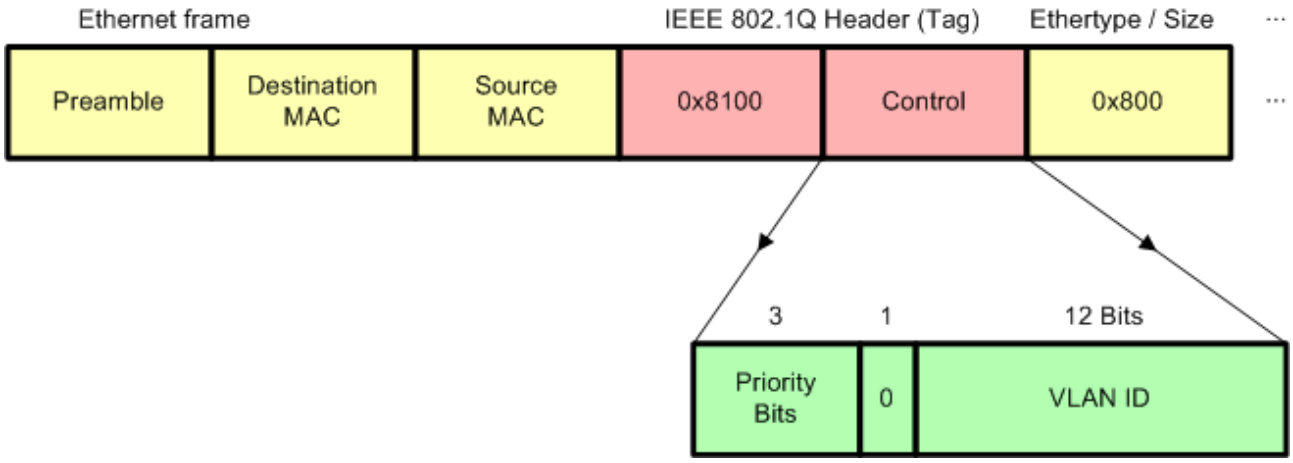


Figure 6: Ethernet Frame with IEEE 802.1Q Header

As the header definition reserves only 3 bits for the priority (see figure below), 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 which is defined in 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 66: Default Assignment of 802.1D/Q Priorities in EtherNet/IP

2.10.4 The QoS Object

Within the EtherNet/IP implementation of QoS, the DiffServ mechanism is usually always present and does not need to be activated explicitly. In contrast to this, 802.1Q must explicitly be activated on all participating devices. The main capabilities of the QoS object are therefore:

- To enable 802.1Q (VLAN tagging)
- To enable setting parameters related to DiffServ (DSCP parameters)

For more information on the QoS object in the Hilscher EtherNet/IP adapter protocol stack see section *Quality of Service Object (Class Code: 0x48)* on page 35.

2.10.4.1 Enable 802.1Q (VLAN tagging)

The 802.1Q VLAN tagging mechanism can be turned on and off by setting attribute 1 (802.1Q Tag Enable) of the QoS object to value 1.

2.11 Device Level Ring (DLR)

This section gives a brief overview about 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. It 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 kinds of nodes in the network:

- Ring supervisors
- Ring nodes

DLR requires all nodes to be equipped with two Ethernet ports and internal switching technology. Each sent frame propagates on both ports, in both directions through the ring.

On reception, each module within the DLR network checks the target address of the received frame whether it matches its own MAC address.

- If the frame is targeting the node's MAC address, it consumes and processes the frame. Thus, the frame will not propagate any further through the ring.
- If the frame targets another MAC, the node propagates the packet to the next ring node by sending it on its other port

The active ring supervisor uses to disable one of its ports in order to, technically, achieve a line topology and prevent looping packets.

2.11.1 Ring Supervisors

There are two kinds of supervisors defined:

- Active supervisors
- Back-up supervisors

Note: The Hilscher EtherNet/IP stack does not support the ring supervisor mode.

Active supervisors

The active supervisor has the following duties:

- It periodically sends beacon and announce frames.
- It permanently verifies the ring integrity.
- It reconfigures the ring in order 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 necessary that each DLR network has 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. Therefore, each supervisor is assigned a precedence value. The supervisor with the highest precedence becomes the active ring supervisor, whereas all others stay passive in the role of back-up supervisors.

2.11.2 Beacon and Announce Frames

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

They differ in the following:

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 every beacon interval, with a typical interval 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

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

2.11.3 Ring Nodes

This subsection deals with modules in the ring, which do not have supervisor capabilities. These are denominated as (normal) ring nodes.

There are two types of normal ring nodes within the network:

- Beacon-based
- Announce-based

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

Nodes of type beacon-based have the following capabilities

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

Nodes of type announce-based have the following capabilities

- They implement the DLR protocol, but without the ring supervisor capability
- They do not process beacon frames, they just forward beacon frames
- They must be able to process announce frames
- This type is often only a software solution

Note: Hilscher devices running an EtherNet/IP firmware always run as a beacon-based ring node.

A ring node (independently whether it works beacon-based or announce-based) may have three internal states.

- IDLE_STATE
- FAULT_STATE
- NORMAL_STATE

For a beacon-based ring node, these states are defined as follows:

- IDLE_STATE

  The IDLE_STATE is the state which is reached after power-on. In IDLE_STATE the network operates as linear network, there is no ring support active. If one port receives a beacon frame from a supervisor, the state changes to FAULT_STATE.

- FAULT_STATE

  The Ring node reaches the FAULT_STATE after the following conditions:

  A. If a beacon frame from a supervisor is received on at least one port

  B. If at least one port receives a beacon frame from a different supervisor than the currently active one and the precedence of this supervisor is higher than that of the currently active one.

  The FAULT_STATE provides partial ring support, but the ring is still not operative in FAULT_STATE. If the beacon frames have a time-out on both ports, the state will change to the IDLE_STATE. If both ports receive a beacon frame and a beacon frame with RING_NORMAL_STATE has been received, the state changes to NORMAL_STATE.

- NORMAL_STATE

  The Ring node reaches the NORMAL_STATE only after the following condition:

  If a beacon frame from the active supervisor is received on both ports and a beacon frame with RING_NORMAL_STATE has been received

  The NORMAL_STATE provides full ring support. The following conditions will cause a change to the FAULT_STATE:

  A. A link failure has been detected.

  B. A beacon frame with RING_FAULT_STATE has been received from the active supervisor on at least one port.

  C. A beacon frame from the active supervisor had a time-out on at least one port

  D. A beacon frame from a different supervisor than the currently active one is received on at least one port and the precedence of this supervisor is higher than that of the currently active one.

For an announce-based ring node, these states are defined as follows:

- IDLE_STATE

  The IDLE_STATE is the state which is reached after power-on. It can also be reached from any other state if the announce frame from the active supervisor has a time-out. In IDLE_STATE the network operates as linear network, there is no ring support active. If an announce frame with FAULT_STATE is received from a supervisor, the state changes to FAULT_STATE.

- FAULT_STATE

  The Ring node reaches the FAULT_STATE after the following conditions:

  - If the network is in IDLE_STATE and an announce frame with FAULT_STATE is received from any supervisor.
  - If the network is in NORMAL_STATE and an announce frame with FAULT_STATE is received from the active or a different supervisor.
  - If the network is in NORMAL_STATE and a link failure has been detected.

  The FAULT_STATE provides partial ring support, but the ring is still not operative in FAULT_STATE.

  If the announce frame from the active supervisor has a time-out, the state will fall back to the IDLE_STATE.

  If an announce frame with NORMAL_STATE has been received from the active or a different supervisor, the state changes to NORMAL_STATE.

- NORMAL_STATE

  The Ring node reaches the NORMAL_STATE only after the following condition:

  - If the network is in IDLE_STATE and an announce frame with NORMAL_STATE is received from any supervisor.
  - If the network is in FAULT_STATE and an announce frame with NORMAL_STATE is received from the active or a different supervisor.

  The NORMAL_STATE provides full ring support. The following conditions will cause a fall back to the FAULT_STATE:

  - A link failure has been detected.
  - A announce frame with FAULT_STATE has been received from the active or a different supervisor.

  The following conditions will cause a fall back to the IDLE _STATE:

  - The announce frame from the active supervisor has a time-out.

2.11.4 Normal Network Operation

In normal operation, the supervisor sends beacon and announce frames in order 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.

2.11.5 Rapid Fault/Restore Cycles

Sometimes a series of rapid fault and restore cycles may occur in the DLR network for instance if a connector is faulty. If the supervisor detects 5 faults within a time period of 30 seconds, it sets a flag (Rapid Fault/Restore Cycles) which must explicitly be reset by the user then. This can be accomplished via the “Clear Rapid Faults” service.

2.11.6 States of Supervisor

A ring supervisor may have five internal states.

- IDLE_STATE
- FAULT_STATE (active)
- NORMAL_STATE (active)
- FAULT_STATE (backup)
- NORMAL_STATE (backup)

For a ring supervisor, these states are defined as follows:

- FAULT_STATE (active)

  The FAULT_STATE (active) is the state, which is reached after power-on if the supervisor has been configured as supervisor.

  The supervisor reaches the FAULT_STATE (active) after the following conditions:

  A. As mentioned above, at power-on

  B. From NORMAL_STATE (active):

     If a link failure occurs or if a link status frame indicating a link failure is received from a ring node or if the beacon time-out timer expires on one port

  C. From FAULT_STATE (backup):

     If on both ports, there is a time-out of the beacon frame from the currently active supervisor

  The FAULT_STATE (active) provides partial ring support, but the ring is still not operative in FAULT_STATE (active).

  If a beacon frame from a different supervisor than the currently active one is received on at least one port and the precedence of this supervisor is higher, the state will fall back to the FAULT_STATE (backup).

  If on both ports an own beacon frame has been received, the state changes to NORMAL_STATE (active).

- NORMAL_STATE (active)

  The supervisor reaches the NORMAL_STATE (active) only after the following condition:

  - If an own beacon frame is received on both ports during FAULT_STATE (active).

  The NORMAL_STATE provides full ring support.

  The following conditions will cause a change to the FAULT_STATE (active):

  A. A link failure has been detected.

B. A link status frame indicating a link failure is received from a ring node

C. The beacon time-out timer expires on one port

The following conditions will cause a change to the FAULT_STATE (backup):

A. A beacon frame from the active supervisor had a time-out on at least one port

B. If a beacon frame from a different supervisor with higher precedence is received on at least one port.

- FAULT_STATE (backup)

The supervisor reaches the FAULT_STATE (backup) after the following conditions:

A. From NORMAL_STATE (active):

A beacon frame from a supervisor with higher precedence is received on at least one port.

B. From FAULT_STATE (active):

A beacon frame from a different supervisor with higher precedence and the precedence of this supervisor is higher.

C. From NORMAL_STATE (backup):

i. A link failure has been detected.

ii. A beacon frame with RING_FAULT_STATE is received from the active supervisor

iii. The beacon time-out timer (from the active supervisor) expires on one port

iv. A beacon frame from a different supervisor with higher precedence and the precedence of this supervisor is higher.

D. From IDLE_STATE:

A beacon frame is received from any supervisor on one port

The FAULT_STATE (backup) provides partial ring support, but the ring is still not fully operative in FAULT_STATE (backup).

The following condition will cause a transition to the FAULT_STATE (active):

i. The beacon time-out timer (from the active supervisor) expires on both ports

The following condition will cause a transition to the NORMAL_STATE (backup):

ii. Beacon frames from the active supervisor are received on both ports and a beacon frame with RING_NORMAL_STATE has been received.

The following condition will cause a transition to the IDLE_STATE:

iii. The beacon time-out timer (from the active supervisor) expires on both ports

- NORMAL_STATE (backup)

  The supervisor reaches the NORMAL_STATE (backup) only after the following condition:

  - Beacon frames from the active supervisor are received on both ports and a beacon frame with RING_NORMAL_STATE has been received.

  The NORMAL_STATE (backup) provides full ring support. The following conditions will cause a change to the FAULT_STATE (backup):

  A. A link failure has been detected.

  B. A beacon frame with RING_FAULT_STATE has been received from the active supervisor on at least one port.

  C. The beacon time-out timer (from the active supervisor) expires on both ports.

  D. A beacon frame from a different supervisor with higher precedence and the precedence of this supervisor is higher.

- IDLE_STATE

  The IDLE_STATE is the state which is reached after power-on if the supervisor has not been configured as supervisor.

  In IDLE_STATE the network operates as linear network, there is no ring support active. If on one port a beacon frame from a supervisor is received, the state changes to FAULT_STATE (backup).

For more details refer to the DLR specification in reference [8], section “9-5 Device Level Ring”.

2.12 CIP Device protection

2.12.1 Introduction

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

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

You will find the complete description of device protection mode in the **CIP specification, Volume 1, Chapter 5A-2** - **Identity object**.

2.12.2 Protection modes

The protection mode maps to attribute 19 of the Identity object. Table 64 and 65 describe this attribute in detail.

CIP defines two different protection modes, as follows:

- **Implicit protection** is enabled implicitly when at least one active I/O connection is established with the device. As soon as the last I/O connection closes, the implicit protection mode is disabled. It cannot be modified programmatically.
- **Explicit protection** may be set explicitly by the application on demand by means of the bit `CIP_ID_PROTECTION_MODE_EXPLICIT_PROTECTION` in the attribute's value.

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

2.12.3 Protection Policy

Enabled device protection, regardless of whether it is due to implicit or explicit protection (or both), has the following effects:

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

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

Class	Instace 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
Ethernet Link (0xF6)	1, 2	6	Interface Control
		9	Admin State
		768	Mdix Config
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 676: Hilscher's default protection policy

The user application can modify the protection policy. Therefore, the services "Get Attribute Option" and "Set Attribute Option" allow setting and clearing of the attribute option flag CIP_FLG_TREAT_PROTECTED of a particular attribute in order to set it protected or unprotected.

2.13 QuickConnect

This version of the protocol stack supports QuickConnect, which can be broken down into the following technical goals and measures.

The main goal of QuickConnect is to allow an I/O connection to be established quickly after the device is powered on.

From the specification:

> After power-on, a connection with the Adapter shall be established in 500ms or less. The device shall be ready to accept TCP connections in less than 350ms after power on.

Thus, with QuickConnect, the network PHYs are enabled early during system start, since the time to link is a key element, and they are enabled with fixed duplex modes and MAU types, skipping costly automatic detection.

From the specification:

> QuickConnect devices shall implement the ability to set forced speed/duplex mode (for 10/100 Mb Ethernet at least), via the Ethernet Link Object. On devices with 2 external Ethernetports:
>
> a. The labels for the 2 ports shall include an ordinal indication (e.g.: Port ½)
> b. The port with the lower ordinal indication shall be configured as MDI.
> c. The port with the upper ordinal indication shall be configured as MDIX.

Another feature that limits the time-to-network after power on, is the ACD two-second-long probing stage in which the device figures out whether the to-be-applied IP address is safe to use without any collision. With QuickConnect enabled, the protocol stack will thus skip the ACD probing stage.

From the specification:

> With QuickConnect, the address conflict detection (ACD) mechanism will have to change its behavior and just emit up to 40 ARP announces to the network, one each 25ms, or go into ongoing detection phase immediately as soon as the I/O connection is established

Also, support of the QuickConnect feature needs to be resembled in a devices EDS file, by adding the specified keywords and values. Please refer to the CIP specification for any details on this.

QuickConnect is reflected and controlled by attribute 12 of the TCP/IP object. This attribute is remanently stored (see section *Remanent data* on page 88).

The application configures for QuickConnect depending on the Configuration Packet Set:

- For the Basic Configuration Set, the attribute bQuickConnectFlags provides two bits which control the QuickConnect configuration. Please refer to Table 79.
- For the Extended Configuration Set, the application has to manually enable the attribute (see section 2.4.1.1 “Attribute Option Flags”) 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 whatsoever current device settings. When QuickConnect is disabled, the previous settings will get active again.

While QuickConnect is active, changes to EtherNetLink object's attribute 6 are rejected.

3 Getting Started / Configuration

3.1 Configuration Methods

The EtherNet/IP Adapter stack requires configuration parameters. The 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* on page 80).
2. The application can set the configuration parameters using the Extended Configuration Packet Set (see section *Configuration using the Packet API* on page 80).
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. The protocol stack firmware loads this database in order to configure the protocol stack.

3.2 Host Application Behavior

The following diagram gives an overview of how the host application shall behave in different scenarios.

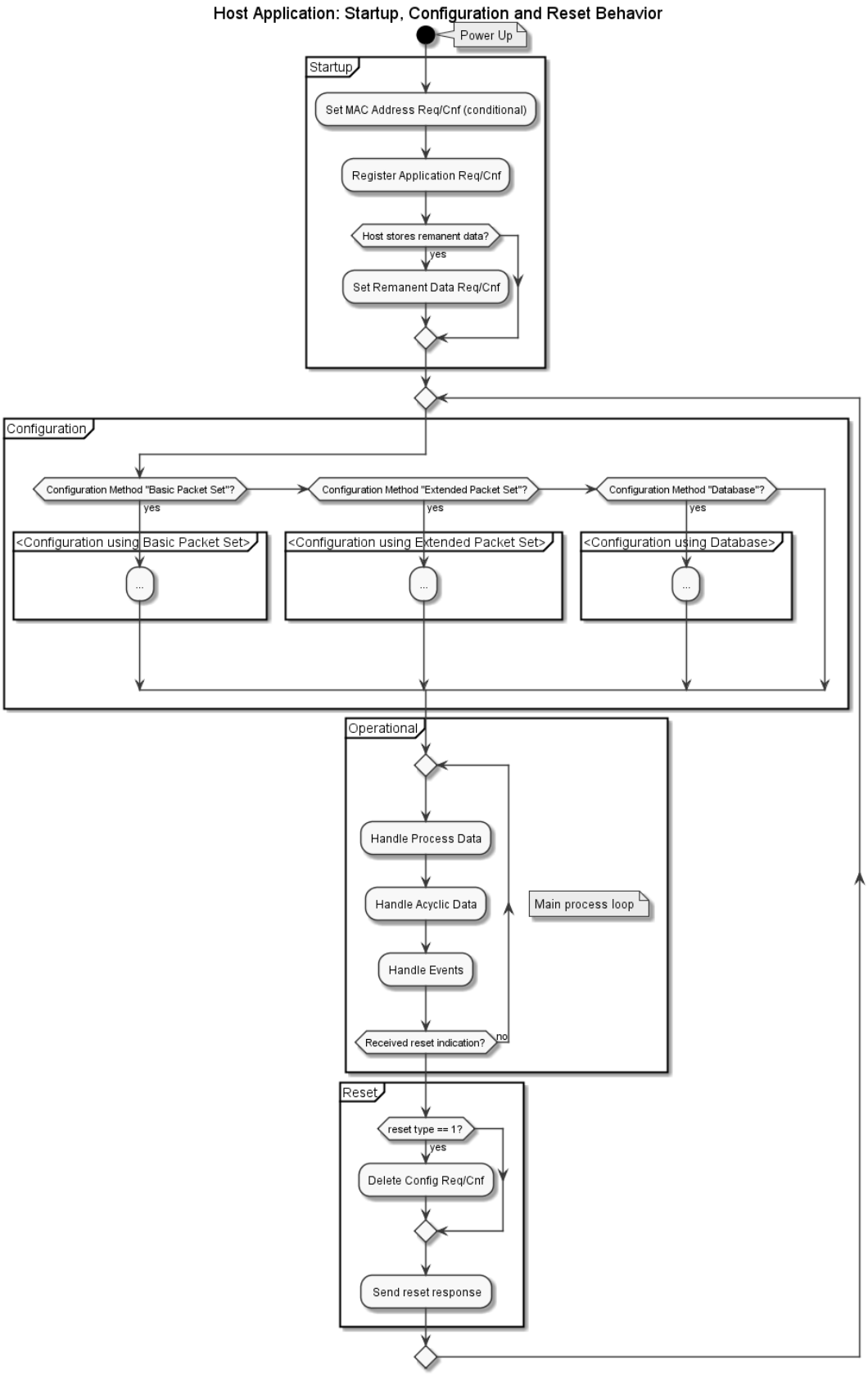


Figure 7: Host Application: Startup, Configuration and Reset Behavior

3.2.1 Startup

Note: Setting the MAC address is conditional. For more information, see section *Ethernet MAC Address* on page 53.

Note: Setting remanent data is conditional. For more information, see section *Remanent data* on page 88.

3.2.2 Operational

In the operational state, the host application enters its main process loop. This includes IO data handling, protocol stack event handling.

3.2.3 Configuration

The configuration behavior depends on the chosen configuration method (see 3.1). The different configuration sequences are illustrated in the corresponding sections that talk about the specific configuration methods.

1. See section *Configuration Sequence* (on page 82) using the Basic Configuration Packet Set.
2. See section *Configuration Sequence* (on page 85) using the Extended Configuration Packet Set.
3. See section ***Configuration sequence*** (on page **87**) using SYCON.net.

3.2.4 Reset

The reset behavior is independent of the chosen configuration method. For more information regarding the “Reset Indication” and “Delete Config” handling, see section 4.2.2 “Indication of a Reset Request from the network” and 4.3.5 “Delete Configuration”.

3.3 Configuration using the Packet API

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

In section *Hilscher EtherNet/IP Stack Capabilities* on page 15 the default Hilscher CIP Object Model has been described. 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 68 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 Device Level Ring (DLR) protocol. ▪ Support of ACD (Address Conflict Detection) ▪ Implementation of additional CIP objects, which might be mandatory when using a special CIP Profile. These objects are also accessible via acyclic/explicit messages. With the Basic Configuration Packet Set, a default CIP object model as illustrated in Figure 3 is established. If your device needs advanced functionality that is not covered by the basic Configuration Set, please use the Extended Configuration Set described below. Limitations when using this configuration packet set: ▪ Not more than 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 device's CIP object model in all aspects. In addition to the functionality available through the Basic Configuration Packet Set, this extended configuration variant allows: ▪ Up to 32 assembly instances. Default is 10. This also includes configuration assembly instances. ▪ Optional configuration assemblies (necessary if the device needs configuration parameters from the Scanner/Master/PLC before going into cyclic communication). ▪ CIP Sync is supported. Therefore, the CIP Time Sync object needs to be activated (see 4.1.4 Register an additional Object Class). The Extended Configuration allows establishing configurations, which are a superset of those that can be established with the Basic Configuration Packet Set.

Table 68: 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)	Page
Register/Unregister Application	0x2F10/0x2F11	125
Set Configuration Parameters	0x3612/0x3613	95
Set Remanent Data Request (conditional)	0x2F8C/0x2F8D	166
Channel Init	0x2F80/0x2F81	125

Table 69: Basic Configuration Packet Set - Configuration Packets

3.3.1.2 Optional Request Packets

In addition to the request packets, which are required for configuration, the application can optionally issue the following requests during the configuration phase. When your application uses these optional packets, we recommend Application Controlled Start as configurable per member `ulSystemFlags` of request packet `EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ`.

Packet name	Page
Set Parameter Flags	104
Get Module Status/ Network Status	164
Register an additional Object Class	108
Register Service	117
Set Parameter	119

Table 70: Additional Request Packets Using the Basic Configuration Packet Set

3.3.1.3 Indication Packets the Host Application Needs 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)	Page
Indication of a Reset Request from the network	0x1A24/0x1A25	127
Connection State Change Indication	0x1A2E/0x1A2F	129
Acyclic Data Transfer	0x1A3E/0x1A3F	137
CIP Object Change Indication	0x1AFA/0x1AFB	143
Store Remanent Data Indication (conditional)	0x2F8E/0x2F8F	161

Table 71: Indication Packets Using the Basic Configuration Packet Set

3.3.1.4 Configuration Sequence

Figure 8 below illustrates the configuration packet sequence when using the Basic Configuration Packet Set. See also 3.2 “Host Application Behavior” for more information on how the configuration sequence is integrated into the host application’s general behavior.

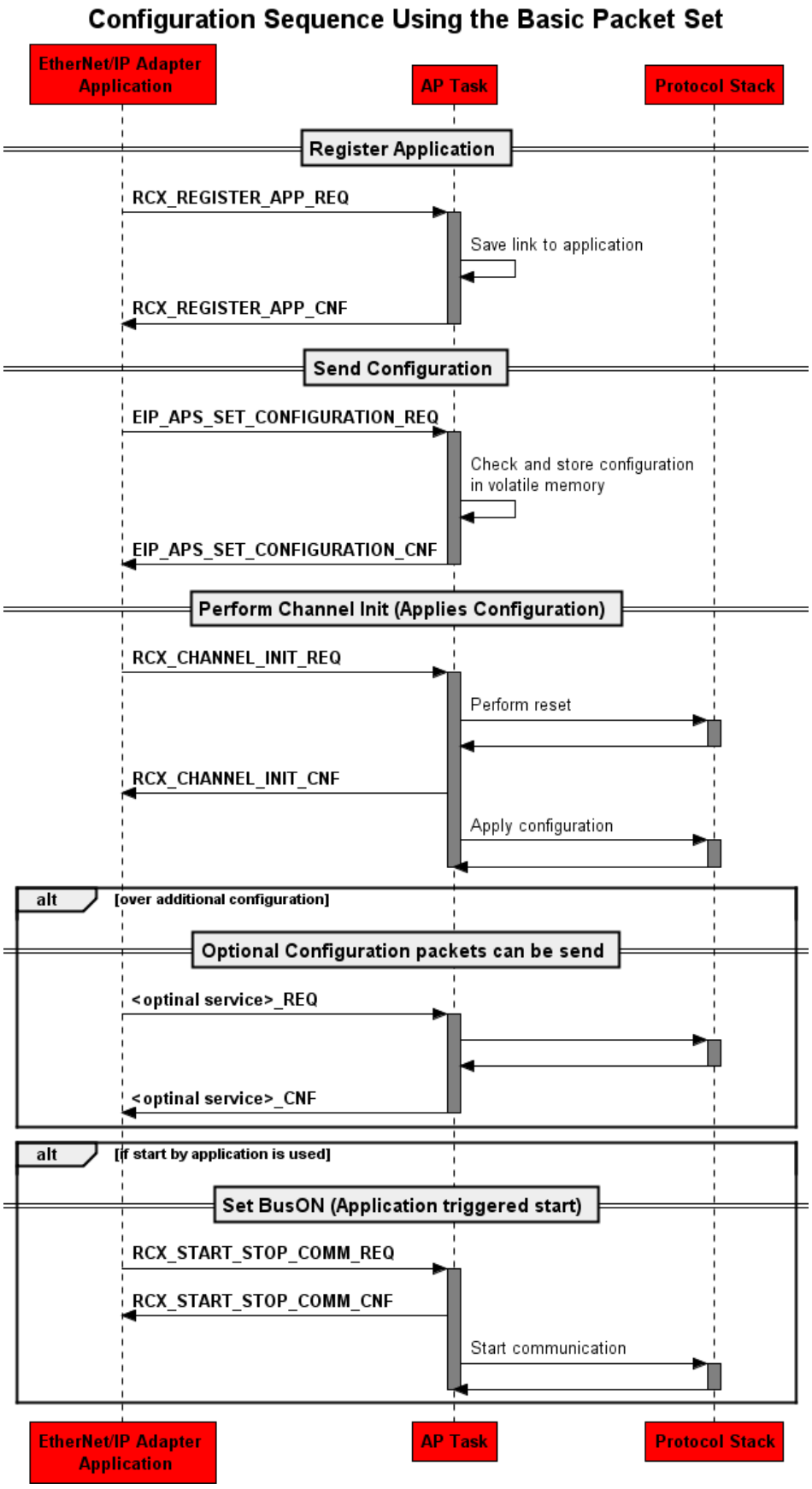


Figure 8: Configuration Sequence Using the Basic Configuration Packet Set

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)	Page
Register/Unregister Application	0x2F10/0x2F11	125
CIP Service Request	0x1AF8/0x1AF9	119
Register a new Assembly Instance	0x1A0C/0x1A0D	111
Finish configuration of CIP Objects	0x3614/0x3615	106
Set Remanent Data Request (conditional)	0x2F8C/0x2F8D	166

Table 72: Extended Configuration Packet Set - Configuration Packets

3.3.2.2 Optional Request Packets

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

Packet name	Command code (REQ/CNF)	Page
Set Parameter Flags	0x360A/0x360B	104
Get Module Status/ Network Status	0x360E/0x360F	164
Register an additional Object Class	0x1A02/0x1A03	108
Register Service	0x1A44/0x1A45	117

Table 73: Additional Request Packets Using the Basic Configuration Packet Set

3.3.2.3 Indication Packets the Host Application Needs 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)	Page
Indication of a Reset Request from the network	0x1A24/ 0x1A25	127
Connection State Change Indication	0x1A2E/ 0x1A2F	129
Acyclic Data Transfer	0x1A3E/ 0x1A3F	137
CIP Object Change Indication	0x1AFA/ 0x1AFB	143
Store Remanent Data Indication (conditional)	0x2F8E/ 0x2F8F	161

Table 74: Indication Packets Using the Extended Configuration Set

3.3.2.4 Configuration Sequence

Figure 9 below illustrates the configuration packet sequence when using the Extended Configuration Packet Set. See also section *Host Application Behavior* (page 78) for more information on how the configuration sequence is integrated into the host application's general behavior.

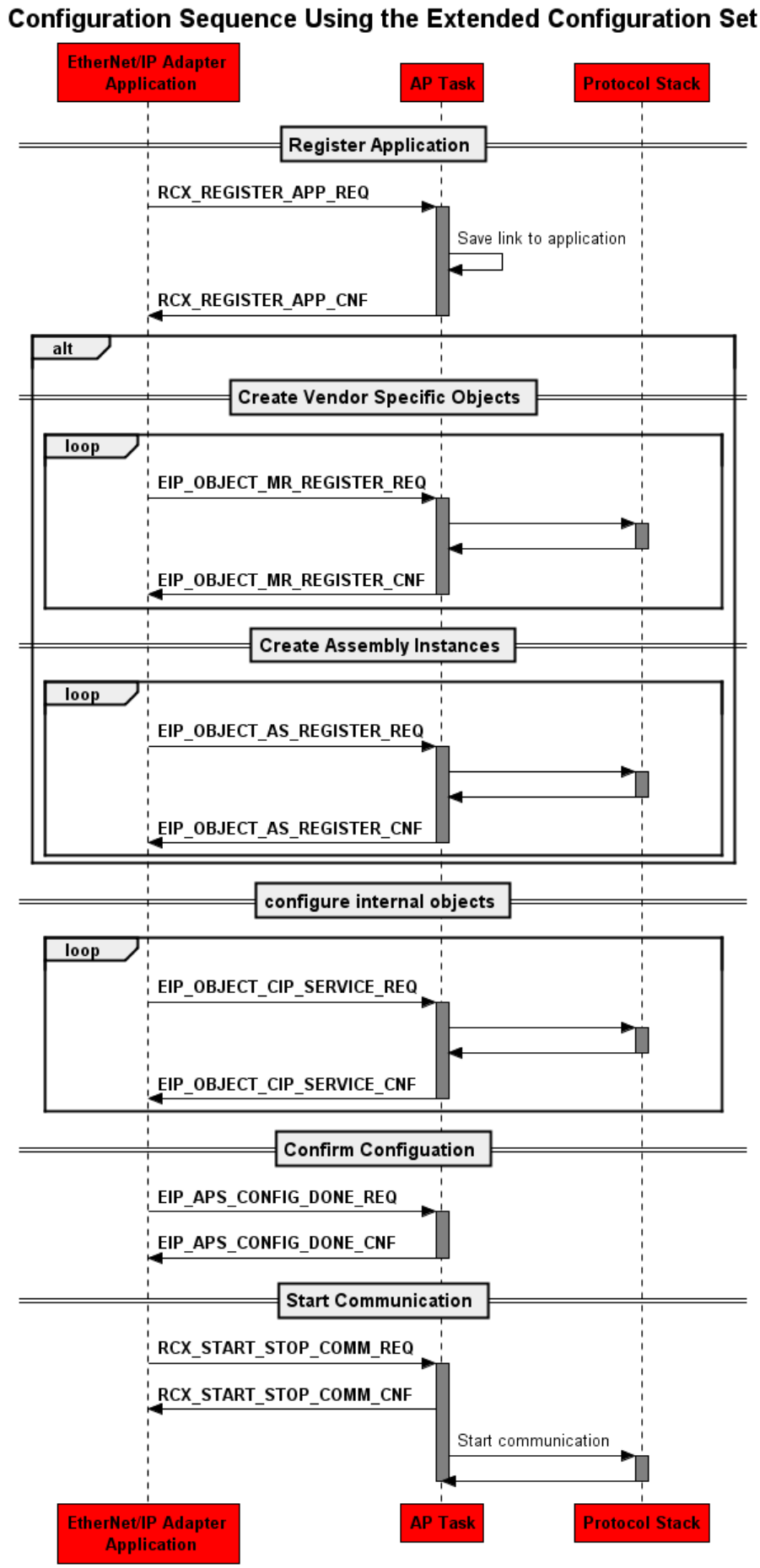


Figure 9: Configuration Sequence Using the Extended Configuration Set

3.4 Configuration Using Sycon.NET

3.4.1 Configuration sequence

Figure 10 below illustrates the configuration sequence when using the data base configuration. See also 3.2 “Host Application Behavior” for more information on how the configuration sequence is integrated into the host application’s general behavior.

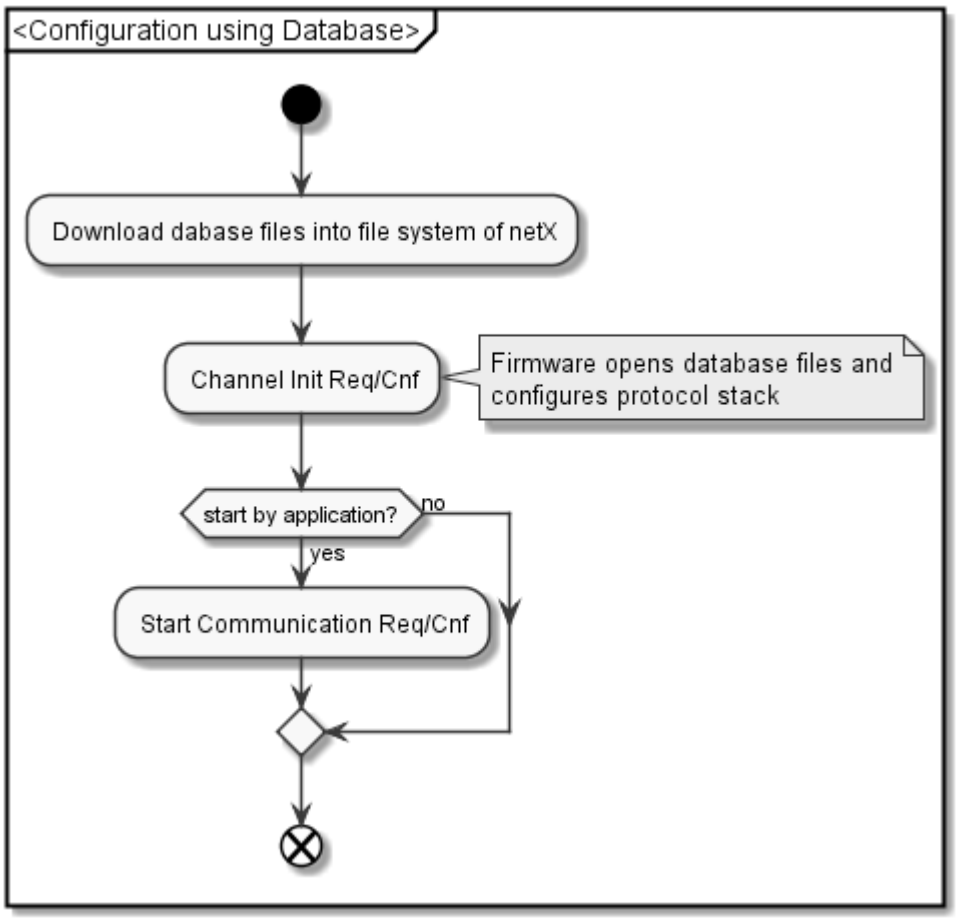


Figure 10: Configuration Sequence Using the Database

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 through 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 either volatile, meaning that their value will not be retained over power cycles, or non-volatile, in which case 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.

Modification of non-volatile attributes over either the network or the host interface causes a Flash write access. On device reconfiguration, the previously stored state of these attributes is applied on top of the configuration of the device.

The CIP Identity object offers a Reset Service which allows the device configuration to be reset to the ***factory default configuration***. This basically is implemented by deleting the remanent data prior to the reset. Remanent data can be deleted with the command `HIL_DELETE_CONFIG_REQ`.

3.5.2 Remanent Data Responsibility

When you design your application, you have to decide whether

- the **protocol stack** stores the remanent data or
- the **application** stores the remanent data

If the system designer decides for application-side storage of remanent data, then the firmware's taglist must be modified as described in section 5.

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 which also bear non-volatile attributes, they will have to be handled completely in the scope of the host application. The latter case is not supported by mechanisms of the stack and thus 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 image. This is the default setting in a firmware.
Application	The application stores the remanent data In case 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 *Set Remanent Data* service (`HIL_SET_REMANENT_DATA_REQ`) and to support the *Store Remanent Data* service (`HIL_STORE_REMAMENT_DATA_IND`). **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’s taglist. **Configuration** The application has to use the *Set Remanent Data* service (`HIL_SET_REMANENT_DATA_REQ`) to provide the remanent data any time the host application starts up for the first time (e.g. when coming from power up) and before the application sends any other service request (except for HIL_REGISTER_APP_REQ). For a state diagram, see section *Host Application Behavior* on page 78. **During runtime** The stack component indicates to the application the *Store Remanent Data* service (`HIL_STORE_REMAMENT_DATA_IND`) each time remanent data has been changed. The stack component provides the remanent data as a block towards the application. The application has to store the remanent data with each indication.

Table 75: 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 no valid such BLOB is available, the remanent data counts as unavailable/deleted. Figure 11: Remanent data state transitions illustrates this.

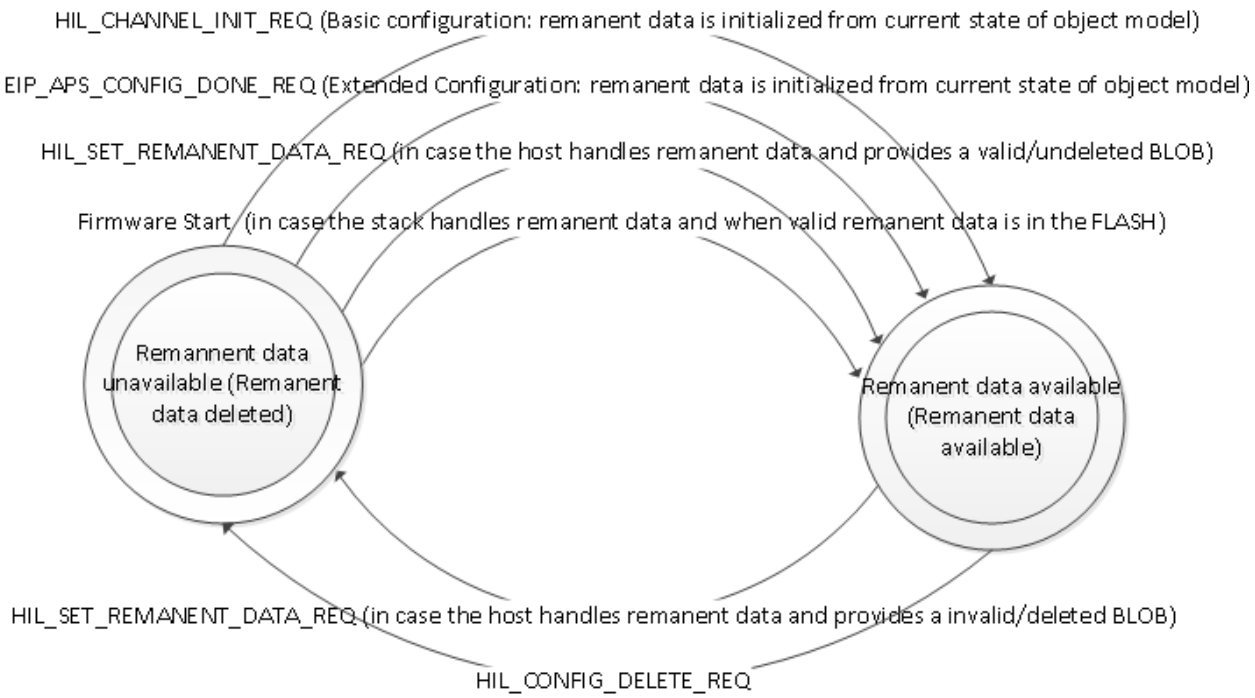


Figure 11: Remanent data state transitions

3.5.4 Remanent Data Flow

When using the Basic Configuration Packet Set the protocol stack is initialized primarily with the two packets EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ/CNF and HIL_CHANNEL_INIT_REQ/CNF. This leads to a data flow of configuration data and remanent data as illustrated in Figure 12: Remanent Data Flow with Basic Configuration Packets.

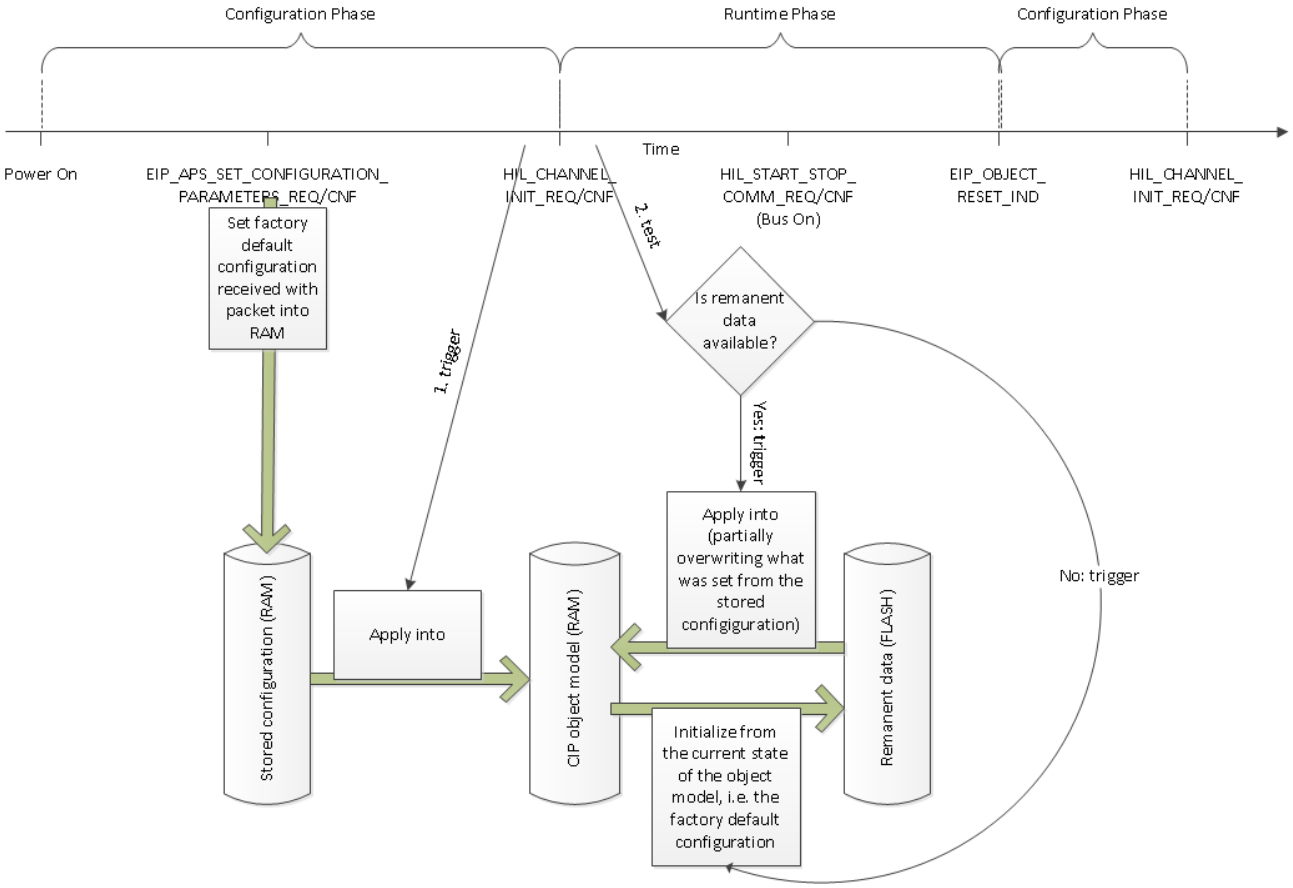


Figure 12: Remanent Data Flow with Basic Configuration Packets

With the Extended Configuration Packet Set, EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ/CNF is not used and thus the stored configuration is pointless in this scenario. Instead, the factory default configuration is set through CIP_SERVICE_REQ/CNF, Set_Attribute_Single directly into the object model. The EIP_APS_CONFIG_DONE_REQ/CNF is used instead of the HIL_CHANNEL_INIT_REQ to trigger the application of remanent data on top of these factory defaults.

The HIL_CHANNEL_INIT_REQ/CNF instead has a different purpose in case no stored configuration is available: It will reset the protocol stack into its initial state, so that a fresh (factory default) configuration can be set. Figure 13: Remanent Data Flow with Extended Configuration Packets illustrates this.

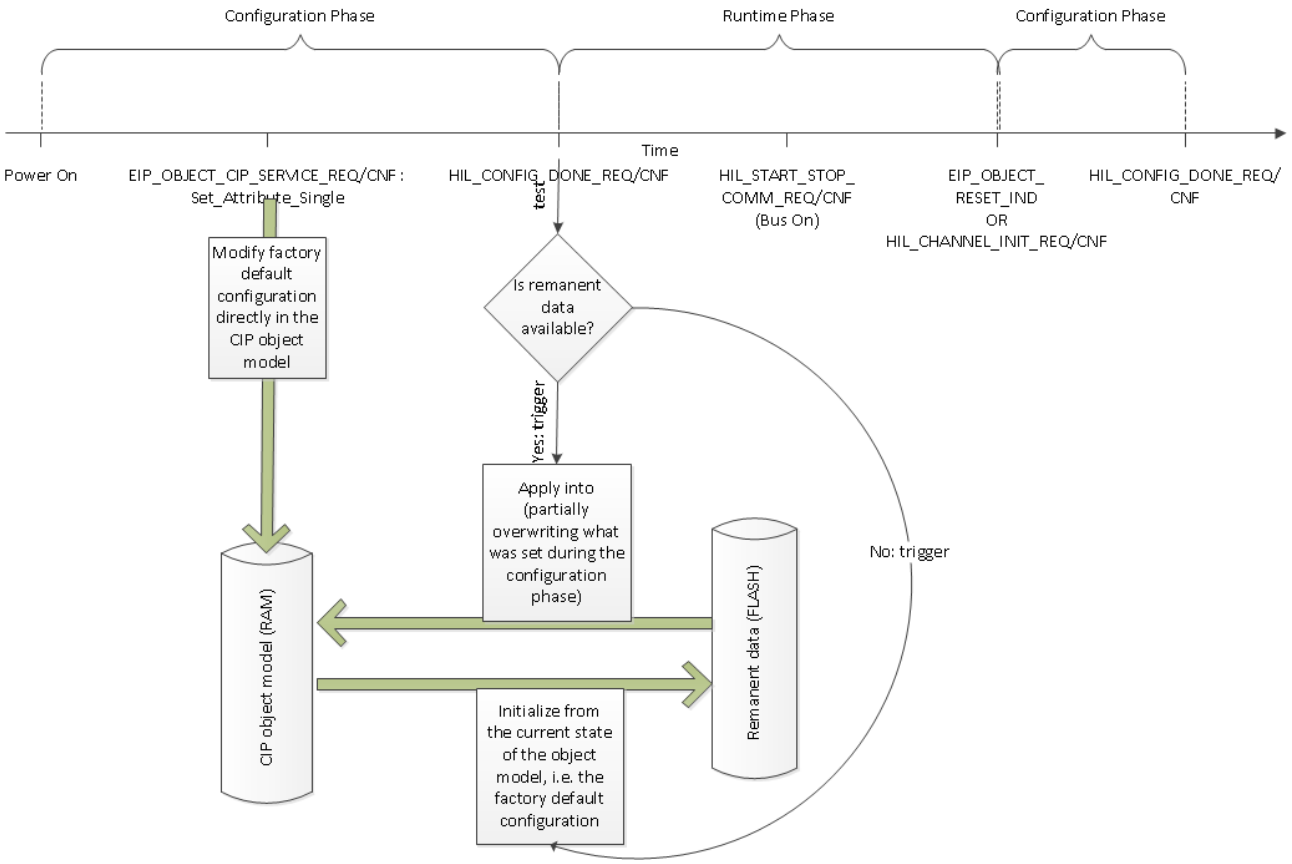


Figure 13: Remanent Data Flow with Extended Configuration Packets

What is not depicted in the above figures, but has to be emphasized for both configuration packet sets, is that there is a difference in behavior of the EIP_OBJECT_CIP_SERVICE_REQ/CNF: 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	Setting of attributes is manifested in the object model only. Thus, it alters the factory default configuration. With the basic configuration packet set, this may be overwritten later with the stored configuration. With both configuration packet sets, this may be overwritten later with the remanent data.
Runtime phase	Setting of attributes is manifested in the object model and in the remanent data. Thus, it alters the device runtime configuration.

3.5.5 Remanent Data Content

All implemented attributes of built-in CIP objects, which, according to the CIP specification, are to be kept non-volatile, contribute to the remanent data. These are:

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

Table 76: Remanently stored CIP attributes

Note: This table is just for informational purpose. In case 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.

4 Application Interface

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

4.1 Configuring the EtherNet/IP Adapter

This chapter explains the packets used for configuring the EtherNet/IP Adapter using the DPM/Packet Interface. Section 3.3 explains some details about the configuration sequence.

The following packets are available for the configuration:

Packet	Command code (REQ)	Page
EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ	0x00003612	95
EIP_APS_SET_PARAMETER_REQ	0x0000360A	104
EIP_APS_CONFIG_DONE_REQ	0x00003614	165
EIP_OBJECT_MR_REGISTER_REQ	0x00001A02	108
EIP_OBJECT_AS_REGISTER_REQ	0x00001A0C	111
EIP_OBJECT_REGISTER_SERVICE_REQ	0x00001A44	117
EIP_OBJECT_CIP_SERVICE_REQ	0x00001AF8	119
HIL_SET_WATCHDOG_TIME_REQ	0x00002F04	125
HIL_REGISTER_APP_REQ	0x00002F10	125
HIL_START_STOP_COMM_REQ	0x00002F30	125
HIL_CHANNEL_INIT_REQ	0x00002F80	125

Table 77: Overview: Configuration packets of the EtherNet/IP Adapter

4.1.1 Set Configuration Parameters

The host application uses this service in order to configure the device with configuration parameters. This packet is part of the Basic Configuration Set and provides a basic configuration to all default CIP objects within the stack.

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 dual port memory (relative offset to the base addresses of the input and output areas of the DPM).

Note: If you set the Revision Information or the Product Name to zero or the empty string, respectively, the protocol stack will apply default Hilscher-specific values.

Note: If you set the Serial Number to zero, the protocol stack will apply the device specific information from the Security Memory or FDL, if available.

On `EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ`, the EtherNet/IP Adapter protocol stack will:

- Test the “configuration locked” condition and eventually reject the request (see 4.3.6).
- Perform consistency and integrity checks on the received configuration and reject it on errors.
- On success, render the selected configuration pending and to be applied on the next channel initialization (`HIL_CHANNEL_INIT_REQ`).

Note, that 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 Dual-Port-Memory Manual in order to receive indication packets from the netX (see section 4.1.10).

Note: Only the parameter set V3 (and newer) is supported. The stack will reject any older version with error code `ERR_HIL_INVALID_PARAMETER` (0xC0000009)

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST  EIP_DPMINTF_QOS_CONFIG_Ttag
{
  uint32_t    ulQoSFlags;
  uint8_t     bTag802Enable;        /* QoS Attribute 1 */
  uint8_t     bDSCP_PTP_Event;      /* QoS Attribute 2 */
  uint8_t     bDSCP_PTP_General;    /* QoS Attribute 3 */
  uint8_t     bDSCP_Urgent;         /* QoS Attribute 4 */
  uint8_t     bDSCP_Scheduled;      /* QoS Attribute 5 */
  uint8_t     bDSCP_High;           /* QoS Attribute 6 */
  uint8_t     bDSCP_Low;            /* QoS Attribute 7 */
  uint8_t     bDSCP_Explicit;       /* QoS Attribute 8 */
} EIP_DPMINTF_QOS_CONFIG_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_DPMINTF_TI_ACD_LAST_CONFLICT_Ttag
{
  uint8_t    bAcdActivity;      /*!< State of ACD activity when last
                                    conflict detected */

  uint8_t    abRemoteMac[6];    /*!< MAC address of remote node from
                                    the ARP PDU in which a conflict was
                                    detected */

  uint8_t    abArpPdu[28];      /*!< Copy of the raw ARP PDU in which
                                    a conflict was detected. */
} EIP_DPMINTF_TI_ACD_LAST_CONFLICT_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_DPMINTF_TI_MCAST_CONFIG_Ttag
{
  uint8_t    bAllocControl;     /* Multicast address allocation control
                                   word. Determines how addresses are
                                   allocated. */
  uint8_t    bReserved;
  uint16_t   usNumMCast;        /* Number of IP multicast addresses
                                   to allocate for EtherNet/IP */
  uint32_t   ulMcastStartAddr;  /* Starting multicast address from which */
} EIP_DPMINTF_TI_MCAST_CONFIG_T;

/*############################################################################################*/

#define EIP_APS_CONFIGURATION_PARAMETER_SET_V3 3

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_APS_CONFIGURATION_PARAMETER_SET_V3_Ttag
{

  uint32_t  ulSystemFlags;
  uint32_t  ulWdgTime;
  uint32_t  ulInputLen;
  uint32_t  ulOutputLen;
  uint32_t  ulTcpFlag;
  uint32_t  ulIpAddr;
  uint32_t  ulNetMask;
  uint32_t  ulGateway;
  uint16_t  usVendId;
  uint16_t  usProductType;
  uint16_t  usProductCode;
  uint32_t  ulSerialNumber;
  uint8_t   bMinorRev;
  uint8_t   bMajorRev;
  uint8_t   abDeviceName[32];
  uint32_t  ulInputAssInstance;
  uint32_t  ulInputAssFlags;
  uint32_t  ulOutputAssInstance;
  uint32_t   ulOutputAssFlags;
  EIP_DPMINTF_QOS_CONFIG_T tQoS_Config;
  uint32_t   ulNameServer;
  uint32_t   ulNameServer_2;
  uint8_t    abDomainName[48 + 2];
  uint8_t    abHostName[64+2];
  uint8_t    bSelectAcd;
  EIP_DPMINTF_TI_ACD_LAST_CONFLICT_T  tLastConflictDetected;
  uint8_t                            bQuickConnectFlags;
  uint8_t                            abAdminState[2];
  uint8_t                            bTTLValue;
  EIP_DPMINTF_TI_MCAST_CONFIG_T       tMCastConfig;
  uint16_t                           usEncapInactivityTimer;
} EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T;
```

```
#define EIP_APS_CONFIGURATION_PARAMETER_SET_V3_SIZE
(sizeof(EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T) )
/*****************************************************************************/

/* Request Packet */

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ_Ttag
{
  uint32_t  ulParameterVersion;  /*!< Version related to the following configuration union,
                                      only EIP_APS_CONFIGURATION_PARAMETER_SET_V3 is supported.
                                 */

  __HIL_PACKED_PRE union __HIL_PACKED_POST
  {
#if 0
    /* Parameter sets V1 and V2 are no longer supported with
     * EtherNet/IP Adapter v3.4.1.0 and newer versions
     */
    EIP_APS_CONFIGURATION_PARAMETER_SET_V1_T tV1;
    EIP_APS_CONFIGURATION_PARAMETER_SET_V2_T tV2;
#endif
    EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T tV3;
  } unConfig;
} EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ_Ttag
{
  HIL_PACKET_HEADER_T                       tHead;
  EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ_T tData;

} EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ_T;
```

Packet description

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x20	Destination
ulLen	uint32_t	EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ_SIZE plus EIP_APS_CONFIGURATION_PARAMETER_SET_V3_SIZE	Packet Data Length in bytes
ulSta	uint32_t	0	See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x3612	EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ - Command
Data			
ulParameterVersion	uint32_t	3 (latest version)	Version of the following parameter structure
unConfig.tV3	union		See Table 79

Table 78: EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ – Set Configuration Parameters Request

Structure EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T			
ulSystemFlags	uint32_t (Bit field)	0, 1	System flags area The start of the device can be performed either application controlled or automatically: Automatic (0): Network connections are opened automatically without taking care of the state of the host application. Communication with a controller after a device start is allowed without BUS_ON flag, but the communication will be interrupted if the BUS_ON flag changes state to 0 Application controlled (1): The channel firmware is forced to wait for the host application to wait for the Application Ready flag in the communication change of state register. Communication with controller is allowed only with the BUS_ON flag. For more information, concerning 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 80 on page 102. Default value: 0x00000410 (both ports set to DHCP + Autoneg)
ulIPAddr	uint32_t	All valid IP-addresses Default: 0.0.0.0	IP Address
ulNetMask	uint32_t	All valid masks Default: 0.0.0.0	Network Mask
ulGateway	uint32_t	All valid IP-addresses Default: 0.0.0.0	Gateway Address
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 - 0x64 Vendor specific: 0x64 - 0xC7 Reserved by CIP: 0xC8 - 0xFF Publicly defined: 0x100 - 0x2FF Vendor specific: 0x300 - 0x4FF Reserved by CIP: 0x500 - 0xFFFF Default: 0x0C (Communication Device)
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 2.6.1 for details.
bMinorRev	uint8_t	1..255	Minor revision
bMajorRev	uint8_t	1..127	Major revision
abDeviceName	uint8_t [32]		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”
ulInputAssInstance	uint32_t	1 … 0xFFFFFFFE Default: 100	Instance number of input assembly (O→T direction) See Table 90 on page 111. **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	uint8_t	Bit mask	Input assembly (O→T) flags See Table 92 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 90 on page 111. **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	uint8_t	Bit mask	Output assembly (T→O) flags See Table 92 for a description of available Assembly flags.
tQoS_Config	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)* on page 37 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)* on page 37 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)* on page 37 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)* on page 37 for more information.
bSelectAcd	uint8_t		Select ACD This parameter configures attribute 10 of the TCP/IP Interface Object. See section *TCP/IP Interface Object (Class Code: 0xF5)* on page 37 for more information.
tLastConflictDetected	EIP_DPMINTF_TI_ACD_LAST_CONFLICT_T		Last Detected Conflict This parameter configures attribute 11 of the TCP/IP Interface Object. See section *TCP/IP Interface Object (Class Code: 0xF5)* on page 37 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)* on page 37 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 actual 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)* on page 40 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	0-3600 Default: 120 seconds	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 non-volatile memory.
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 79: EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ – Configuration Parameter Set V3

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

Bits	Description
31 ... 29	Reserved for future use
28	Speed Selection (Ethernet Port 2): Only evaluated if bit 15 is set. Behaves the same as bit 12.
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
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
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
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 80: Available TCP flags in bit field `ulTcpFlag` of the Basic Configuration Packet

Packet Structure Reference

```
/* Confirmation Packet */

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST
EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_CNF_Ttag
{
  HIL_PACKET_HEADER_T                       tHead;
} EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulLen	UINT32	0	Packet Data Length in bytes
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x3613	EIP_APS_SET_CONFIGURATION_PARAMETERS_CNF - Command

Table 81: EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_CNF – Set Configuration Parameters Confirmation

4.1.2 Set Parameter Flags

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.

Table 82 shows all available flags:

Bit	Description
0	Flag IP_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 (see section *Module and Network Status* on page 58 for more information). When enabled, the protocol stack generates indication packets *Module Network Status Change* as described on page 147 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.

Table 82: EIP_APS_SET_PARAMETER_REQ Flags

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_APS_SET_PARAMETER_REQ_Ttag
{
  uint32_t  ulParameterFlags;    /*!< Parameter flags \n
} EIP_APS_SET_PARAMETER_REQ_T;

#define EIP_APS_SET_PARAMETER_REQ_SIZE (sizeof(EIP_APS_SET_PARAMETER_REQ_T))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_APS_PACKET_SET_PARAMETER_REQ_Ttag
{
  HIL_PACKET_HEADER_T             tHead;
  EIP_APS_SET_PARAMETER_REQ_T     tData;
} EIP_APS_PACKET_SET_PARAMETER_REQ_T;
```

Packet description

Variable	Type	Value / Range	Description
structure **HIL_PACKET_HEADER_T**			
ulDest	uint32_t	0x20	Destination
ulLen	uint32_t	4	Packet Data Length in bytes
ulSta	uint32_t	0	See chapter Status/Error Codes Overview*Status/Error Codes Overview*
ulCmd	uint32_t	0x360A	EIP_APS_SET_PARAMETER_REQ - Command
structure **EIP_APS_SET_PARAMETER_REQ_T** tData			
ulParameterFlags	uint32_t	See Table 82 for possible values	Bit field

Table 83: EIP_APS_SET_PARAMETER_REQ – Set Parameter Flags Request

Packet Structure Reference

```
#define EIP_APS_SET_PARAMETER_CNF_SIZE 0

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_APS_PACKET_SET_PARAMETER_CNF_Ttag
{
  HIL_PACKET_HEADER_T tHead;
} EIP_APS_PACKET_SET_PARAMETER_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
structure HIL_PACKET_HEADER_T tHead			
ulLen	uint32_t	0	Packet Data Length in bytes
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x360B	EIP_APS_SET_PARAMETER_CNF - Command

Table 84: EIP_APS_SET_PARAMETER_CNF – Confirmation to Set Parameter Flags Request

4.1.3 Finish configuration of CIP Objects

The packet `EIP_APS_PACKET_CONFIG_DONE_REQ_T` 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.

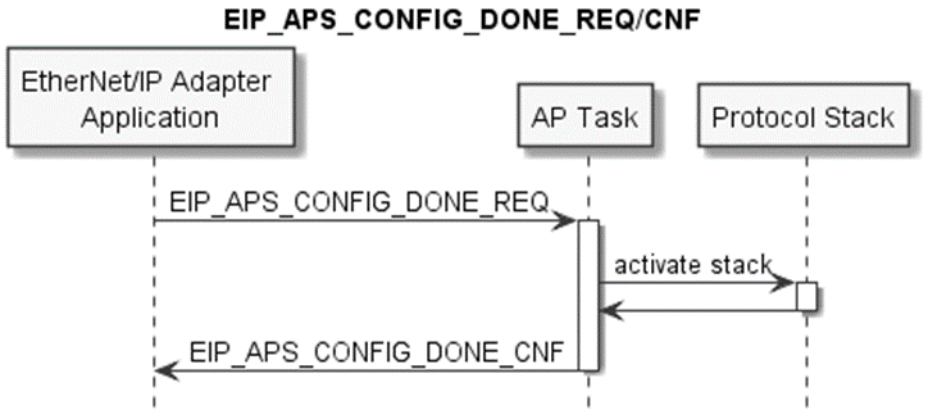


Figure 14: Sequence Diagram for the EIP_APS_CONFIG_DONE_REQ/CNF Packet

Packet Structure Reference

```
#define EIP_APS_CONFIG_DONE_REQ_SIZE       0

typedef struct EIP_APS_PACKET_CONFIG_DONE_REQ_Ttag
{
  HIL_PACKET_HEADER_T             tHead;
} EIP_APS_PACKET_CONFIG_DONE_REQ_T;
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulDest	uint32_t	0x20	Destination
ulLen	uint32_t	0	Packet Data Length in bytes
ulSta	uint32_t	0	See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x3614	EIP_APS_CONFIG_DONE_REQ - Command

Table 85: EIP_APS_CONFIG_DONE_REQ – Signal end of configuration request

Packet Structure Reference

```
#define EIP_APS_CONFIG_DONE_CNF_SIZE 0

typedef struct EIP_APS_PACKET_CONFIG_DONE_CNF_Ttag
{
  HIL_PACKET_HEADER_T      tHead;
} EIP_APS_PACKET_CONFIG_DONE_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulLen	uint32_t	0	Packet Data Length in bytes
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x3615	EIP_APS_CONFIG_DONE_CNF - Command

Table 86: EIP_APS_CONFIG_DONE_CNF – Confirmation of end of configuration Request

4.1.4 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 3 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.
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` (section *Acyclic Data Transfer* on page 137).

The class code parameter `ulClass` uniquely identifies the object class. According to the CIP specification Vol. 1 chapter 5, the overall range of class codes splits into certain ranges as illustrated in Table 87:

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 87: 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.

Packet Structure Reference

```
typedef enum EIP_OBJECT_MR_REGISTER_OPTION_FLAGS_Etag
{
EIP_OBJECT_MR_REGISTER_OPTION_FLAGS_USE_OBJECT_PROVIDED_BY_STACK = 1,
} EIP_OBJECT_MR_REGISTER_OPTION_FLAGS_E;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_MR_REGISTER_REQ_Ttag
{
  uint32_t ulReserved1;              /*!< Reserved, set to 0x00 */
  uint32_t ulClass;                 /*!< Object class identifier
  uint32_t ulOptionFlags;           /*!< Option flags
} EIP_OBJECT_MR_REGISTER_REQ_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_MR_PACKET_REGISTER_REQ_Ttag
{
  HIL_PACKET_HEADER_T           tHead;
  EIP_OBJECT_MR_REGISTER_REQ_T  tData;
} EIP_OBJECT_MR_PACKET_REGISTER_REQ_T;

#define EIP_OBJECT_MR_REGISTER_REQ_SIZE   (sizeof(EIP_OBJECT_MR_REGISTER_REQ_T) )
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulDest	uint32_t	0, 0x20	Destination. Set to 0: Destination is operating system 32 (0x20): Destination is the protocol stack
ulLen	uint32_t	12	EIP_OBJECT_MR_REGISTER_REQ_SIZE – Packet data length in bytes
ulSta	uint32_t	0	See *Table 44: EIP_OBJECT_MR_REGISTER_REQ – Packet Status/Error*
ulCmd	uint32_t	0x1A02	EIP_OBJECT_MR_REGISTER_REQ – Command
tData - Structure EIP_OBJECT_MR_REGISTER_REQ_T Data			
ulReserved1	uint32_t	0	Reserved, set to 0
ulClass	uint32_t	1..0xFFFF	Class identifier (predefined class code as described in the CIP specification Vol. 1 chapter 5 (reference [5]) Take care of the address ranges specified above within *Table 87: Address Ranges for the ulClass parameter.*
ulOptionFlags	uint32_t		For type 1, set flag EIP_OBJECT_MR_REGISTER_OPTION_FLAGS_USE_OBJECT_PROVIDED_BY_STACK For type 2, set to 0 Additional CIP object that can be registered with type 1: - Time Sync object (class code 0x43)

Table 88: EIP_OBJECT_MR_REGISTER_REQ – Request Command for register a new class object

Packet Structure Reference

```
typedef  __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_MR_PACKET_REGISTER_CNF_Ttag
{
  HIL_PACKET_HEADER_T           tHead;
} EIP_OBJECT_MR_PACKET_REGISTER_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulLen	uint32_t	0	Packet data length in bytes
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x1A03	EIP_OBJECT_MR_REGISTER_CNF - Command

Table 89: EIP_OBJECT_MR_REGISTER_CNF – Confirmation Command of register a new class object

4.1.5 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 90 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 90: 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. Please refer to Table 92 for descriptions of the Valid Assembly 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.

Packet Structure Reference

```
typedef  __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_AS_REGISTER_REQ_Ttag
{
  uint32_t      ulInstance;
  uint32_t      ulDPMOffset;
  uint32_t      ulSize;
  uint32_t      ulFlags;
} EIP_OBJECT_AS_REGISTER_REQ_T;

typedef  __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_AS_PACKET_REGISTER_REQ_Ttag
{
  HIL_PACKET_HEADER_T           tHead;
  EIP_OBJECT_AS_REGISTER_REQ_T  tData;
} EIP_OBJECT_AS_PACKET_REGISTER_REQ_T;

#define EIP_OBJECT_AS_REGISTER_REQ_SIZE   (sizeof(EIP_OBJECT_AS_REGISTER_REQ_T) )
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulDest	uint32_t	0,0x20	Destination 0: Destination is operating system 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 chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x1A0C	EIP_OBJECT_AS_REGISTER_REQ - Command
Data			
ulInstance	uint32_t	0x0000001... 0xFFFFFFFF (except 0xC0 and 0xC1, see description above)	Assembly instance number See Table 90 (page 111).
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 images 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 range of these instances does not overlap in the DPM.
ulSize	uint32_t	1..504	Size of the data area for the assembly instance data.
ulFlags	uint32_t	Bitmap	Property Flags for the assembly instance, see Table 92 (page 115).

Table 91: EIP_OBJECT_AS_REGISTER_REQ – Request Command for create 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...12	Reserved	Reserved for future use
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 page 60), 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. 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. 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

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

Bits	Name (Bitmask)	Description
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.
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.

Table 92: Assembly Types and Option Flags

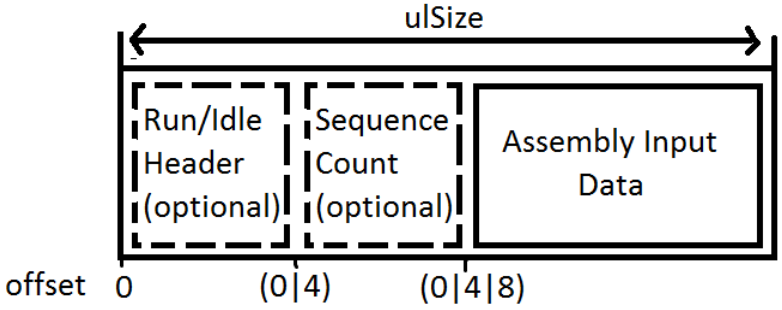


Figure 15: DPM Input area layout according to options EIP_AS_OPTION_MAP_RUNIDLE and EIP_AS_OPTION_MAP_SEQCOUNT and the given Assembly size

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;
```

Packet Structure Reference

```
typedef  __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_AS_REGISTER_CNF_Ttag
{
  uint32_t  ulInstance;      /*!< Assembly instance number from the request packet              */
  uint32_t  ulDPMOffset;     /*!< DPM offset for the instance data area from the request packet */
  uint32_t  ulSize;          /*!< Size of the data area from the request packet                 */
  uint32_t  ulFlags;         /*!< Assembly flags from the request packet                        */
  void*     hDataBuf;        /*!< Handle of the triple data buffer                              */
} EIP_OBJECT_AS_REGISTER_CNF_T;

typedef  __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_AS_PACKET_REGISTER_CNF_Ttag
{
  HIL_PACKET_HEADER_T           tHead;
  EIP_OBJECT_AS_REGISTER_CNF_T  tData;
} EIP_OBJECT_AS_PACKET_REGISTER_CNF_T;

#define EIP_OBJECT_AS_REGISTER_CNF_SIZE   (sizeof(EIP_OBJECT_AS_REGISTER_CNF_T) )
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulLen	uint32_t	20	EIP_OBJECT_AS_REGISTER_CNF_SIZE - Packet data length in bytes
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x1A0D	EIP_OBJECT_AS_REGISTER_CNF - Command
tData - Structure EIP_OBJECT_AS_REGISTER_CNF_TData			
ulInstance	uint32_t		Instance of the Assembly Object (from the request packet)
ulDPMOffset	uint32_t		Offset of the data in the dual port memory (from the request packet)
ulSize	uint32_t	<=504	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		Deprecated

Table 93: EIP_OBJECT_AS_REGISTER_CNF – Confirmation Command of register a new class object

4.1.6 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 116). 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` (section *Acyclic Data Transfer* on page 137).

Packet Structure Reference

```
typedef  __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_REGISTER_SERVICE_REQ_Ttag
{
  uint32_t ulService;
} EIP_OBJECT_REGISTER_SERVICE_REQ_T;

/* command for register a new object to the message router */
typedef  __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_REGISTER_SERVICE_REQ_Ttag
{
  HIL_PACKET_HEADER_T                tHead;
  EIP_OBJECT_REGISTER_SERVICE_REQ_T  tData;
} EIP_OBJECT_PACKET_REGISTER_SERVICE_REQ_T;

#define EIP_OBJECT_REGISTER_SERVICE_REQ_SIZE (sizeof(EIP_OBJECT_REGISTER_SERVICE_REQ_T))
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
`ulDest`	uint32_t	`0, 0x20`	Destination. Set to 0: Destination is operating system 32 (0x20): Destination is the protocol stack
`ulLen`	uint32_t	4	Packet Data Length (In Bytes)
`ulSta`	uint32_t	0	See chapter *Status/Error Codes Overview*
`ulCmd`	uint32_t	`0x00001A44`	`EIP_OBJECT_REGISTER_SERVICE_REQ` - Command / Response
tData - structureData EIP_OBJECT_REGISTER_SERVICE_REQ_T			
`ulService`	uint32_t		Vendor specific service code (see Table 116)

Table 94: `EIP_OBJECT_REGISTER_SERVICE_REQ` - Register Service

Packet Structure Reference

```
typedef  __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_REGISTER_SERVICE_CNF_Ttag
{
  HIL_PACKET_HEADER_T           tHead;
} EIP_OBJECT_PACKET_REGISTER_SERVICE_CNF_T;

#define EIP_OBJECT_REGISTER_SERVICE_CNF_SIZE 0
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulLen	uint32_t	0	Packet Data Length (In Bytes)
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x00001A45	EIP_OBJECT_REGISTER_SERVICE_CNF - Command / Response

Table 95: EIP_OBJECT_REGISTER_SERVICE_CNF – Confirmation Command for Register Service Confirmation

4.1.7 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 96 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` (see 4.2.8). 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` (see 4.2.10). 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.
2-31	**Reserved** Must be set to 0

Table 96: `EIP_OBJECT_SET_PARAMETER_REQ` – Packet Status/Error

Packet Structure Reference

```
#define EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING                       0x00000001
#define EIP_OBJECT_PRM_DISABLE_FLASH_LEDS_SERVICE                       0x00000002

/* Set Parameter Request */
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_SET_PARAMETER_REQ_Ttag
{
  uint32_t ulParameterFlags;
} EIP_OBJECT_SET_PARAMETER_REQ_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_SET_PARAMETER_REQ_Ttag
{
  HIL_PACKET_HEADER_T             tHead;
  EIP_OBJECT_SET_PARAMETER_REQ_T  tData;
}EIP_OBJECT_PACKET_SET_PARAMETER_REQ_T;

#define EIP_OBJECT_SET_PARAMETER_REQ_SIZE  (sizeof(EIP_OBJECT_SET_PARAMETER_REQ_T))
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
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 chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x00001AF2	EIP_OBJECT_SET_PARAMETER_REQ – Command
tData - structure EIP_OBJECT_SET_PARAMETER_REQ_TData			
ulParameterFlags	uint32_t		See *Table 96: EIP_OBJECT_SET_PARAMETER_REQ – Packet Status/Error*

Table 97: EIP_OBJECT_SET_PARAMETER_REQ – Set Parameter Request Request

Packet Structure Reference

```
  typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_SET_PARAMETER_CNF_Ttag
{
  HIL_PACKET_HEADER_T          tHead;
}EIP_OBJECT_PACKET_SET_PARAMETER_CNF_T;

#define EIP_OBJECT_SET_PARAMETER_CNF_SIZE  0
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulLen	uint32_t	0	Packet Data Length (In Bytes)
ulSta	uint32_t		See *Status/Error Codes Overview*
ulCmd	uint32_t	0x00001AF3	EIP_OBJECT_SET_PARAMETER_CNF– Command

Table 98: EIP_OBJECT_SET_PARAMETER_CNF – Set Parameter Confirmation Packet

4.1.8 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 `ulService` 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` and `ulAttribute`.

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 99 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, please refer to section 6 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

Define	Value	Name
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
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 99: 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, please refer to the CIP specification [1], section 3-5.5.

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_CIP_SERVICE_REQ_Ttag
{
  uint32_t    ulService;                          /*!< CIP service code          */
  uint32_t    ulClass;                            /*!< CIP class ID              */
  uint32_t    ulInstance;                         /*!< CIP instance number       */
  uint32_t    ulAttribute;                        /*!< CIP attribute number      */
  uint8_t     abData[EIP_OBJECT_MAX_PACKET_LEN];  /*!< CIP Service Data. <br><br>*/
} EIP_OBJECT_CIP_SERVICE_REQ_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_CIP_SERVICE_REQ_Ttag
{
  HIL_PACKET_HEADER_T            tHead;
  EIP_OBJECT_CIP_SERVICE_REQ_T   tData;
}EIP_OBJECT_PACKET_CIP_SERVICE_REQ_T;

#define EIP_OBJECT_CIP_SERVICE_REQ_SIZE     (sizeof(EIP_OBJECT_CIP_SERVICE_REQ_T) -
EIP_OBJECT_MAX_PACKET_LEN)
```

Packet description

Variable	Type	Value / Range	Description
tHead - Structure HIL_PACKET_HEADER_T			
ulDest	uint32_t	0x20	Destination. Set to 0: Destination is operating system 32 (0x20): Destination is the protocol stack
ulLen	uint32_t	16+n	Packet Data Length in bytes n = Length of service data in bytes (see field abData[])
ulSta	uint32_t	0	See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x1AF8	EIP_OBJECT_CIP_SERVICE_REQ - Command
tData - Structure EIP_OBJECT_CIP_SERVICE_REQ_TData			
ulService	uint32_t	1-31	CIP Service Code
ulClass	uint32_t	Valid Class ID	CIP Class ID (according to *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1")* For available object classes see section *Hilscher EtherNet/IP Stack Capabilities* on page 15.
ulInstance	uint32_t	Valid Instance number	CIP Object Instance number. For available object classes and instances, see section *Hilscher EtherNet/IP Stack Capabilities* on page 15.
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* on page 15
abData[1520]	uint8_t [ ]	0-1520	CIP Service data Number of bytes n provided in this field 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

Table 100: EIP_OBJECT_CIP_SERVICE_REQ – CIP Service Request

Packet Structure Reference

```
#define EIP_OBJECT_MAX_PACKET_LEN   1520           /*!< Maximum packet length */

typedef struct EIP_OBJECT_CIP_SERVICE_CNF_Ttag
{
  uint32_t    ulService;                          /*!< CIP service code          */
  uint32_t    ulClass;                            /*!< CIP class ID              */
  uint32_t    ulInstance;                         /*!< CIP instance number       */
  uint32_t    ulAttribute;                        /*!< CIP attribute number      */

  uint32_t    ulGRC;                              /*!< Generic Error Code        */
  uint32_t    ulERC;                              /*!< Extended Error Code       */

  uint8_t     abData[EIP_OBJECT_MAX_PACKET_LEN];  /*!< CIP service data. <br><br>
} EIP_OBJECT_CIP_SERVICE_CNF_T;


typedef struct EIP_OBJECT_PACKET_CIP_SERVICE_CNF_Ttag
{
  HIL_PACKET_HEADER_T             tHead;
  EIP_OBJECT_CIP_SERVICE_CNF_T    tData;
} EIP_OBJECT_PACKET_CIP_SERVICE_CNF_T;

#define EIP_OBJECT_CIP_SERVICE_CNF_SIZE     (sizeof(EIP_OBJECT_CIP_SERVICE_CNF_T)) -
EIP_OBJECT_MAX_PACKET_LEN
```

Packet Description

Variable	Type	Value / Range	Description
tHead - Structure HIL_PACKET_HEADER_T			
ulDest	uint32_t		Destination
ulLen	uint32_t	24+n	Packet Data Length in bytes n = Length of service data in bytes
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x1AF9	`EIP_OBJECT_CIP_SERVICE_CNF` - Command
tData - Structure EIP_OBJECT_CIP_SERVICE_CNF_TData			
ulService	uint32_t	1-31	CIP Service Code
ulClass	uint32_t	Valid Class ID	CIP Class ID (according to *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 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)
ulGRC	uint32_t		Generic error code according to *"The CIP Networks Library, Volume 1 Common Industrial Protocol Specification Chapter 5, Appendix B-1. Volume 1*) (see also Table 99)
ulERC	uint32_t		Additional error code.
abData[1520]	uint8_t[ ]		CIP Service data Number of bytes provided in this field 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 101: `EIP_OBJECT_CIP_SERVICE_CNF` – Confirmation to CIP Service Request

4.1.9 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. Please refer to reference [2] for further information.

4.1.10 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. Please refer to reference [2] for further information.

4.1.11 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. Please refer to reference [2] for further information.

4.1.12 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. Please refer to reference [2] for a comprehensive description.

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:

- 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
- Apply configuration from database, if applicable
- Apply configuration from Basic Configuration Packet Set (see section 3.3.1), if applicable
- Reply with HIL_CHANNEL_INIT_REQ

4.2 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)	Page
EIP_OBJECT_RESET_IND	0x00001A24	127
EIP_OBJECT_CONNECTION_IND	0x00001A2E	129
EIP_OBJECT_CL3_SERVICE_IND	0x00001A3E	137
EIP_OBJECT_CIP_OBJECT_CHANGE_IND	0x00001AFA	143
HIL_LINK_STATUS_CHANGE_IND	0x00002F8A	147
EIP_OBJECT_LFWD_OPEN_FWD_IND	0x00001A60	150
EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND	0x00001A4C	155
EIP_OBJECT_FWD_CLOSE_FWD_IND	0x00001A4E	157
HIL_STORE_REMANENT_DATA_IND	0x00002F8E	161

Table 102: Overview: Indications of the EtherNet/IP Adapter

4.2.1 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 146: 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.

4.2.2 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 3.2 “Host Application Behavior” and 3.2.4 “Reset”).

The reset service indication is assigned an internal timeout of three seconds as described in section 4.2.1. 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 103 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 (see 4.3.5).

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, since it is not yet specified for EtherNet/IP devices.
3 - 99	Reserved by CIP
100 - 199	Vendor-specific
200 - 255	Reserved by CIP

Table 103: Allowed Values of ulResetTyp

The host application has the possibility to deny the reset request by setting a non-zero status code in the ulSta member of the response packet EIP_OBJECT_RESET_RES. In this case, the protocol stack rejects the reset service request with status CIP_GSR_DEV_IN_WRONG_STATE.

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_RESET_IND_Ttag
{
  uint32_t ulDataIdx;                /*!< Index of the service */
  uint32_t ulResetTyp;               /*!< Type of the reset
}EIP_OBJECT_RESET_IND_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_RESET_IND_Ttag
{
  HIL_PACKET_HEADER_T     tHead;
  EIP_OBJECT_RESET_IND_T  tData;
}EIP_OBJECT_PACKET_RESET_IND_T;

#define EIP_OBJECT_RESET_IND_SIZE          (sizeof(EIP_OBJECT_RESET_IND_T))
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
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 - Command / Response
tData - structure EIP_OBJECT_RESET_IND_TData			
ulDataIdx	uint32_t		Index of the service (the host application does not need to evaluate this parameter)
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 104: EIP_OBJECT_RESET_IND – Reset Request from Bus Indication

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_RESET_RES_Ttag
{
  HIL_PACKET_HEADER_T     tHead;
}EIP_OBJECT_PACKET_RESET_RES_T;

#define EIP_OBJECT_RESET_RES_SIZE          0
```

Packet Description

Structure EIP_OBJECT_PACKET_RESET_RES_T			Type: Response
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 chapter *Status/Error Codes Overview* SUCCESS_HIL_OK– *reset is accepted* ERR_HIL_FAIL– *reset is denied*
ulCmd	uint32_t	0x00001A25	EIP_OBJECT_RESET_RES – Response

Table 105: EIP_OBJECT_RESET_RES – Response to Indication to Reset Request

4.2.3 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. 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 1, otherwise 0.
EIP_CONNECTED	1	Connection has been established

Table 106: Meaning of variable ulConnectionState

Number of Exclusive Owner Connections – usNumExclusiveowner

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

Number of Input Only Connections – usNumInputOnly

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

Number of Listen Only Connections – usNumListenOnly

The number of currently established implicit connections (CIP Class 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

Table 107: 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 108: 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 with which the connection was opened.

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 109: 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 110: 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 111 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 111: 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 112 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 112: Priority

- Connection Type

  Table 113 specifies the connection type encoding within bits 14-13 of the connection parameters:

Bit 30	Bit 29	Connection Type
0	0	Null – connection may be reconfigured
0	1	Multicast
1	0	Point-to-point connection
1	1	Reserved

Table 113: 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 future extendibility of the implementation.

- Redundant Owner

  The redundant owner bit is set, if more than one owner of the connection should be 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 the Forward Open message with which the connection was opened, in number of bytes.

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 the Forward Open message with which the connection was opened, in units of microseconds.

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 the Forward Open message with which the connection was opened, in number of bytes.

This size may be smaller 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 the Forward Open message with which the connection was opened, in units of milliseconds. A value of zero disables the production inhibit timer.

Extended State – `ulExtendedState`

The extended state provides additional information about the cause of the connection state change. This value is meaningful only when `ulConnectionState` equals `EIP_UNCONNECT`. Table 114 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 114: Extended State

Packet Structure Reference

```
#define EIP_OBJECT_CONNECTION_IND_SIZE \
                           sizeof(EIP_OBJECT_CONNECTION_IND_T)

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_CONNECTION_IND_Ttag
{
  uint32_t ulConnectionState;

  uint16_t usNumExclusiveOwner;
  uint16_t usNumInputOnly;
  uint16_t usNumListenOnly;
  uint16_t usNumExplicitMessaging;

  uint8_t   bConnType;
  uint8_t   abReserved[3];

  uint32_t  ulClass;
  uint32_t  ulInstance;
  uint32_t  ulOTConnectionPoints;
  uint32_t  ulTOConnectionPoints;

  uint16_t  usConnSerialNum;
  uint16_t  usVendorId;
  uint32_t  ulOSerialNum;

  uint8_t   bPriority;
  uint8_t   bTimeOutTicks;
  uint8_t   bTimeoutMultiple;
  uint8_t   bTriggerType;

  uint32_t  ulOTConnID;
  uint32_t  ulTOConnID;

  uint32_t  ulOTRpi;
  uint16_t  usOTConnParam;
  uint16_t  usOTConnSize;
  uint32_t  ulTORpi;
  uint16_t  usTOConnParam;
  uint16_t  usTOConnSize;

  uint32_t  ulProInhib;

  uint32_t ulExtendedState;        /*!< Extended status (see \ref EIP_EXT_CONNECTION_STATE_Ttag) */
} EIP_OBJECT_CONNECTION_IND_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_CONNECTION_IND_Ttag
{
  HIL_PACKET_HEADER_T          tHead;
  EIP_OBJECT_CONNECTION_IND_T  tData;
} EIP_OBJECT_PACKET_CONNECTION_IND_T;
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulLen	uint32_t	76	*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 - Command
Data			
ulConnectionState	uint32_t	0, 1	Reason of changing the connection state Connection established (1) Connection disconnected (0)
usNumExclusiveOwner	uint16_t		Number of established exclusive owner connections
usNumInputOnly	uint16_t		Number of established input only connections
usNumListenOnly	uint16_t		Number of established listen only connections
usNumExplicitMessaging	uint16_t		Number of established explicit connections
bConnType	uint8_t	1-5	Connection Type: 1 - Exclusive Owner 3 – Listen Only 4 – Input Only 5 – Explicit Messaging
abReserved	uint8_t[3]	0	Reserved. Always set to 0.
ulClass	uint32_t		Class to which the connection was directed
ulInstance	uint32_t		Corresponding class instance
ulOTConnectionPoints	uint32_t		Output connection point
ulTOConnectionPoints	uint32_t		Input connection point
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		Producer inhibit time
ulExtendedState	uint32_t		0: No extended status 1: Connection timeout

Table 115: EIP_OBJECT_CONNECTION_IND – Indication of Connection

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_CONNECTION_RES_Ttag
{
  HIL_PACKET_HEADER_T     tHead;
} EIP_OBJECT_PACKET_CONNECTION_RES_T;
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulDest	uint32_t		Destination. Use value from Indication
ulLen	uint32_t	0	Packet Data Length (In Bytes)
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x00001A2F	Command / Response

Table 5: EIP_OBJECT_CONNECTION_RES – Response to indication of Connection

4.2.4 Configuration Assemblies

Configuration assemblies hold a fixed or variable size configuration data bytestream 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.

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.

4.2.5 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/CNF` (see section *Register an additional Object Class* on page 108 of this document) 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/CNF` (see section *Register Service* on page 117 of this document) 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:

- If the service was issued over a class 3 connection (connected explicit), the O→T connection ID of that connection
- The CIP Service Code `ulService`
- The CIP Object Class ID `ulObject`, Instance ID `ulInstance` and Attribute ID `ulAttribute` addressed by the service
- 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 and Attribute ID correspond to the normal CIP Addressing. These fields are used for the most common services that use the addressing format “Service → Class → Instance → Attribute”. 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 116: Specified Ranges of numeric Values of Service Codes (Variable ulService)* gives an overview. This table is taken from the CIP specification (“*Volume 1 Common Industrial Protocol Specification Chapter 4,* Table 4-9.6”, s*ee reference [5]*).

Range of numeric value of service code (variable `ulService`)	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 #3).
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 Chapter 2 of reference [8])
0x0100-0xFFFF	Hilscher-specific services to manage objects from application side.

Table 116: Specified Ranges of numeric Values of Service Codes (Variable ulService)

Note: It is specific to each object which services it implements. For Object Class IDs in the Open range, please refer to the CIP specification about the requirements to fulfill.
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 117 lists the service codes for the Common Services. This table is taken from the CIP specification (“Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1”, see reference [5]).

Service code (numeric value of `ulService`)	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)

Service code (numeric value of `ulService`)	Service to be executed
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 117: 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. Please refer to Table 99 for the stack's definitions of generic status codes.

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 16 displays a sequence diagram for the EIP_OBJECT_CL3_SERVICE_IND/RES packet (see 3.3 “Configuration using the Packet API”).

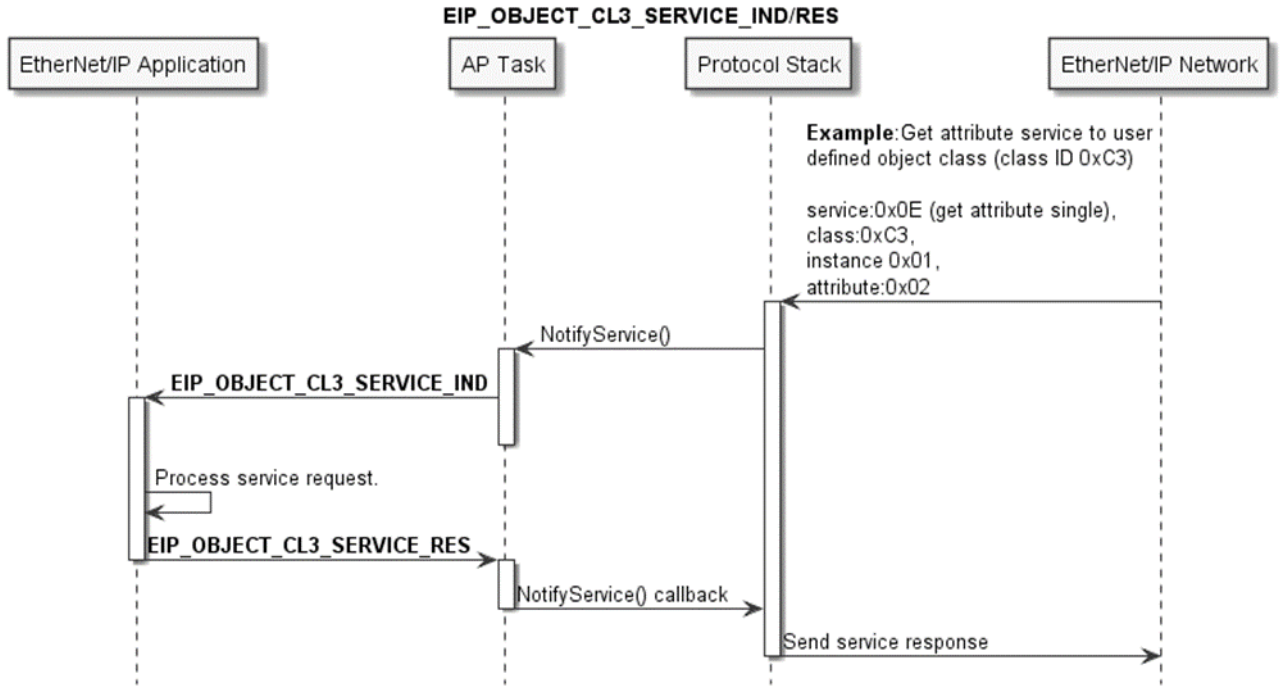


Figure 16: Sequence Diagram for the EIP_OBJECT_CL3_SERVICE_IND/RES Packet for the Extended Packet Set

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_CL3_SERVICE_IND_Ttag
{
  uint32_t    ulConnectionId;          /*!< Connection Handle \n
  uint32_t    ulService;               /*!< Service \n
  uint32_t    ulObject;                /*!< Object class \n
  uint32_t    ulInstance;              /*!< Instance \n
  uint32_t    ulAttribute;             /*!< Attribute \n
  uint8_t     abData[1];
} EIP_OBJECT_CL3_SERVICE_IND_T ;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_CL3_SERVICE_IND_Ttag
{
  HIL_PACKET_HEADER_T             tHead;
  EIP_OBJECT_CL3_SERVICE_IND_T    tData;
} EIP_OBJECT_PACKET_CL3_SERVICE_IND_T;

#define EIP_OBJECT_CL3_SERVICE_IND_SIZE (sizeof(EIP_OBJECT_CL3_SERVICE_IND_T)-1)
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulLen	uint32_t	20 + n	Packet Data Length (In Bytes) n = Length of Service Data Area
ulSta	uint32_t	0	Status not in use for indication.
ulCmd	uint32_t	0x1A3E	EIP_OBJECT_CL3_SERVICE_IND - Command / Response
tData - Structure EIP_OBJECT_CL3_SERVICE_IND_TData			
ulConnectionId	uint32_t	0 ... 2^{32}-1	Unused. Always set to zero.
ulService	uint32_t	1-0xFFFF	CIP Service Code
ulObject	uint32_t	1-0xFFFF	CIP Class ID
ulInstance	uint32_t	1-0xFFFF	CIP Instance Number
ulAttribute	uint32_t	0-0xFFFF	CIP Attribute Number The attribute number is 0, if the service does not address a specific attribute but the whole instance.
abData[]	uint8_t [n]		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 118: EIP_OBJECT_CL3_SERVICE_IND - Indication of acyclic Data Transfer

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST IP_OBJECT_CL3_SERVICE_RES_Ttag
{
  uint32_t    ulConnectionId;          /*!< Connection Handle \n */
  uint32_t    ulService;               /*!< Service \n */
  uint32_t    ulObject;                /*!< Object class \n */
  uint32_t    ulInstance;              /*!< Instance \n */
  uint32_t    ulAttribute;             /*!< Attribute \n */

  uint32_t    ulGRC;                 /*!< Generic Error Code \n*/
  uint32_t    ulERC;                 /*!< Extended Error Code \n */

  uint8_t     abData[1];             /*!< Response service  data */
} EIP_OBJECT_CL3_SERVICE_RES_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_CL3_SERVICE_RES_Ttag
{
  HIL_PACKET_HEADER_T            tHead;
  EIP_OBJECT_CL3_SERVICE_RES_T   tData;
} EIP_OBJECT_PACKET_CL3_SERVICE_RES_T;

#define EIP_OBJECT_CL3_SERVICE_RES_SIZE (sizeof(EIP_OBJECT_CL3_SERVICE_RES_T)-1)
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulDest	uint32_t		Destination. Use value from Indication
ulLen	uint32_t	28 + n	Packet Data Length (In Bytes) where n = Length of Service Data Area
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x00001A3F	EIP_OBJECT_CL3_SERVICE_RES - Command / Response
tData - Structure EIP_OBJECT_CL3_SERVICE_RES_TData			
ulConnectionId	uint32_t	0 ... 2^{32}-1	Unique Id from the indication packet
ulService	uint32_t	1-0xFFFF	CIP Service Code from the indication packet
ulObject	uint32_t	1-0xFFFF	CIP Object from the indication packet
ulInstance	uint32_t	1-0xFFFF	CIP Instance from the indication packet
ulAttribute	uint32_t	0-0xFFFF	CIP Attribute 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 119: EIP_OBJECT_CL3_SERVICE_RES – Response to Indication of acyclic Data Transfer

4.2.6 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 default CIP objects implemented by the EtherNet/IP Stack. 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 sections 2.4.1.1 "Attribute Option Flags").

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. Please refer to section 4.2.1 for a description of this mechanism.

As an example, Figure 17 displays a sequence diagram for the EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES packet in case the host application stores remanent data.

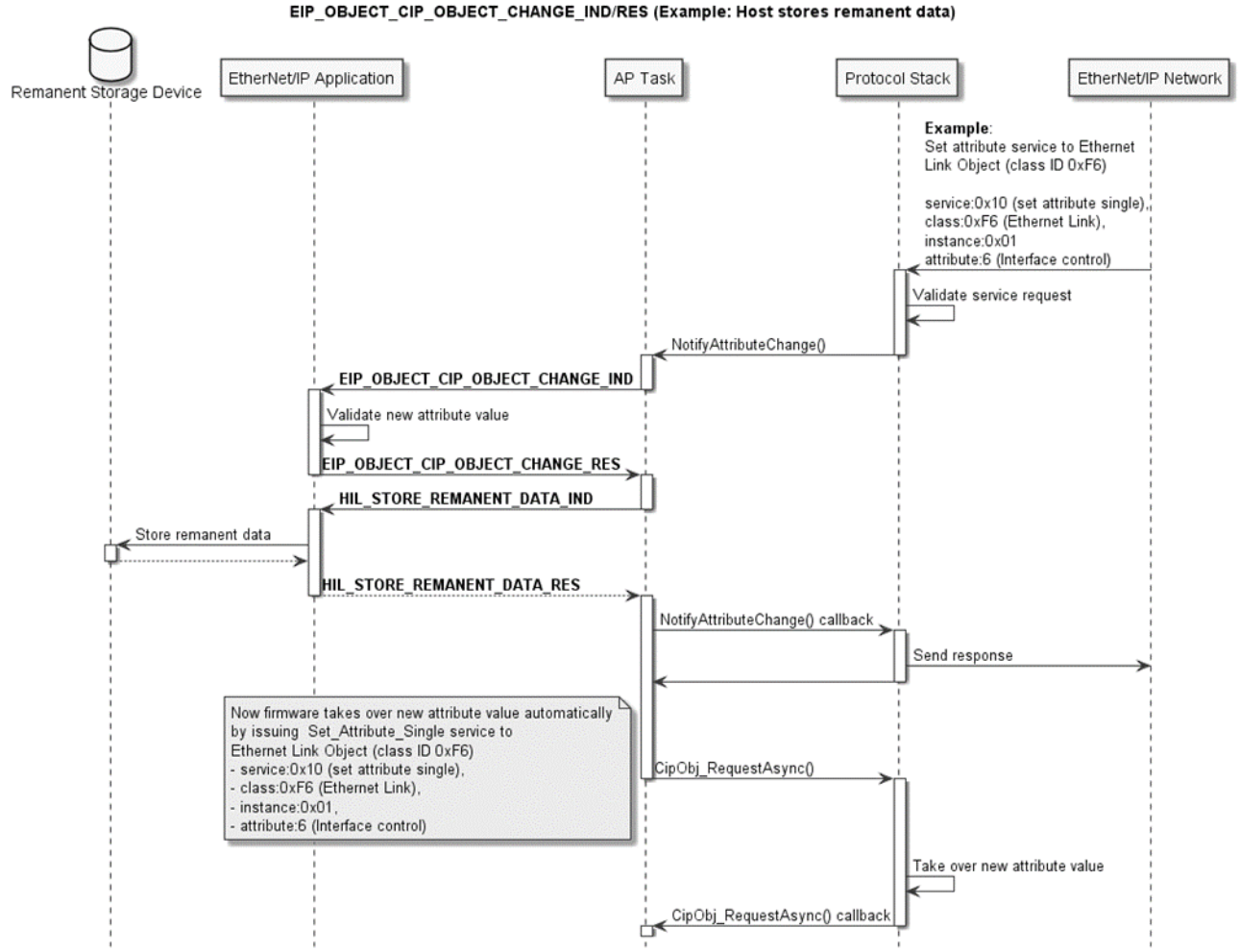


Figure 17: Exemplary sequence diagram for the EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES packet sequence

Packet Structure Reference

```
#define EIP_OBJECT_CIP_OBJECT_CHANGE_IND_PROPOSE  (0x10)    /*!< The attribute change is pending
                                                                 and the host is given the chance
to decide */
#define EIP_OBJECT_CIP_OBJECT_CHANGE_IND_INFORM    (0x20)    /*!< The attribute change already took
place
                                                                 and the indication is simply
informed about it */

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_CIP_OBJECT_CHANGE_IND_Ttag
{
  uint32_t    ulInfoFlags;                        /*!< Information flags     */
  uint32_t    ulService;                          /*!< CIP service code      */
  uint32_t    ulClass;                            /*!< CIP class ID          */
  uint32_t    ulInstance;                         /*!< CIP instance number   */
  uint32_t    ulAttribute;                        /*!< CIP attribute number  */
  uint8_t     abData[EIP_OBJECT_MAX_PACKET_LEN];  /*!< Service Data          */
} EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_CIP_OBJECT_CHANGE_IND_Ttag
{
  HIL_PACKET_HEADER_T                 tHead;
  EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T   tData;
} EIP_OBJECT_PACKET_CIP_OBJECT_CHANGE_IND_T;

#define EIP_OBJECT_CIP_OBJECT_CHANGE_IND_SIZE     (sizeof(EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T) -
EIP_OBJECT_MAX_PACKET_LEN)
```

Packet Description

Variable	Type	Value / Range	Description
ulLen	uint32_t	20+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 - Comm
structure `EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T`			
ulInfoFlags	uint32_t	Either EIP_OBJECT_CIP_OBJECT_CHANGE_IND_PROPOSE or EIP_OBJECT_CIP_OBJECT_CHANGE_IND_INFORM	Flags specifying the type of change indication. For ob changes of the type EIP_OBJECT_CIP_OBJECT_CHANGE_IND_PROPO , the host application can decide whether or not to tak over the new attribute value by means of the status co returned in the response packet. The object change w only be applied in case the host application replies wit success status. For object changes of the type EIP_OBJECT_CIP_OBJECT_CHANGE_IND_INFOR no such means are provided.
ulService	uint32_t	0x10	CIP service code Currently only the *SetAttributeSingle* service is used i this indication.
ulClass	uint32_t		CIP class ID
ulInstance	uint32_t		CIP instance number
ulAttribute	uint32_t		CIP attribute number
abData[]	uint8_t[]		Attribute Data Number of bytes n provided in abData = `tHead.ulLen - EIP_OBJECT_CIP_OBJECT_CHANGE_IND_SIZE`

Table 120: `EIP_OBJECT_CIP_OBJECT_CHANGE_IND` – CIP Object Change Indication

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_CIP_OBJECT_CHANGE_RES_Ttag
{
  HIL_PACKET_HEADER_T                  tHead;
  EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T   tData;
} EIP_OBJECT_PACKET_CIP_OBJECT_CHANGE_RES_T;

#define EIP_OBJECT_CIP_OBJECT_CHANGE_RES_SIZE    (sizeof(EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T) -
EIP_OBJECT_MAX_PACKET_LEN)
```

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		See chapter *Status/Error Codes Overview* All status code other than SUCCESS_HIL_OK (0) will be treated equally (error reply on the network and value not taken over). Note that this is only effective for changes of the type EIP_OBJECT_CIP_OBJECT_CHANGE_IND_PROPOSE.
ulCmd	uint32_t	0x1AFB	EIP_OBJECT_CIP_OBJECT_CHANGE_RES - Command

Table 121: `EIP_OBJECT_CIP_OBJECT_CHANGE_RES` – Response to CIP Object Change Indication

4.2.7 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 – 0x2F10).

Packet Structure Reference

```
/* LINK STATUS CHANGE INDICATION */
#define HIL_LINK_STATUS_CHANGE_IND              0x00002F8A

typedef struct HIL_LINK_STATUS_Ttag
{
  uint32_t        ulPort;           /*!< Port the link status is for */
  uint32_t        fIsFullDuplex;    /*!< If a full duplex link is available on this port */
  uint32_t        fIsLinkUp;        /*!< If a link is available on this port */
  uint32_t        ulSpeed;          /*!< Speed of the link \n\n
                                     \valueRange
                                     0:   No link \n
                                     10:  10MBit  \n
                                     100: 100MBit \n */
} HIL_LINK_STATUS_T;

typedef struct HIL_LINK_STATUS_CHANGE_IND_DATA_Ttag
{
  HIL_LINK_STATUS_T  atLinkData[2];
} HIL_LINK_STATUS_CHANGE_IND_DATA_T;

typedef struct HIL_LINK_STATUS_CHANGE_IND_Ttag
{
  HIL_PACKET_HEADER                tHead;
  HIL_LINK_STATUS_CHANGE_IND_DATA_T tData;
} HIL_LINK_STATUS_CHANGE_IND_T;
```

Packet Description

Variable	Type	Value / Range	Description
structure HIL_PACKET_HEADER_T			
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-command
structure HIL_LINK_STATUS_CHANGE_IND_DATA_TData			
atLinkData[2]	HIL_LINK_STATUS_T		Link status information for two ports. If only one port is available, ignore second entry.

Table 122: 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 123: Structure HIL_LINK_STATUS_CHANGE_IND_DATA_T

Packet Structure Reference

```
typedef struct HIL_LINK_STATUS_CHANGE_RES_Ttag
{
  HIL_PACKET_HEADER_T             tHead;
} HIL_LINK_STATUS_CHANGE_RES_T;

#define HIL_LINK_STATUS_CHANGE_RES_SIZE   (0)
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
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		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x2F8B	HIL_LINK_STATUS_CHANGE_RES – Command

Table 124: HIL_LINK_STATUS_CHANGE_RES_T - Link Status Change Response

4.2.8 Forward_Open Indication

Note: The Forward Open Forwarding functionality must be enabled by setting the Parameter flag EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING using command EIP_OBJECT_SET_PARAMETER_REQ (0x00001AF2).

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 (described in section 4.2.8)
- EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND (described in section 4.2.9)
- EIP_OBJECT_FWD_CLOSE_FWD_IND (described in section 4.2.10)

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.

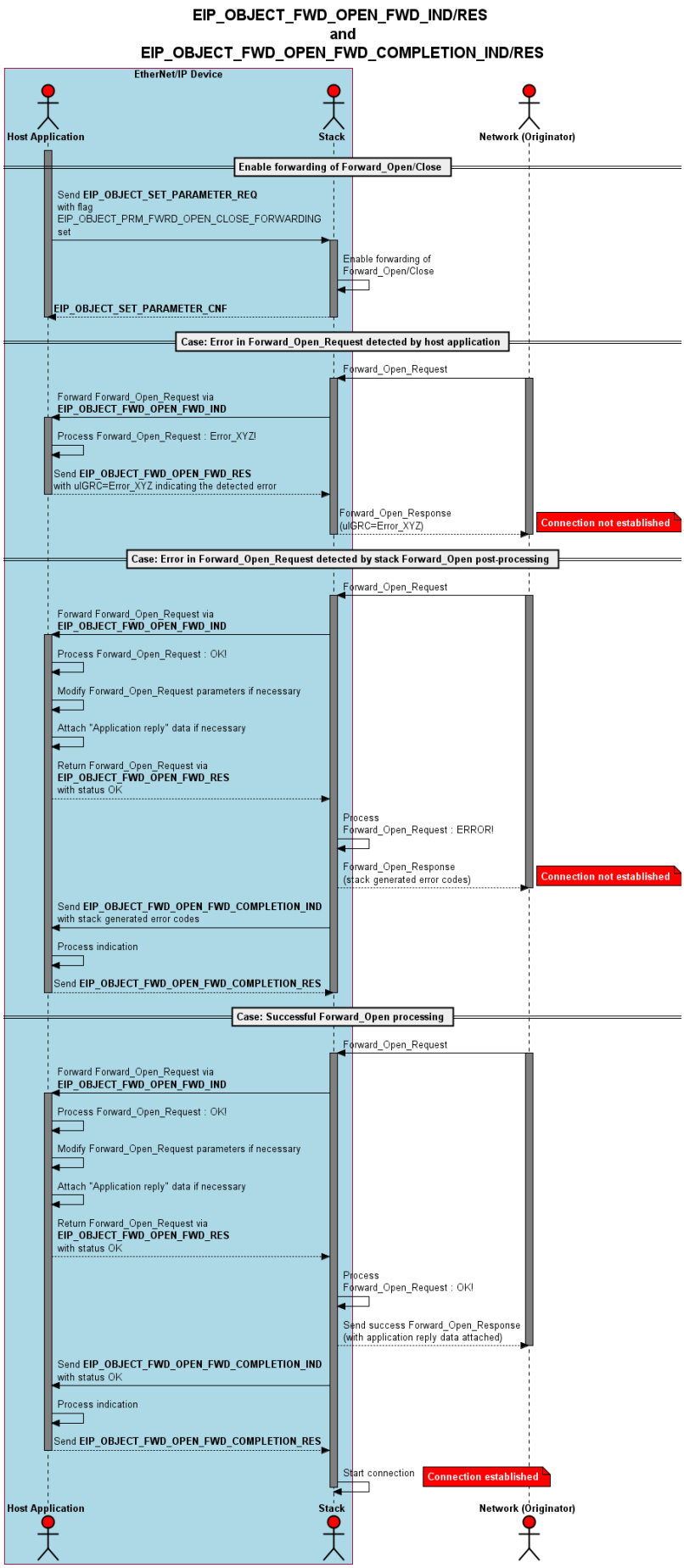
Furthermore, the application can attach “Application Reply Data” to the EIP_OBJECT_LFWD_OPEN_FWD_RES response message, which will be sent back to the originator on success. By setting a nonzero CIP status code in the response packet, the host application can effectively reject the opening or closing 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 *Figure 18*.

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.



Packet Structure Reference

```
/*####################################################################################*/

/**
   Packets defined for forwarding of forward close and forward open
   from Stack to host application.
   Packets are defined on the assumption/rule that the host application uses the received
   packet for generating the response.
   That's why the indication reserves some space that are used for response parameters
*/

#define EIP_DEFAULT_PATH_LEN             1000

/* Large Forward Open Service - Request Data */
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_CM_LARGEFWOPEN_REQ_Ttag
{
  uint8_t   bPriority;          /* used to calculate request timeout information    */
  uint8_t   bTimeOutTicks;      /* used to calculate request timeout information    */
  uint32_t  ulOTConnID;         /* Network connection ID originator to target       */
  uint32_t  ulTOConnID;         /* Network connection ID target to originator       */
  uint16_t  usConnSerialNum;    /* Connection serial number                         */
  uint16_t  usVendorId;         /* Originator Vendor ID                             */
  uint32_t  ulOSerialNum;       /* Originator serial number                         */
  uint8_t   bTimeoutMultiple;   /* Connection timeout multiple                      */
  uint8_t   abReserved1[3];     /* reserved                                         */
  uint32_t  ulOTRpi;            /* Originator to target requested packet rate in us */
  uint32_t  ulOTConnParam;      /* Originator to target connection parameter        */
  uint32_t  ulTORpi;            /* target to originator requested packet rate in us */
  uint32_t  ulTOConnParam;      /* target to originator connection parameter        */
  uint8_t   bTriggerType;       /* Transport type/trigger                           */
  uint8_t   bConnPathSize;      /* Connection path size                             */
  uint8_t   bConnPath[EIP_DEFAULT_PATH_LEN];        /* connection path              */
} EIP_CM_LARGEFWOPEN_REQ_T;

/* Deliver Forward Open to host application */
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_LFWD_OPEN_FWD_IND_Ttag
{
  void*                    pRouteMsg;
  /**< Link to to remember underlying Encapsulation request (must not be modified by app)   */
  uint32_t                 aulReserved[4];
  /**< Place holder to be filled by response parameters, see  EIP_OBJECT_LFWD_OPEN_FWD_RES_T */
  EIP_CM_LARGEFWOPEN_REQ_T tFwdOpenData;
  /**< Forward Open request data to be delivered to host                                  */
} EIP_OBJECT_LFWD_OPEN_FWD_IND_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_LFWD_OPEN_FWD_IND_Ttag
{
  HIL_PACKET_HEADER_T                 tHead;
  EIP_OBJECT_LFWD_OPEN_FWD_IND_T      tData;
} EIP_OBJECT_PACKET_LFWD_OPEN_FWD_IND_T;

#define EIP_OBJECT_LFWD_OPEN_FWD_IND_SIZE  (sizeof(EIP_OBJECT_LFWD_OPEN_FWD_IND_T) -
EIP_OBJECT_MAX_PACKET_LEN)
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
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 - Command
tData - Structure EIP_OBJECT_LFWD_OPEN_FWD_IND_T			
pRouteMsg	void*		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_CM_LARGEFWOPEN_REQ_T		Forward Open data (See *Table 126*)

Table 125: EIP_OBJECT_LFWD_OPEN_FWD_IND – Forward_Open indication

The following *Table 126* explains the structure EIP_CM_APP_LFWOPEN_IND_T:

Structure EIP_CM_APP_LFWOPEN_IND_T		
		Description
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
usOTConnParam	uint16_t	Originator to target connection parameter
ulTORpi	uint32_t	Target to originator requested packet rate in microseconds
usTOConnParam	uint16_t	Target to originator connection parameter
bTriggerType	uint8_t	Transport type/trigger
bConnPathSize	uint8_t	Connection path size in 16 bit words
abConnPath[EIP_DEFAULT_PATH_LEN]	uint8_t	Connection path

Table 126: EIP_CM_APP_LFWOPEN_IND_T - Forward_Open request data

For a detailed description on these parameters, see section *Connection State Change Indication*.

Packet Structure Reference

```
/**
   Deliver Forward Open to host application - Response
   Contains the potentially modified forward open data (since host application may need to modify
connection points e.g. for safety)
   Additional parameters are: Status (host application result) and a reference (size/offset) to the
application reply data
   that needs to be appended to the Forward open response generated by the stack
*/

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_LFWD_OPEN_FWD_RES_Ttag
{
  void*                    pRouteMsg;          /**< Link to underlying Encapsulation request
*/
  uint32_t                 ulGRC;              /**< General Error Code
*/
  uint32_t                 ulERC;              /**< Extended Error Code
*/
  uint32_t                 ulAppReplyOffset;   /**< Offset of Application Reply data
*/
  uint32_t                 ulAppReplySize;     /**< Byte-size of Application Reply data
*/
  EIP_CM_LARGEFWOPEN_REQ_T tFwdOpenData;       /**< modified forward open (Note that the application
                                               reply data is appended, which is not visible here) */
} EIP_OBJECT_LFWD_OPEN_FWD_RES_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_LFWD_OPEN_FWD_RES_Ttag
{
  HIL_PACKET_HEADER_T                       tHead;
  EIP_OBJECT_LFWD_OPEN_FWD_RES_T            tData;
} EIP_OBJECT_PACKET_LFWD_OPEN_FWD_RES_T;

#define EIP_OBJECT_LFWD_OPEN_FWD_RES_SIZE (sizeof(EIP_OBJECT_LFWD_OPEN_FWD_RES_T) -
EIP_DEFAULT_PATH_LEN)
```

Packet Description

Variable	Type	Value / Range	Description
tHead - Structure HIL_PACKET_HEADER_T			
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		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x1A61	EIP_OBJECT_LFWD_OPEN_FWD_RES - Command
tData - Structure EIP_OBJECT_LFWD_OPEN_FWD_IND_T			
pRouteMsg	void*		Pointer to underlying Encapsulation request
ulGRC	uint32_t		General Error Code, see Table 99
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_CM_LARGEFWOPEN_REQ_T		Forward Open data (See *Table 126*)

Table 127: EIP_OBJECT_LFWD_OPEN_FWD_RES – Response of Forward_Open indication

4.2.9 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 (0x00001AF2).

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.

Please refer to Figure 18 on page 151 to get an overview about the possible packet sequences.

Packet Structure Reference

```
/** Status indication of Forward Open, that was previously processed by the
 * host (see EIP_OBJECT_LFWD_OPEN_FWD_IND)*/
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_Ttag
{
  uint16_t  usCmInstance;       /**< Connection manager instance\n */
  uint16_t  usConnSerialNum;    /**< Connection serial number           */
  uint16_t  usVendorId;         /**< Originator Vendor ID               */
  uint32_t  ulOSerialNum;       /**< Originator serial number           */
  uint32_t  ulGRC;              /**< General Error Code                 */
  uint32_t  ulERC;              /**< Extended Error Code                */
} EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_FWD_OPEN_FWD_COMPLETION_IND_Ttag
{
  HIL_PACKET_HEADER_T                       tHead;
  EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_T  tData;
} EIP_OBJECT_PACKET_FWD_OPEN_FWD_COMPLETION_IND_T;

#define EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_SIZE
(sizeof(EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_T))
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
ulLen	uint32_t	EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_SIZE	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 - Command
tData - Structure EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_T			
usCmInstance	uint16_t	0 - 64	Connection Manager Instance. Value 0 is not a valid instance number. It will be present if the connection was not established (ulGRC != 0).
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 99
ulERC	uint32_t		Extended Error Code

Table 128: EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND – Forward_Open completion indication

For more information on the parameters `usConnSerialNum`, `usVendorId` and `ulOSerialNum`, see section *Connection State Change Indication* on page 129.

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_FWD_OPEN_FWD_COMPLETION_RES_Ttag
{
  HIL_PACKET_HEADER_T                       tHead;
} EIP_OBJECT_PACKET_FWD_OPEN_FWD_COMPLETION_RES_T;

#define EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES_SIZE   0
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
`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		See chapter *Status/Error Codes Overview*
`ulCmd`	uint32_t	`0x1A4D`	`EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES` - Command

Table 129: `EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES` – Response of Forward_Open completion indication

4.2.10 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 (0x00001AF2).

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. Please refer to Figure 18 on page 151 to get an overview about the possible packet sequences.

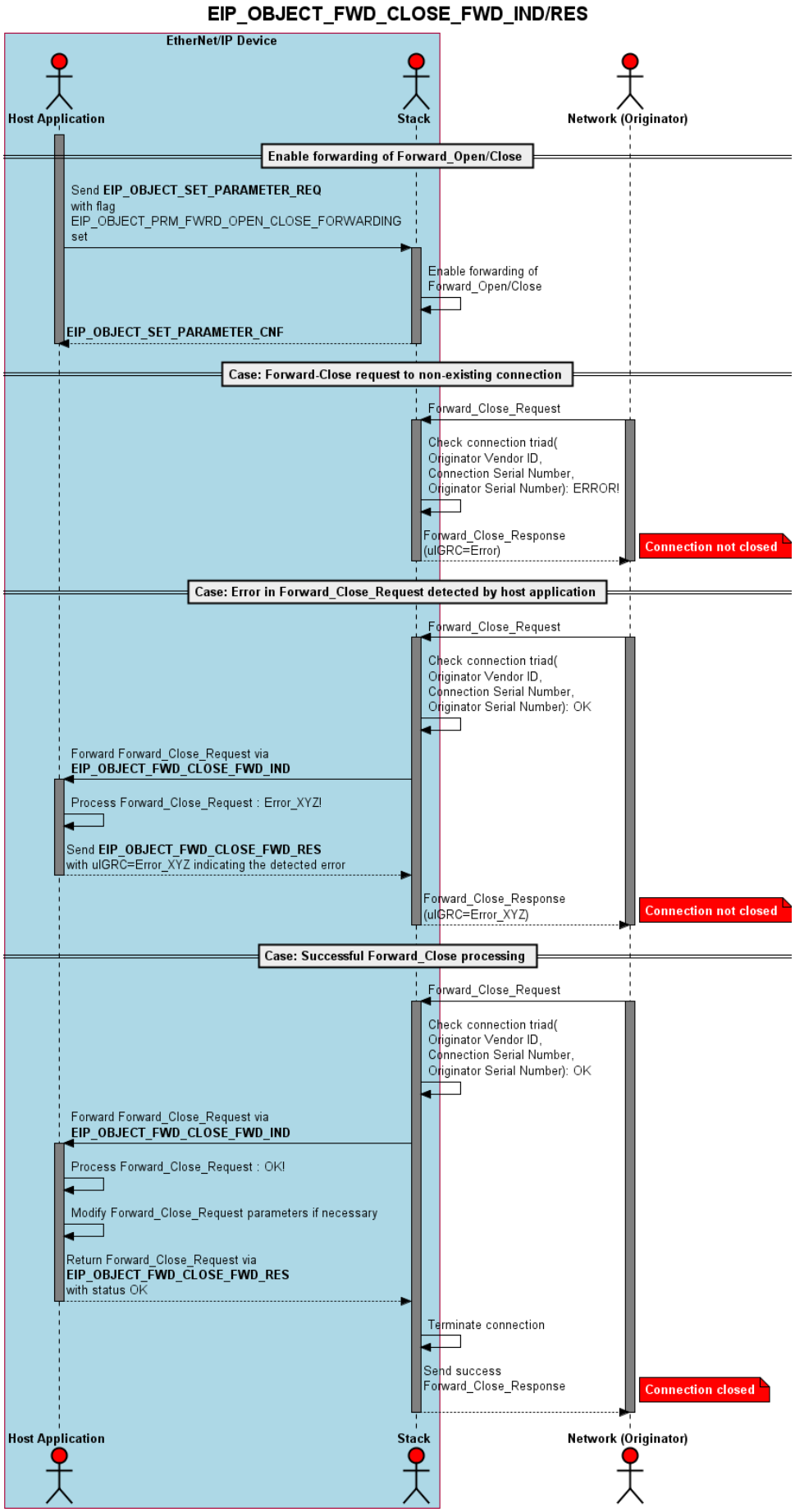


Figure 19: Packet sequence for Forward_Close forwarding functionality

Packet Structure Reference

```
#define EIP_OBJECT_MAX_PACKET_LEN   1520

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_CM_APP_FWCLOSE_IND_Ttag
{
  uint8_t   bPriority;                          /* Used to calculate request timeout information*/
  uint8_t   bTimeOutTicks;                      /* Used to calculate request timeout information*/
  uint16_t  usConnSerialNum;                    /* Connection serial number                     */
  uint16_t  usVendorId;                         /* Originator Vendor ID                         */
  uint32_t  ulOSerialNum;                       /* Originator serial number                     */
  uint8_t   bConnPathSize;                      /* Connection path size in 16bit words          */
  uint8_t   bReserved1;
  uint8_t   bConnPath[EIP_OBJECT_MAX_PACKET_LEN]; /* connection path                            */
} EIP_CM_APP_FWCLOSE_IND_T;

/* Deliver Forward Close to host application - indication */
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_FWD_CLOSE_FWD_IND_Ttag
{
  void*                    pRouteMsg;             /*!< Link to underlying Encapsulation request */
  uint32_t                aulReserved[2];         /*!< Placeholder to be filled by response
parameters,
                                                       see  EIP_OBJECT_FWD_CLOSE_FWD_RES_T      */
  EIP_CM_APP_FWCLOSE_IND_T  tFwdCloseData;        /*!< Forward close request data to be delivered
to host */
} EIP_OBJECT_FWD_CLOSE_FWD_IND_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_FWD_CLOSE_FWD_IND_Ttag
{
  HIL_PACKET_HEADER_T             tHead;
  EIP_OBJECT_FWD_CLOSE_FWD_IND_T  tData;
} EIP_OBJECT_PACKET_FWD_CLOSE_FWD_IND_T;

#define EIP_OBJECT_FWD_CLOSE_FWD_IND_SIZE   (sizeof(EIP_OBJECT_FWD_CLOSE_FWD_IND_T) -
EIP_OBJECT_MAX_PACKET_LEN)
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
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 - Command
tData - Structure EIP_OBJECT_FWD_CLOSE_FWD_IND_T			
pRouteMsg	void		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 131: EIP_CM_APP_FWCLOSE_IND_T - Forward_Close request data* *)*

Table 130: EIP_OBJECT_FWD_CLOSE_FWD_IND – 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
bConnPath[EIP_OBJECT_MAX_PACKET_LEN]	uint8_t	Connection path

Table 131: EIP_CM_APP_FWCLOSE_IND_T - Forward_Close request data

Packet Structure Reference

```
/*
  Deliver Forward Close to host application - response
  Contains the modified forward close (since application may need to modify connection points e.g.
for safety)
  Additional parameters are: Status (host application result)
*/
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_FWD_CLOSE_FWD_RES_Ttag
{
  void*                  pRouteMsg;            /*!< Link to underlying Encapsulation request */
  uint32_t               ulGRC;                /*!< General Error Code                       */
  uint32_t               ulERC;                /*!< Extended Error Code                      */
  EIP_CM_APP_FWCLOSE_IND_T  tFwdCloseData;     /*!< Modified forward close                   */
} EIP_OBJECT_FWD_CLOSE_FWD_RES_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_OBJECT_PACKET_FWD_CLOSE_FWD_RES_Ttag
{
  HIL_PACKET_HEADER_T              tHead;
  EIP_OBJECT_FWD_CLOSE_FWD_RES_T   tData;
} EIP_OBJECT_PACKET_FWD_CLOSE_FWD_RES_T;

#define EIP_OBJECT_FWD_CLOSE_FWD_RES_SIZE   (sizeof(EIP_OBJECT_FWD_CLOSE_FWD_RES_T) -
EIP_OBJECT_MAX_PACKET_LEN)
```

Packet Description

Variable	Type	Value / Range	Description
tHead – Structure HIL_PACKET_HEADER_T			
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		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x1A4F	EIP_OBJECT_FWD_CLOSE_FWD_RES - Command
tData - Structure EIP_OBJECT_FWD_CLOSE_FWD_RES_T			
pRouteMsg	void*		Pointer to underlying Encapsulation request
ulGRC	uint32_t		General Error Code, see Table 99
ulERC	uint32_t		Extended Error Code
tFwdCloseData	EIP_CM_APP_FWCLOSE_IND_T		*Forward Close data (See Table 131: EIP_CM_APP_FWCLOSE_IND_T - Forward_Close request data)*

Table 132: EIP_OBJECT_FWD_CLOSE_FWD_RES – Response of Forward_Close indication

4.2.11 Store Remanent Data Indication

In case the host application is responsible to store remanent data (section *Remanent data* on page 88), then it must handle his service.

The netX firmware sends this service to the host application every time it wants the remanent data to be stored. The remanent data indication always contains the complete remanent data block, not only the data that has changed. The host application shall compare the newly received remanent data block with the last stored remanent data block in order to avoid writing the same data repeatedly. It is up to the host application to consider the wear of the storage device.

Note: The response packet shall be sent synchronously to remanent data storage, i.e. after all data has been written to the storage device.

Store Remanent Data Indication

Structure HIL_STORE_REMANENT_DATA_IND_T			**Type: Indication**
Variable	Type	Value / Range	Description
`ulLen`	`uint32_t`	4 + n	Packet Data Length in bytes. n = number of bytes of remanent data
`ulSta`	`uint32_t`	0	Status not in use for indication.
`ulCmd`	`uint32_t`	0x00002F8E	HIL_STORE_REMANENT_DATA_IND - Command
Data			
`ulComponentID`	`uint32_t`	0x00590000	Unique identifier of the EtherNet/IP Stack
`abData[]`	`uint8_t[]`		Remanent data that shall be stored by the host application

Table 133: HIL_STORE_REMANENT_DATA_IND_T – Store Remanent Data Indication

Packet Structure Reference

```
/* SET REMANENT DATA REQUEST */
```

Value for ulComponentID

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

typedef struct HIL_STORE_REMANENT_DATA_IND_DATA_Ttag
{
   uint32_t  ulComponentId;
   uint8_t   abData[];
} HIL_STORE_REMANENT_DATA_IND_DATA_T;

typedef struct HIL_STORE_REMANENT_DATA_IND_Ttag
{
  HIL_PACKET_HEADER_T                tHead;
  HIL_STORE_REMANENT_DATA_IND_DATA_T  tData;
} HIL_STORE_REMANENT_DATA_IND_T;
```

The firmware returns the following packet:

Structure Information: *HIL_STORE_REMANENT_DATA_RES_T*			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
ulDest	uint32_t		Destination. Use value from Indication
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulSta	uint32_t		See section 6.1
ulCmd	uint32_t	0x00002F8D	HIL_STORE_REMANENT_DATA_RES
Data			
ulComponentID	uint32_t	0x00590000	Unique identifier of the EtherNet/IP Stack

Table 134: HIL_STORE_REMANENT_DATA_RES_T – Store Remanent Data

Packet Structure Reference

```
/* STORE REMANENT DATA RESPONSE */
#define HIL_STORE_REMANENT_DATA_RES              HIL_STORE_REMANENT_DATA_IND + 1

typedef struct HIL_STORE_REMANENT_DATA_RES_DATA_Ttag
{
  uint32_t  ulComponentId;

} HIL_STORE_REMANENT_DATA_RES_DATA_T;

typedef struct HIL_STORE_REMANENT_DATA_RES_Ttag
{
  HIL_PACKET_HEADER_T                tHead;
  HIL_STORE_REMANENT_DATA_RES_DATA_T tData;
} HIL_STORE_REMANENT_DATA_RES_T;
```

4.3 Additional services requested by the application

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

Packet	Command code (REQ/CNF or IND/RES)	Page
EIP_APS_GET_MS_NS_REQ	0x0000360E	164
HIL_SET_WATCHDOG_TIME_REQ	0x00002F04	165
HIL_GET_WATCHDOG_TIME_REQ	0x00002F02	165
HIL_GET_DPM_IO_INFO_REQ	0x00002F0C	165
HIL_UNREGISTER_APP_REQ	0x00002F12	125
HIL_DELETE_CONFIG_REQ	0x00002F14	166
HIL_LOCK_UNLOCK_CONFIG_REQ	0x00002F32	166
HIL_FIRMWARE_IDENTIFY_REQ (see reference [2])	0x00001EB6	-
HIL_SET_REMANENT_DATA_REQ	0x00002F8C	166
HIL_SET_TRIGGER_TYPE_REQ	0x00002F90	167
HIL_GET_TRIGGER_TYPE_REQ	0x00002F92	170

Table 135: Overview: Additional services of the EtherNet/IP Adapter

4.3.1 Get Module Status/ Network Status

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

Table 62 on page 58 lists all possible values of the Module Status (Parameter `ulModuleStatus` of the confirmation packet) and their meaning.

Table 63 on page 59 lists all possible values of the Network Status (Parameter `ulNetworkStatus` of the confirmation packet) and their meaning.

Packet Structure Reference

```
#define EIP_APS_GET_MS_NS_REQ_SIZE       0

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_APS_PACKET_GET_MS_NS_REQ_Ttag
{
  HIL_PACKET_HEADER_T             tHead;
} EIP_APS_PACKET_GET_MS_NS_REQ_T;
```

Packet Description

Variable	Type	Value / Range	Description
structure `HIL_PACKET_HEADER_T`			
ulDest	uint32_t	0x20	Destination
ulLen	uint32_t	0	Packet Data Length in bytes
ulSta	uint32_t	0	See Packet Structure Reference
ulCmd	uint32_t	0x360E	`EIP_APS_GET_MS_NS_REQ` - Command

Table 136: `EIP_APS_GET_MS_NS_REQ` – Get Module Status/ Network Status Request

Packet Structure Reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_APS_GET_MS_NS_CNF_Ttag
{
  uint32_t  ulModuleStatus;     /*!< Module Status \n
  uint32_t  ulNetworkStatus;     /*!< Network Status \n
} EIP_APS_GET_MS_NS_CNF_T;

#define EIP_APS_GET_MS_NS_CNF_SIZE  (sizeof(EIP_APS_GET_MS_NS_CNF_T))

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST EIP_APS_PACKET_GET_MS_NS_CNF_Ttag
{
  HIL_PACKET_HEADER_T      tHead;
  EIP_APS_GET_MS_NS_CNF_T  tData;
} EIP_APS_PACKET_GET_MS_NS_CNF_T;
```

Packet Description

Variable	Type	Value / Range	Description
structure `HIL_PACKET_HEADER_T`			
ulLen	uint32_t	EIP_APS_GET_MS_NS_CNF_SIZE	Packet Data Length in bytes
ulSta	uint32_t		See Packet Structure Reference
ulCmd	uint32_t	0x360F	`EIP_APS_GET_MS_NS_CNF` - Command
structure 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 62 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 63 for more information.

Table 137: `EIP_APS_GET_MS_NS_CNF` – Confirmation of Get Module Status / Network Status Request

4.3.2 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. Please refer to reference [2] for further information.

4.3.3 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. Please refer to reference [2] for further information.

4.3.4 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. Please refer to reference [2] for further information.

4.3.5 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 file system 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. Please refer to reference [2] for further information.

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 (see 4.2.11 “Store Remanent Data Indication”). In this case, the HIL_DELETE_CONFIG_REQ will not be confirmed until that indication is replied to by the host application.

4.3.6 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. Please refer to reference [2] for further information.

4.3.7 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. Please refer to reference [2] for further information.

4.3.8 Set Remanent Data Request

In case the host application is responsible to store remanent data (section *Remanent data* on page 88), the application must use this service during startup to provide the remanent data to the firmware. In case no remanent data is available (e.g. in case the system starts up for the very first time), the service shall be sent with the correct Component ID but with remanent data size set to zero. Otherwise, it is not ensured that the netX firmware starts up properly.

For a description of this service and the indication and respone packet, see reference [2]. For a state diagram, see section *Host Application Behavior* on page 78.

Value for ulComponentID

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

4.3.9 Set Trigger Type

The host application sends the HIL_SET_TRIGGER_TYPE_REQ packet to the stack in order 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 was reached: HIL_TRIGGER_TYPE_*_TIMED_ACTIVATION

All trigger modes are functionally independent and can be used individually or combined. Anyway, we recommend to decide for either a time-triggered or an event-triggered interface design and not to use 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).

Please refer to section 2.9 for a more specific description of the handshake modes supported by the EtherNet/IP stack.

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.

This request packet is used by the application to modify the trigger mode of the protocol stack:

Packet structure reference

```
#define HIL_SET_TRIGGER_TYPE_REQ                0x00002F90

/*!< No input data synchronization (free-run). */
#define HIL_TRIGGER_TYPE_PDIN_NONE                        0x0010
/*!< Input data will be updated when new data was received. (bus cycle synchronous). */
#define HIL_TRIGGER_TYPE_PDIN_RX_DATA_RECEIVED            0x0011
/*!< Input data will be updated on time event (time isochronous). */
#define HIL_TRIGGER_TYPE_PDIN_TIMED_ACTIVATION            0x0012

/*!< No output data synchronization (free-run). */
#define HIL_TRIGGER_TYPE_PDOUT_NONE                       0x0010

/*!< No sync signal generation */
#define HIL_TRIGGER_TYPE_SYNC_NONE                        0x0010
/*!< Generate Sync event when data shall be applied. */
#define HIL_TRIGGER_TYPE_SYNC_TIMED_ACTIVATION            0x0014

/*! Set data exchange trigger data. */
typedef struct HIL_SET_TRIGGER_TYPE_REQ_DATA_Ttag
{
  /*! Consumer data trigger type HIL_TRIGGER_TYPE_PDIN_*. */
  uint16_t usPdInHskTriggerType;
  /*! Provider data trigger type HIL_TRIGGER_TYPE_PDOUT_*. */
  uint16_t usPdOutHskTriggerType;
  /*! Synchronization trigger type HIL_TRIGGER_TYPE_SYNC_*. */
  uint16_t usSyncHskTriggerType;
} HIL_SET_TRIGGER_TYPE_REQ_DATA_T;

/*! Set data exchange trigger request. */
typedef struct HIL_SET_TRIGGER_TYPE_REQ_Ttag
{
  HIL_PACKET_HEADER_T               tHead;  /*!< Packet header. */
  HIL_SET_TRIGGER_TYPE_REQ_DATA_T   tData;  /*!< Packet data. */
} HIL_SET_TRIGGER_TYPE_REQ_T;
```

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
Data			
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 138: HIL_SET_TRIGGER_TYPE_REQ_T – Set Trigger Type request

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

Packet structure reference

```
#define HIL_SET_TRIGGER_TYPE_CNF                0x2F91

/*! Set data exchange trigger confirmation structure. */
typedef HIL_PACKET_HEADER_T      HIL_SET_TRIGGER_TYPE_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	0	Packet Data Length (in Bytes)
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x00002F91	HIL_SET_TRIGGER_TYPE_CNF

Table 139: HIL_SET_TRIGGER_TYPE_CNF_T – Set Trigger Type confirmation

4.3.10 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.

Packet structure reference

```
#define HIL_GET_TRIGGER_TYPE_REQ                0x2F92

/*! Get data exchange trigger request. */
typedef HIL_PACKET_HEADER_T      HIL_GET_TRIGGER_TYPE_REQ_T;
```

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 chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x00002F92	HIL_GET_TRIGGER_TYPE_REQ

Table 140: HIL_GET_TRIGGER_TYPE_REQ_T – Get Trigger Type request

Packet Structure Reference

```
#define HIL_GET_TRIGGER_TYPE_CNF                0x00002F93

/*! Get data exchange trigger data. */
typedef struct HIL_GET_TRIGGER_TYPE_CNF_DATA_Ttag
{
  /*! Input process data trigger type. */
  uint16_t usPdInHskTriggerType;
  /*! Output process data trigger type. */
  uint16_t usPdOutHskTriggerType;
  /*! Synchronization trigger type. */
  uint16_t usSyncHskTriggerType;
  /*! Minimal provide/consumer data update interval in free-run mode. */
  uint16_t usMinFreeRunUpdateInterval;
} HIL_GET_TRIGGER_TYPE_CNF_DATA_T;

/*! Get data exchange trigger confirmation structure. */
typedef struct HIL_GET_TRIGGER_TYPE_CNF_Ttag
{
  HIL_PACKET_HEADER_T               tHead;  /*!< Packet header. */
  HIL_GET_TRIGGER_TYPE_CNF_DATA_T   tData;  /*!< Packet data. */
} HIL_GET_TRIGGER_TYPE_CNF_T;
```

Packet description

Variable	Type	Value / Range	Description
ulLen	uint32_t	8	Packet Data Length (in Bytes)
ulSta	uint32_t		See chapter *Status/Error Codes Overview*
ulCmd	uint32_t	0x00002F93	HIL_GET_TRIGGER_TYPE_CNF
Data			
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 141: HIL_GET_TRIGGER_TYPE_CNF_T – Get Trigger Type confirmation

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.

The EtherNet/IP Adapter V5 currently supports at least the following tags:

Tag list parameter

Tag list	Parameter / Tag	Value	Description
Remanent Data responsibility	HIL_TAG_REMANENT_DATA_RESPONSIBLE	Disabled	Communication firmware stores remanent data (default).
		Enabled	Application stores remanent data. For a description, see section *Remanent data* on page 88.
Raw Ethernet/NDIS support	HIL_TAG_EIF_NDIS_ENABLE	Disabled	Raw Ethernet/NDIS support is disabled (default).
		Enabled	Raw Ethernet/NDIS support is enabled on DPM communication channel 1 Fourth DDP MAC address will be used
Initial DDP mode after power on	HIL_TAG_DDP_MODE_AFTER_STARTUP	Active	The Device Data Provider is active from the very beginning, providing data from the Flash Device Label (FDL) or SecMem data sources.
		Passive	The Device Data Provider is passive until set to "active" by the application, after having set base device information, e.g. MAC addresses.

Table 142: Tag list parameter

6 Status/Error Codes Overview

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. See also General Error Code and Extended Error Code for more details.
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.

Hexadecimal Value	Definition Description
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
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 143: 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 144: 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 145: 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.

General Status Code (specified hexadecimally)	Status Name	Description
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.
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 146: General Error Codes according to CIP Standard

7 Appendix

7.1 List of Figures

7.2 List of Tables

7.3 Legal Notes

Copyright

Important notes

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

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

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

Liability disclaimer

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

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

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

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

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

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

Warranty

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

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

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

Costs of support, maintenance, customization and product care

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

Additional guarantees

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

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

Confidentiality

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

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

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

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

Export provisions

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

7.4 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:

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

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. 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.
3. 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.

7.5 Contacts

Headquarters

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

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

Subsidiaries

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

Support
Phone: +86 (0) 21-6355-5161
E-Mail: cn.support@hilscher.com

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

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

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