Protocol API

EtherNet/IP Adapter

V2.15.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC060301API22EN | Revision 22 | English | 2021-05 | Released | Update 01 | Public

Table of Contents

1 Introduction

1.1 Abstract

This manual describes the user interface of the EtherNet/IP Adapter implementation on the netX chip. The aim of this manual is to support the integration of devices based on the netX chip into own applications based on driver functions or direct access to the dual-port memory.

The general mechanism of data transfer, for example how to send and receive a message or how to perform a warmstart is independent from the protocol. These procedures are common to all devices and are described in the 'netX DPM Interface manual'.

1.2 List of Revisions

Rev	Date	Name	Revisions
21	2019-10-07	MB, KM	Firmware/stack version V2.14.0 ▪ Section Technical Data: Feature Null Forward Open added. ▪ Section Identity Object (Class Code: 0x01), subsection Instance Attributes: Bit 2 of attribute 5 is now settable by the host application. ▪ Section Connection Manager Object (Class Code: 0x06), subsection Supported Services: Sercives Forward_Open and Forward_Clsoe added. ▪ Chapter Non-Null Forward Open and Null Forward Open added. ▪ Section Example Configuration Process removed. ▪ Section "*Optional sequence count handling*" of packet EIP_OBJECT_CL3_SERVICE_IND removed. ▪ Define value 0 for input and output assmeblies in packet `EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ/CNF` – Configure the Device with Configuration Parameter ▪ Correct ulCmd field of Table 142 (`EIP_OBJECT_FWD_OPEN_FWD_RES` – Response of Forward_Open indication) ▪ Clarification of Object revision of TCP/IP Interface object. Revision is 4. ▪ Clarification of Object revision of QoS object. Revision is 1.
22 Update 01	2021-03-02	KMI	Firmware/stack version V2.15.0 ▪ Added description of new parameter flag (Table 120) and handling of AOT and COS connections (6.2.14.1) ▪ Added description of packet `EIP_OBJECT_AS_TRIGGER_TYPE_IND/RES` – Indication of the currently used trigger type (6.2.15) ▪ Added clarification for the behavior of EIP_OBJECT_CONNECTION_CONFIG_IND in case ForwardOpen/Close forwarding is used ▪ Added new assembly flag EIP_AS_FLAG_FORWARD_SEQUENCE_COUNT ▪ Added new packet EIP_APS_SET_MODULE_STATUS_REQ ▪ Added new parameter flag EIP_OBJECT_PRM_APPLICATION_CONTROLS_IDENTITY_STATE_ATTRIBUTE ▪ Decrease packet data size (abData) to 1400 bytes: EIP_OBJECT_CIP_OBJECT_CHANGE_IND, EIP_OBJECT_FWD_OPEN_FWD_IND, EIP_OBJECT_FWD_OPEN_FWD_RES, EIP_OBJECT_FWD_CLOSE_FWD_IND, EIP_OBJECT_FWD_CLOSE_FWD_RES ▪ Increase object change indication timeout value from 3 to 10 seconds ▪ Support of Identity Object revision 2 ▪ The host application is now able to write the data (attribute 3) of a configuration assembly instance (PSEIPCORE-275)

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
- operating system rcX

1.4 Intended Audience

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

- Knowledge of the TCP/IP Protocol Interface Manual
- 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 V2.15.0.

This firmware/stack has been written to meet the requirements of a subset outlined in the CIP Vol. 1 and CIP Vol. 2 specifications.

1.5.1 Technical Data

Maximum number of input data	504 bytes per assembly instance	
Maximum number of output data	504 bytes per assembly instance	
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*	
Explicit Messages	Connected and unconnected	
Max. number of connections	8 (sum of connected explicit and implicit connections)	
Max. number of user specific objects	20	
Unconnected Message Manager (UCMM)	supported	
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)
	Time Sync Object	(0x43)
DHCP	supported	
BOOTP	supported	
Baud rates	10 and 100 MBit/s	
Duplex modes	Half Duplex, Full Duplex, Auto-Negotiation	
MDI modes	MDI, MDI-X, Auto-MDIX	
Data transport layer	Ethernet II, IEEE 802.3	

ACD	supported (from firmware version 2.4.1)
DLR V2 (ring topology)	supported
Quick Connect	supported
CIP Sync	supported
Integrated switch	supported
Reset services	Identity Object Reset Service of Type 0 and 1
Integrated Web Server	supported (since firmware version 2.5.15, for details of Web Server, see reference #5)
Null Forward Open	supported

* depending on number of connections and number of input and output data

Firmware/stack available for netX

netX 50	yes
netX 51	yes (from firmware version 2.7.4)
netX 100, netX 500	yes

PCI

DMA Support for PCI targets	yes

Slot Number

Slot number supported for	CIFX 50-RE, CIFX 50E-RE

Configuration

- Configuration by tool SYCON.net (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

1.5.2 Limitations

- Symbolic TAGs are not supported

1.5.3 Protocol Task System

To manage the EtherNet/IP implementation six tasks are involved into the system. To send packets to a task, the task main queue have to be identifier. For the identifier for the tasks and their queues are the following naming conversion:

Task Name	Queue Name	Description
EIS_ENCAP_TASK	ENCAP_QUE	Encapsulation Layer
EIS_OBJECT_TASK	OBJECT_QUE	EtherNet/IP Objects
EIS_CL1_TASK	QUE_EIP_CL1	Class 1 communication
EIS_TCPUDP	EN_TCPUDP_QUE	TCP/IP Task
EIP_DLR	QUE_EIP_DLR	DLR Task
EIS_APS_TASK	DPMINTF_QUE	Dual Port Memory Interface or Application Task Slave
PTP_TASK	No Queue available	Precision Time Protocol Task

Table 2: Names of Tasks in EtherNet/IP Firmware

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
ASCII	American Standard Code for Information Interchange
BOOTP	Boot Protocol
CC	Connection Configuration Object
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)
DMA	Direct Memory Access
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)
PCI	Peripheral Component Interconnect
QoS	Quality of Service
RPI	Requested Packet Interval
SNN	Safety Network Number
TCP	Transmission Control Protocol
UCMM	Unconnected Message Manager
UDP	User Datagram Protocol
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 References to documents

This document is based on the following specifications:

[1] Hilscher Gesellschaft für Systemautomation mbH: Dual-Port Memory Interface Manual, netX Dual-Port Memory Interface, DOC060302DPM17EN, Revision 17, English, 2020-06.

[2] Hilscher Gesellschaft für Systemautomation mbH: Packet API, netX Dual-Port Memory, Packet-based services (netX 10/50/51/52/100/500), DOC161001API04EN, Revision 4, English, 2020-06.

[3] Hilscher Gesellschaft für Systemautomation mbH: Protocol API, TCP/IP, Packet Interface, V2.6, DOC050201API17EN, Revision 17, English, 2020-10.

[4] ODVA: The CIP Networks Library, Volume 1, “Common Industrial Protocol (CIP™)”, Edition 3.29, English, November 2020.

[5] ODVA: The CIP Networks Library, Volume 2, “EtherNet/IP Adaptation of CIP”, Edition 1.26, English, April 2020.

[6] Hilscher Gesellschaft für Systemautomation mbH: Application Note, Functions of the Integrated WebServer, DOC091203AN06EN, Revision 6, English, 2017-09.

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

[8] IEEE 1588 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems, Revision 2, 2008

[9] Hilscher Gesellschaft für Systemautomation mbH: Application Note, EtherNet/IP Adapter, CIP Sync, V2.9/V3.2 and higher, DOC130104AN05EN, Revision 5, English, 2016-09.

2 The Common Industrial Protocol (CIP)

This chapter introduces the EtherNet/IP protocol as a member of the CIP network family of protocols. It covers mainly the same topics as the paper *"The Common Industrial Protocol (CIP™) and the Family of CIP Networks*" published by the ODVA which is recommended for more detailed information, see reference [7] listed on page 12 of this document.

2.1 Introduction

Currently, the requirements for networks used in manufacturing enterprises are massively changing. These are some of the most important impacts:

- The lack of scalable and coherent enterprise network architectures ranging from the plant floor level to enterprise level (This causes numerous specialized - and often incompatible – network solutions.)
- Adoption of Internet technology
- Company-wide access to manufacturing data and seamless integration of these data with business information systems
- Demand for open systems

From the ODVA's point of view, common application layers are the key to true network integration. Therefore, the ODVA (jointly with ControlNet International) offers a concept for advanced communication based on common application layers. namely the **Common Industrial Protocol (CIP™)**.

These are the main advantages of CIP:

- CIP allows complete integration of control with information, multiple CIP Networks and Internet technologies.
- CIP uses a media-independent platform providing seamless communication from the plant floor to enterprise level with a scalable and coherent architecture,
- CIP allows integration of I/O control, device configuration and data collection across multiple networks.
- CIP decreases engineering and installation time and costs while maximizing ROI.

2.1.1 CIP-based Communication Protocols

A couple of communication protocols have been developed as part of the CIP network family of communication protocols.

Table 4 provides an overview on these:

Protocol name	Year of introduction	Main facts
DeviceNet™	1994	CIP implementation using the popular Controller Area Network (CAN) data link layer. CAN according to ISO 1189810 defines only layers 1 and 2 of the OSI 7-layer model. DeviceNet covers the remaining layers. Advantages: Low cost of implementation, easy to use, many device manufacturers offer DeviceNet capable products. Vendor organization: Open DeviceNet Vendor Association (ODVA, http://www.odva.org).
ControlNet™	1997	new data link layers compared to DeviceNet that allow for much higher speed (5 Mbps), strict determinism and repeatability extending the range of the bus (several kilometers with repeaters) for more demanding applications. Vendor organization: ControlNet International (CI, http://www.controlnet.org)
EtherNet/IP	2000	EtherNet/IP is the CIP implementation based on TCP/IP. It can therefore be deployed over any TCP/IP supported data link and physical layers, such as IEEE 802.311 (Ethernet). Easy future implementations on new physical/data link layers possible.
CompoNet	2006	CompoNet provides a bit-level network to control small, high speed machines and the CIP Network services to connect to the plant and the enterprise. CompoNet is especially designed for applications using large numbers of simple sensors and actuators by CompoNet provides high speed communications with configuration tools Efficient construction, Simple set-up, High availability. CompoNet uses Time Division Multiple Access ("TDMA") in its network layer. CompoNet includes an option for power (24V DC, 5A) and signal in the same cable with the ability to remove and replace nodes under power.

Table 4: Network Protocols for Automation offered by the CIP Family of Protocols

Among these, EtherNet/IP is the CIP implementation based on TCP/IP.

Note that CIP is independent from physical media and data link layer.

The overall relationship between these main implementations of CIP and the ISO/OSI 7-layer model is shown in

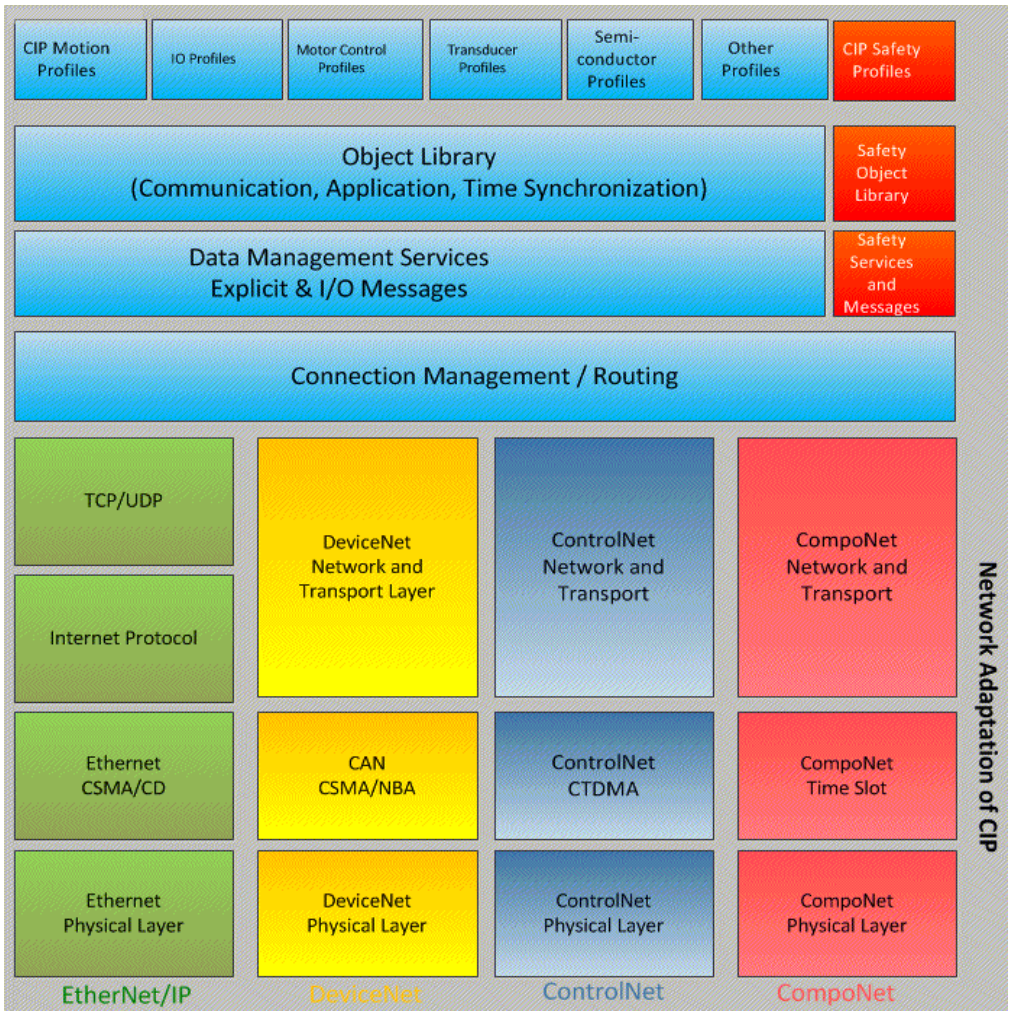

Table 5: The CIP Family of Protocols

2.1.2 Extensions to the CIP Family of Networks

2.1.2.1 CIP Safety

For achieving functional safety for CIP Networks, CIP Safety has been introduced in 2004. It provides users with fail-safe communication between devices, controllers and networks for safety applications.

CIP Safety is a protocol extension that allows the transmission of safety relevant messages. Such messages are governed by additional timing and integrity mechanisms that are guaranteed to detect system flaws to a very high degree, as required by international standards such as IEC 6150814. If anything goes wrong, the system will be brought to a safe state, typically taking the machine to a standstill.

2.1.2.2 CIP Sync and CIP Motion

Two other significant extensions to CIP are CIP Sync and CIP Motion. CIP Sync allows synchronization of applications in distributed systems through precision real-time clocks in all devices. Tight synchronization of these real-time clocks is achieved using the IEEE 1588 standard. The CIP Sync technology provides the ideal basis for motion control applications such as CIP Motion.

CIP Sync is the time synchronization technology for the Common Industrial Protocol (CIP). This technology allows accurate real-time synchronization of devices and controllers connected over CIP networks that require

- time stamping,
- recording sequences of events,
- distributed motion control,
- increased control coordination.

CIP Sync uses the time synchronization technology as defined in standard “IEEE 1588 - Precision Clock Synchronization Protocol for Networked Measurement and Control Systems” which is described in reference [8] and [9].

The main components of CIP Sync are:

- The Precision Time Protocol defined in IEEE 1588:2008. It is a network protocol providing a standard mechanism for time synchronization of communicating clocks across a network of distributed devices.
- The Time Sync object (CIP class ID 0x43) providing a CIP interface to the IEEE 1588 standard.

A more detailed description of this CIP extension with respect to the EtherNet/IP protocol stack from Hilscher is given in the Application Note EtherNet/IP Adapter CIP Sync .

Ordinary devices can operate with CIP Sync or CIP Safety devices simultaneously in the same system. There is no need for strict segmentation into “*Standard*”, “*Sync*” and “*Safety*” networks. It is even possible to combine all three functions in one device.

This chapter focuses on the following aspects of CIP:

- Object Modeling (2.2)
- Services (2.3)
- Messaging Protocol (2.4)
- Object Library (2.5)
- Device profiles (2.7)
- Electronic Data Sheets (2.8)

CIP Sync is expected to be supported by the EtherNet/IP Adapter protocol stack beginning with version 2.8.

2.1.3 Special Terms used by CIP

As CIP uses a producer/consumer architecture instead of the often used client/server architecture, some special terms in this context should be explained here precisely.

Client — A client is a device sending a request to another node on the network (the server) and expecting a response from the server.

Server — A server is a device receiving a request from another node on the network (the server) and reacting by sending a response to the client.

Producer — According to the CIP specification, a producer is a network node which is responsible for transmitting data. It places a message on the network to be consumed by one or more consumers. The produced message is not directed to a specific consumer (implicit messaging). Instead, the producer sends the data packets along with a unique identifier for the contents of the packet.

Consumer — According to the CIP specification, a consumer is a network node (not necessarily the only one) which receives data from a device acting as a producer on the network (implicit messaging). All interested nodes on the network can access the contents of the packet by filtering for the unique identifier of the packet.

Producer/Consumer Model — The producer/consumer model uses an identifier-based addressing scheme in contrast to the traditional source/destination message addressing scheme which is applied in conjunction with the client/server architecture (see *Figure 1 and* Figure 5).

It offers the following advantages:

1. It is very efficient as it increases the information flow while it decreases the network load.
2. It is very flexible.
3. It can easily handle multicast communication.

The network nodes decide on their own whether to consume or not to consume the data in the corresponding message.

Figure 1: Source/Destination vs. Producer/Consumer Model

Explicit Message

Explicit messages are used within CIP for point-to-point and client/server connections. They contain addressing and service information causing execution of a specific service on a specific part of the network node.

An explicit data transmission protocol is used in the data fraction of the explicit message packet.

Explicit messages can either be connection-oriented or connection-less.

Implicit (I/O) Message

Implicit messages do not contain any transmission protocol in their IO data, for instance there is not any address and/or service information. A dynamically generated unique connection ID allows reliable identification. The data format has already been specified in the EDS file previously. Thus the efficiency of data transmission is improved as the meaning of the data is already known.

Implicit messages can only be connection-oriented. There are no connection-less implicit messages defined within CIP.

Data transmission for implicit messages can be initiated cyclically (by clock/timer) or based on change-of state.

For more details on explicit and implicit messages also see section 2.4.5 “*Types of Ethernet/IP Communication*” on page 31.

2.2 Object Modeling

CIP is based on abstract object modeling. Every device in a CIP network is modeled as a collection of objects.

According to the CIP Specification, an object provides an abstract representation of a particular component within a product. Therefore, anything not described in object form is not visible through CIP.

CIP objects can have the following structured elements:

- classes,
- instances,
- attributes.

Furthermore, objects may contain services offering a well-defined functionality.

A class is a set of objects all representing the same kind of system component. Each class has a unique Class ID number in the range between 1 and 65535. The CIP specification defines an own library of standard objects (described in Part 5 of references). It also offers the possibility to extend the object model by defining own objects.

Sometimes it is necessary to have more than one “copy” of a class within a device. Each such “copy” is denominated as an instance of the given class.

Objects have data variables associated with them. These are called the attributes of the particular object. Typically attributes provide status or govern the operation of the object. To each attribute of an object, an Attribute ID number in the range between 0 and 255 is assigned

There are two kinds of attributes, namely instance and class attributes.

This means, an instance of a particular object is the representation of this object within a class. Each instance has the same set of attributes, but has its own set of attribute values, which makes each instance unique. Instances have a unique Instance ID number (range: 1-65535).

In this context, also see *Figure 2: A class of objects.*

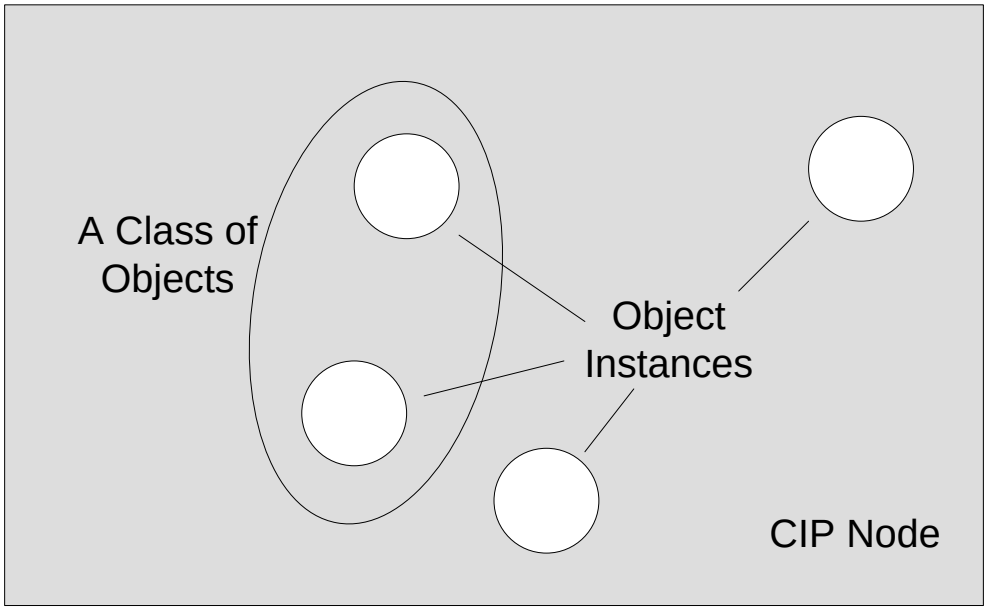

Figure 2: A class of objects

In addition to the instance attributes, there is also another kind of attributes an object class may have, namely the class attributes. These represent attributes that have class-wide scope. I.e. they describe properties of the entire object class, e.g., the number of existing instances of this particular object or the class revision. Class attributes have the instance ID 0.

Uniform Addressing Scheme

Addressing of objects and their components is accomplished by a uniform addressing scheme. The following information is necessary to address data inside a device via the network.

Item	Description
Node Address	An integer identification value assigned to each node on a CIP Network. On EtherNet/IP, the node address is the IP address.
Class Identifier (Class ID)	An integer identification value assigned to each object class accessible from the network.
Instance Identifier (Instance ID)	An integer identification value assigned to an object instance that identifies it among all instances of the same class.
Attribute Identifier (Attribute ID)	An integer identification value assigned to a class or instance attribute.
Service Code	An integer identification value which denotes an action request that can be directed at a particular object instance or object class.

Table 6: Uniform Addressing Scheme

This kind of addressing is used for instance in explicit messaging and also in the internal binding of one object to another. Identification of configurable parameters in the Electronic Device Sheets (EDS files) is also in the same way.

Figure 3 shows the addressing scheme.

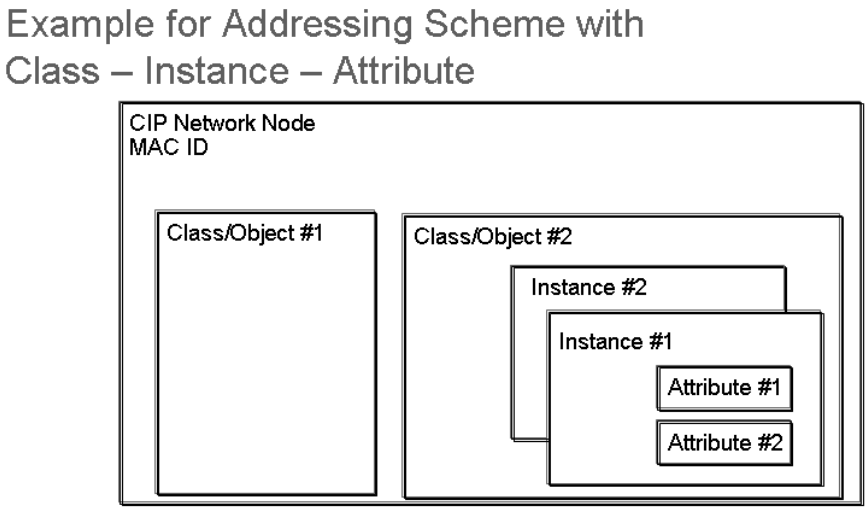

Figure 3: Example for Addressing Schema with Class – Instance – Attribute

According to the CIP Specification (reference [4]), the ranges of the following *Table 7: Ranges for Object Class Identifiers* apply for object class identifiers:

Range of object class identifiers	Meaning
0…0x63	Area for publicly defined objects
0x64…0xC7	Area for vendor-specific objects
0xC8…0xEF	Reserved for future use by ODVA/CI
0xF0…0x2FF	Area for publicly defined objects
0x300…0x4FF	Area for vendor-specific objects
0x500…0xFFFF	Reserved for future use by ODVA/CI

Table 7: Ranges for Object Class Identifiers

For attribute identifiers, the following table applies:

Range of attribute identifiers	Meaning
0…0x63	Area for publicly defined objects
0x64…0xC7	Area for vendor-specific objects
0xC8…0xFF	Reserved for future use by ODVA/CI

Table 8: Ranges for Attribute Identifiers

Figure 4 shows an example on how an object attribute is addressed in CIP.

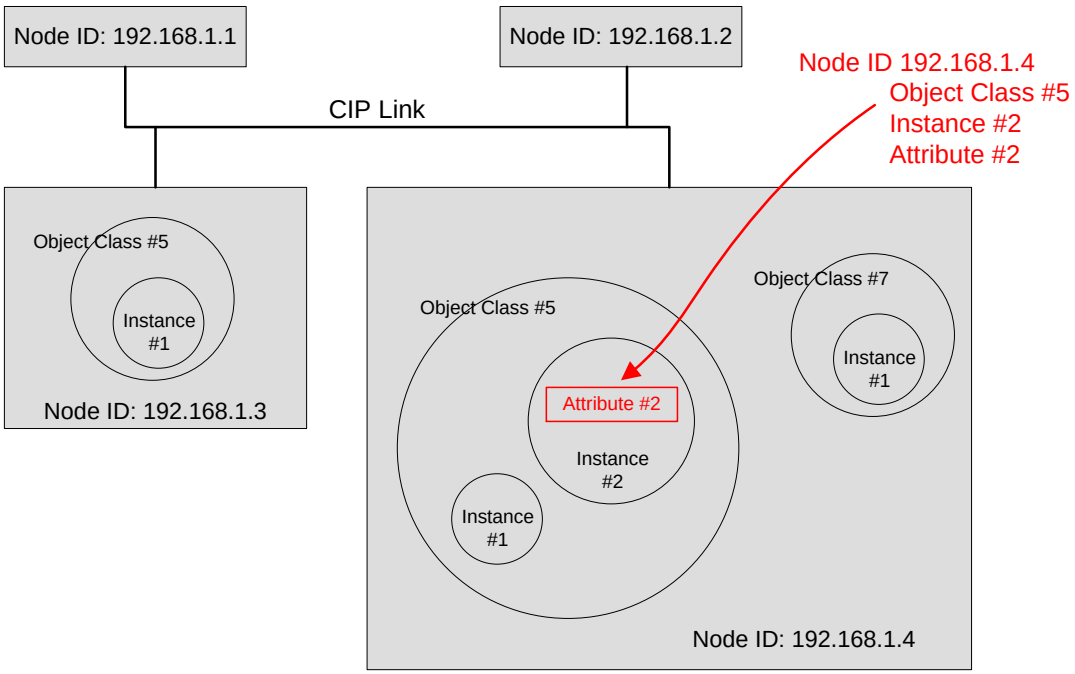

Figure 4: Object Addressing Example

2.3 Services

Objects have associated functions called services. Services are used at explicit messages (also see section *Explicit Messaging* on page 35). Services are identified by their service codes defining the kind of action to take place when an object is entirely or partly addressed through explicit messages according to the addressing scheme (see Table 6 page 21 and Figure 3 page 21).

As Table 9 explains, there are in general three kinds of service available to which specific ranges of service code identifiers have been associated:

Range of service identifiers	Description
0x00 … 0x31	Range for CIP Common Services
0x32 … 0x4A	Range for vendor-specific services
0x4B … 0x63	Range for object class-specific services
0x64 … 0xFF	Reserved

Table 9: Ranges for Service Codes

Besides simple read and write functions, a set of more sophisticated CIP Common Services has been defined within the CIP specification. These services (0x00 … 0x31) may be used in all kinds of CIP networks. They may be applicable or not applicable to a specific object depending on the respective context. Some times the meaning also depends from the class.

In general, the following service codes for CIP Common Services are defined within the CIP specification:

Numeric value of service code	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
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 10: Service Codes according to the CIP specification

2.4 The CIP Messaging Model

CIP (and thus EtherNet/IP) separates between two standard types of messaging: implicit and explicit messaging (see section *Types of Ethernet/IP Communication* on page 31, especially Table 14: Comparison of basic Types of Ethernet/IP Communication: Implicit vs. Explicit Messaging). Additionally, we have to separate between connected and unconnected messaging.

2.4.1 Connected vs. Unconnected Messaging

Connected messaging has the following characteristics.

- Resources are reserved.
- It reduces data handling upon receipt of messages.
- Supports the producer-consumer model and time-out handling
- Explicit and implicit connections available
- It is a controlled connection.
- A connection needs to be configured.
- There is the risk that a node is running out of applicable connections.

Unconnected messaging has the following characteristics.

- Unconnected messaging must be supported on every EtherNet/IP device (minimum messaging requirement a device has to support) and is therefore always available.
- The resources are not reserved in advance, so there is no reservation mechanism at all.
- No configuration or maintenance required.
- The message can be used only when needed.
- It supports all explicit services defined by CIP.
- More overhead per message
- It is mainly used for low-priority messages occurring once or not frequently.
- It is also used during the connection establishment process of connected messaging

2.4.2 Connection Transport Classes

The CIP specification defines seven transport classes (Class 0 to Class 6) of which the following are applicable in the EtherNet/IP context:

- Implicit (Cyclic real-time communication, Producer/Consumer)
  - Transport Class 0
  - Transport Class 1

  These transport classes differ in the existence (Class 1) or absence (Class 0) of a preceding 16 bit sequence count value used for avoiding duplicate packet delivery.
- Explicit (Acyclic non-real-time communication, Client/Server)
  - Transport Class 3

  Class 3 connections are transport classes for bidirectional communication which are appropriate for the client-server model.

2.4.3 Connection Establishment, Timeout and Closing

A CIP connection is established by the EtherNet/IP Scanner (Master). In order to do so, the scanner sends a *Forward_open* request to the EtherNet/IP Adapter. This request includes such information as:

- Identity of originator (Vendor ID, serial number of the connection)
- Timeout information for the connection to be established
- Connection Parameters:
  - Connection Type
  - Priority
  - Connection Size
- Production Trigger
- Transport Class
- Requested speed of data transmission (Request Packet Interval - RPI)
- Connection Path (target assembly instances also called connection points)

When the EtherNet/IP Adapter receives a *Forward_open* request, the protocol stack establishes the connection on its own using the information received from the EtherNet/IP Scanner. If it succeeds, it sends the `EIP_OBJECT_CONNECTION_IND` indication with `ulConnectionState` = `EIP_CONNECTED` = 1 to the application.

When the EtherNet/IP Adapter receives a *Forward_close* request, the connection is closed and connection-related data is cleared. The stack sends an `EIP_OBJECT_CONNECTION_IND` indication with `ulConnectionState` = `EIP_UNCONNECT` = 0 to the application. The indication is also sent when the connection times out.

When talking about CIP connections in the EtherNet/IP context often the terms “target” and “originator” are used. The originator is the device that sends the *Forward_Open* frame to the Target, which then returns the frame to the originator. Usually, a scanner originates a connection and the adapter is the target.

On EtherNet/IP a *Forward_Open* frame usually establishes two connections at the same time, one in the O→T direction and one in the T→O direction.

This is why a scanner has to provide at least two connection points (assembly instances) in order to open a connection. In Figure 6 for example the scanner can use the assembly instances #1 and #2. #1 is the instance that is used for the T→O direction (the adapter sends data to the originator thus produces data on the network) and #2 can be used for the O→T direction (the adapter receives data from the originator thus consumes data from the network).

These connection points are transmitted via the *Forward_Open* in the “Connection Path” field.

The following table gives an overview about the most important parameters that are sent along with the Forward_Open frame.

Parameters Name		Description
Connection Timeout Multiplier		The Connection Timeout Multiplier specifies the multiplier applied to the RPI to obtain the connection timeout value.
O→T RPI		Originator to Target requested packet rate. This is the cycle time the Originator uses to send I/O frames as soon as the connection has been established.
O→T Network Connection Parameters	Connection Type	This field specifies whether the I/O frames are sent as Point to Point or as Multicast
	Connection Size	The size, in bytes, of the data of the connection. The connection size includes the sequence count and the 32-bit real time header, if present. (See section 2.4.3.1 " Real Time Format")
T→O RPI		Target to Originator requested packet rate. This is the cycle time the Target uses to send I/O frames as soon as the connection has been established.
T→O Network Connection Parameters	Connection Type	This field specifies whether the I/O frames are sent as Point to Point or as Multicast
	Connection Size	The size, in bytes, of the data of the connection. The connection size includes the sequence count and the 32-bit real time header, if present. (See section 2.4.3.1 "Real Time Format")
Transport Type/Trigger	Trigger	Cyclic, Change Of State, Application Triggered
	Class	Class 0 / Class 1
Connection Path		Specifies the addressed assembly instances (connection points) Usually, the following order is used: 1) Configuration Assembly Instance 2) Output Assembly Instance (O→T) 3) Input Assembly Instance (T→O)

Table 11: Forward_Open Frame – The Most Important Parameters

What assembly instances are available in the device must be provided with the EDS file. Additionally, all available connections that can be established to the device must be provided in the [Connection Manager] section.

There are two further elements concerning Ethernet/IP connections:

- Real Time Format
- Connection Application Types

These elements are described in the following sections.

2.4.3.1 Real Time Format

Every connection has a pre-defined Real Time Format, which is the format of the data in the O→T and T→O direction. What Real Time Format shall be used is not specified in the *Forward_Open*, but in the [Connection Manager] section of the EDS file. Although the Real Time Format is not provided in the *Forward_Open* frame, it still has influence on the connection sizes within the network connection parameters.

The following Real Time Formats are available:

- 32-Bit Header Format (includes run/idle notification)
- Modeless Format (no run/idle notification)
- Heartbeat Format (no run/idle notification)

2.4.3.2 32-Bit Header Format

The 32 bit header real time format includes 0-n bytes of application data prefixed with 32 bits of header.

The 32-bit real time header format prefixed to the real-time data shall be the following form:

Bits 4-32	Bits 2-3	Bit 1	Bit 0
Reserved	ROO	COO	Run/Idle

Table 12: 32-Bit Real Time Header

The run/idle flag (bit 0) shall be set (1 = RUN) to indicate that the following data shall be sent to the target application. It shall be clear (0 = IDLE) to indicate that the idle event shall be sent to the target application.

The ROO and COO fields (bits 1-3) are used for the connection application type “Redundant Owner” which is not supported by the Hilscher EtherNet/IP Stack.

A class 0 32-bit header real time packet format is:

32-bit real time header	0-n bytes of application data

A class 1 32-bit header real time packet format is:

2 bytes sequence count	32-bit real time header	0-n bytes of application data

2.4.3.3 Modeless Format

The modeless real time format may include 0-n bytes of application data and there is no run/idle notification included with this real time format.

A class 0 modeless real time packet format is:

0-n bytes of application data

A class 1 modeless real time packet format is:

2 bytes sequence count	0-n bytes of application data

2.4.3.4 Heartbeat Format

The heartbeat real time format includes 0 bytes of application data and there is no run/idle notification included with this real time format.

A class 0 heartbeat real time packet format is:

0 bytes of application data

A class 1 heartbeat real time packet format is:

2 bytes sequence count	0 bytes of application data

2.4.4 Connection Application Types

The application type shall determine the target behavior concerning the relationship between different connections each sharing a producer (the same producing assembly instance).

The Hilscher EtherNet/IP Stack supports three different connection application types:

- Exclusive Owner
- Input Only
- Listen Only

One difference between these types is related to the real time format of the data that is transmitted (see section 2.4.3.1 " Real Time Format"). Where Exclusive Owner connections usually have I/O data in both directions, Input Only and Listen Only connections only have I/O data in the T→O direction.

Another characteristic of these connection types is the condition, under which the connection of a particular type can be established. While Exclusive Owner and Input Only connections can always be created, Listen Only connections can only be established if an Exclusive Owner or Input Only connection is already running.

The following table explains the relationship of connections with different application types. The table shows a 1st and a 2nd connection. For each pair of connections it is assumed that the 1st connection is established followed by t he 2nd connection. The column "Expected Result of 2nd Connection" provided the result of the Forward_Open Response when trying to establish the 2nd connection. The last two columns show the behavior of the 2nd connection when the 1st connection times out or is closed.

1st Connection	2nd Connection	Expected Result of 2nd Connection	Timeout of 1st Connection	Close of 1st Connection
IO	EO	Success	EO stays open	EO stays open
IO	IO	Success	2nd IO stays open	2nd IO stays open
IO	LO	Success	LO closes	LO closes
EO	IO	Success	IO closes	IO stays open
EO	LO	Success	LO closes	LO closes
EO	EO	Error 1)	-	-
LO	-	Error	-	-

EO = Exclusive Owner, IO = Input Only, LO = Listen Only

1) Assuming the O→T connection path entry is the same of the 1st and 2nd connection

Table 13: Relationship of Connections with Different Application Connection Types

2.4.4.1 Exclusive Owner Connection

An Exclusive Owner connection is not dependent on any other connection for its existence. A target only accepts one exclusive owner connection per O→T connection point.

The term connection owner refers to the connection originator whose O→T packets are being consumed by the target object. The term owning connection shall refer to the connection associated with connection owner.

When an exclusive owner connection timeout occurs in a target device, the target device stops sending the associated T→O data. The T→O data will not be sent even if one or more input only connections exist. This requirement exists to signal the originator of the exclusive owner connection that the O→T data is no longer being received by the target device.

Most common Real Time Format:

O→T: 32-Bit Run/idle Header

T→O: Modeless

Most Common Connection Types:

O→T: Point-2-Point

T→O: Point-2-Point /Multicast

2.4.4.2 Input Only Connection

An Input Only connection is not dependent on any other connection for its existence.

The O→T data uses the heartbeat format as described in section 2.4.3.1 „Real Time Format“. A target may accept multiple input only connections which specify the same T→O path. In addition, the target may accept listen only connections that use the same multicast T→O data.

Most common Real Time Format:

O→T: Heartbeat

T→O: Modeless

Most Common Connection Types:

O→T: Point-2-Point

T→O: Point-2-Point /Multicast

2.4.4.3 Listen Only Connection

A Listen Only connection is dependent on a non-Listen only application connection for its existence. The O=>T connection shall use the heartbeat format as described in section 2.4.3.1 „Real Time Format“. A target may accept multiple listen only connections which specify the same T→O path. If the last connection on which a listen only connection depends is closed or times out, the target device stops sending the T→O data which will result in the listen only connection being timed out by the originator device.

Most common Real Time Format:

O→T: Heartbeat

T→O: Modeless

Most Common Connection Types:

O→T: Point-2-Point

T→O: Multicast

2.4.5 Types of Ethernet/IP Communication

The following table introduces the two basic types of Ethernet/IP Communication by comparing their most important characteristics:

CIP Message Type	**Explicit**		**Implicit**	
CIP Communication Relationship	**Unconnected**	**Connected**	**Connected**	
Point-to-point or multicast	**Point-to-point**		**Point-to-point**	**Multicast**
Communication Model	Client-Server		Producer-Consumer	
Communication Type	Acyclic Requests and replies, execution of services		Cyclic IO data transfer	
Typical Use/ Example	Data of lower priority and time criticality / Configuration data and diagnostic data		Time-critical real-time data / IO data	
Involved object	Message router object, UCMM		Assembly object	
Transport Protocol	TCP/IP		UDP/IP	
Transport Class	None	Class3	Class0, Class1	

Table 14: Comparison of basic Types of Ethernet/IP Communication: Implicit vs. Explicit Messaging

In the following, implicit and explicit messaging is discussed in more detail.

2.4.6 Implicit Messaging

Implicit messaging is used for cyclic communication, i.e. for periodically repeated transmission of data with the same structure. It has the following characteristics:

- the meaning of transferred data is known at both connection endpoints. Therefore,
- the data can be sent with only a minimum of information overhead.
- Operation is always in connected mode.
- Different transmission triggers available.
- Typically, this kind of communication is multi-cast communication (unicast possible as well).

There are three mechanisms how the data exchange can be triggered, the so called production triggers:

- Cyclic: Messaging is triggered periodically with a specified repetition time (packet rate).
- Change of State (COS): Messaging is triggered by the change of a specific state.
- Application-triggered: Messaging is triggered by the application.

Implicit Messages are based on the producer-consumer model, which supports multicast and unicast (Point-2-Point) messaging.

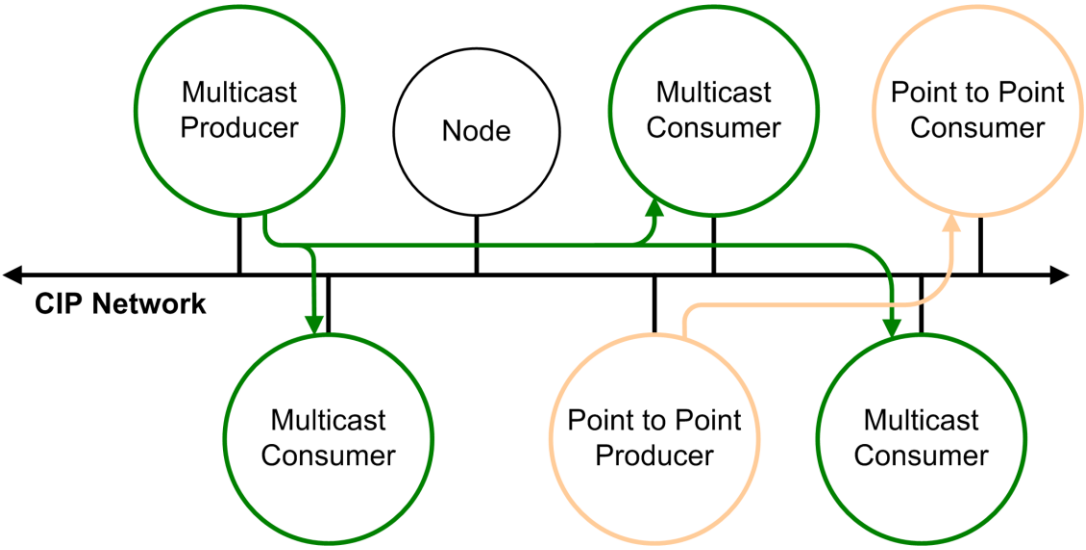

Figure 5: Producer Consumer Model – Point-2-Point vs. Multicast Messaging

2.4.6.1 Structure of Transmitted I/O Data

When opening a CIP I/O connection a scanner usually connects to a pair of assembly instances, also called connection points. Each assembly instance comes with a specific data structure. For example the data of an assembly instances can combine attributes of other object attributes. The following figure illustrates this.

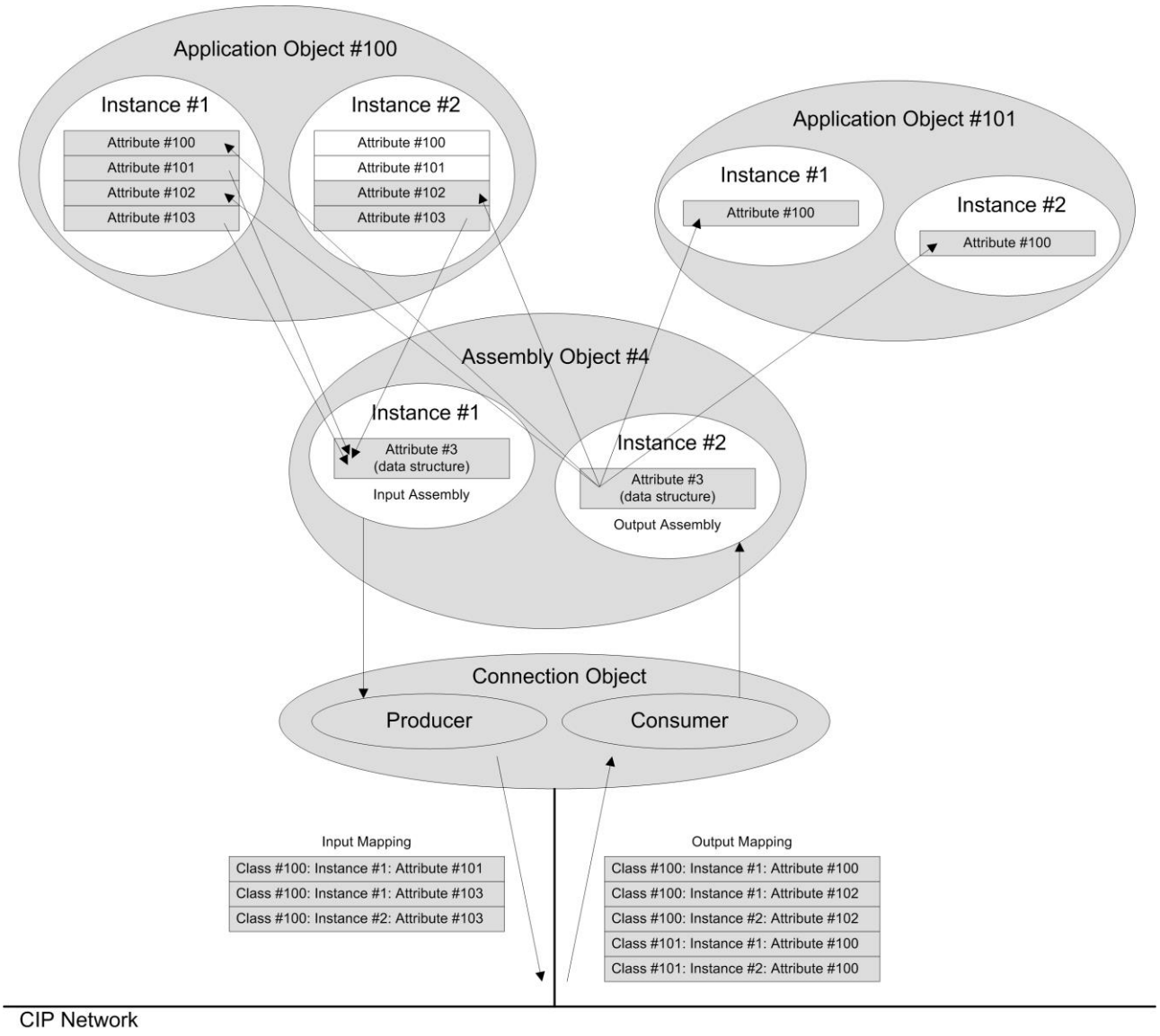

Figure 6: Example of possible Assembly Mapping

This accelerates the access to the IO data by maximizing the efficiency of IO data access. Working with assemblies makes the IO or configuration data available as one single block. This improves the IO performance significantly.

Assembly instances are classified as follows:

Input assembly Instances (Input Connection Points)

Input Assembly Instances produce data on the network.

I/O direction for the EtherNet/IP Adapter: T→O (the adapter sends data to the scanner via this assembly instance)

Output assembly Instances (Output Connection Points)

Output assembly Instances consume data from the network.

I/O direction for the EtherNet/IP Adapter: O→T (the adapter receives data from the scanner via this assembly instance)

Configuration Assembly Instances

An assembly instances carrying configuration data instead of IO data. This allows transferring configuration data upon connection establishment.

Device profiles often contain fix assembly instances for the kind of device they model. The numbering of instances depends on the kind of usage:

If you implement a predefined CIP device profile for your device, then the assembly instances shall use the assembly instance number ranges for open profiles. These are 1…0x63, 0xC8…0x2FF and 0x500…0xFFFFF (also see *Table 98: Assembly Instance Number Ranges*).

If you implement vendor-specific extensions to a CIP device profile or a device profile of your own, then the applicable assembly instance number ranges for vendor-specific profiles shall be used. These are 0x64…0xC7 and 0x300…0x4FF.

2.4.6.2 Restrictions regarding the EtherNetInterface (NDIS) channel

Regarding the provided EtherNetInterface (NDIS) DPM channel, the following restriction applies on Implicit Messaging: Implicit messaging of the EtherNet/IP stack will not be forwarded to the provided EtherNetInterface-Channel, even if data transfer is multicast-based and an application has registered for the actual multicast group address being used.

2.4.7 Explicit Messaging

Explicit messaging is used for point to point messaging that typically takes place only once (or at least not very frequently). Explicit messaging is typically used for non-real data such as:

- Diagnostic
- Information
- Configuration
- Request of data for a single time

In most cases, the real-time requirements for explicit messages are less severe as those for implicit messaging.

Explicit messaging works in unconnected and connected mode. It is used for acyclic data transmission of data having to be transferred only once such as configuration and diagnostic data. Communication takes place in point-to-point mode.

The messaging uses the request/response mechanism based on the client-server model. The support of explicit messaging is mandatory for every CIP device.

2.5 CIP Data Types

The following *Table 15: CIP Data Types* describes common data types that are used in CIP.

Keyword	Description	Number of Bytes
BOOL	Boolean	1 (1-bit encoded into 1-byte)
BYTE	Bit string - 8 bits	1
USINT	Unsigned Short Integer	1
SINT	Short Integer	1
WORD	Bit string – 16 bits	2
UINT	Unsigned Integer	2
INT	Integer	2
DWORD	Bit string – 32 bits	4
UDINT	Unsigned Double Integer	4
DINT	Double Integer	4
SHORT_STRING	character string (1 byte per character, 1 byte length indicator)	1 + n (first byte indicates length)
STRING	character string (1 byte per character, 2 bytes length indicator)	2 + n (first byte indicates length)
STRING2	character string (2 byte per character, 2 bytes length indicator)	2 + n (first byte indicates length)

Table 15: CIP Data Types

2.6 Object Library

The CIP Family of Protocols contains a large collection of commonly defined objects. The overall set of object classes can be subdivided into three types:

- General-use
- Application-specific
- Network-specific

Objects defined in Volume 1 of the CIP Networks Library are available for use on all network adaptations of CIP. Some of these objects may require specific changes or limitations when implemented on some of the network adaptations. These exceptions are noted in the network specific volume.

The following are objects for general use:

- Assembly
- Acknowledge Handler
- Connection
- Connection Configuration
- Connection Manager
- File
- Identity
- Message Router
- Parameter
- Parameter Group
- Port
- Register
- Selection

The following group of objects is application-specific:

- AC/DC Drive
- Analog Group
- Analog Input Group
- Analog Output Group
- Analog Input Point
- Analog Output Point
- Block Sequencer
- Command Block
- Control Supervisor
- Discrete Group
- Discrete Input Group
- Discrete Output Group
- Discrete Input Point
- Discrete Output Point
- Group
- Motor Data
- Overload
- Position Controller
- Position Controller Supervisor
- Position Sensor
- Presence Sensing
- S-Analog Actor
- S-Analog Sensor
- S-Device Supervisor
- S-Gas Calibration
- S-Partial Pressure
- S-Single Stage Controller
- Safety Supervisor
- Safety Validation
- Soft start Starter
- Trip Point

The last group of objects is network-specific:

- ControlNet
- ControlNet Keeper
- ControlNet Scheduling
- DeviceNet
- Ethernet Link
- TCP/IP Interface

The general-use objects can be found in many different devices, while the application-specific objects are typically found only in devices hosting such applications. New objects are added on an ongoing basis.

Although this looks like a large number of object types, typical devices implement only a subset of these objects. Figure 7 shows the object model of such a typical device.

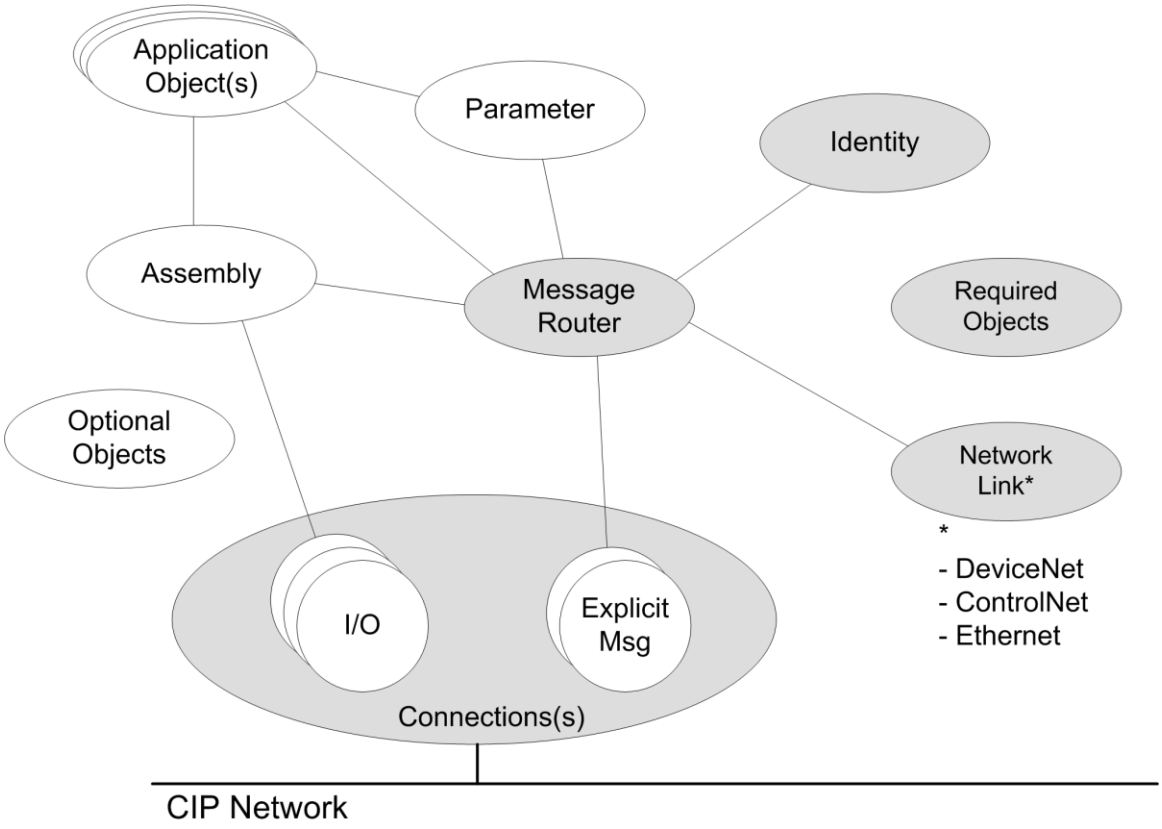

Figure 7: Typical Device Object Model

The objects required in a typical device are:

- Either a Connection Object or a Connection Manager Object
- An Identity Object
- One or several network-specific link objects (EtherNet/IP requires the TCP/IP Interface Object and the Ethernet Link Object)
- A Message Router Object (at least its function)

Further objects are added according to the functionality of the device. This enables scalability for each implementation so that small devices, such as proximity sensors on DeviceNet, are not burdened with unnecessary overhead. Developers typically use publicly defined objects (see above list), but can also create their own objects in the vendor-specific areas, e.g. Class ID 100 -

199. However, they are strongly encouraged to work with the (Joint) Special Interest Groups (JSIGs/SIGs) of ODVA and ControlNet International to create common definitions for additional objects instead of inventing private ones.

Out of the general use objects, several will be described in more detail:

2.7 CIP Device Profiles

It would be possible to design products using only the definitions of communication networks and objects, but this could easily result in similar products having quite different data structures and behavior. To overcome this situation and to make the application of CIP devices much easier, devices of similar functionality have been grouped into Device Types with associated profiles. Such a CIP profile contains the full description of the object structure and behavior. The following Device Types and associated profiles are defined in Volume 1 (see [1]) (profile numbers are bracketed):

- AC Drives Device (0x02)
- CIP Modbus Device (0x28)
- CIP Modbus Translator (0x29)
- CIP Motion Drive (0x25)
- Communications Adapter (0x0C)
- CompoNet Repeater (0x26)
- Contactor (0x15)
- ControlNet Physical Layer Component (0x32)
- ControlNet Programmable Logic Controller (0x0E)
- DC Drives (0x13)
- DC Power Generator (0x1F)
- Encoder (0x22)
- Fluid Flow Controller (0x24)
- General Purpose Discrete I/O (0x07)
- Generic Device (0x2B)
- Human Machine Interface (0x18)
- Inductive Proximity Switch (0x05)
- Limit Switch (0x04)
- Managed Switch (0x2C)
- Mass Flow Controller (0x1A)
- Mass Flow Controller, Enhanced (0x27)
- Motor Overload Device (0x03)
- Motor Starter (0x16)
- Photoelectric Sensor (0x06)
- Pneumatic Valve (0x1B)
- Position Controller (0x10)
- Process Control Valve (0x1D)
- Residual Gas Analyzer (0x1E)
- Resolver (0x09)
- RF Power Generator (0x20)
- Safety Analog I/O Device (0x2A)
- Safety Discrete I/O (0x23)
- Soft start Starter (0x17)
- Turbo molecular Vacuum Pump (0x21)
- Vacuum/Pressure Gauge (0x1C)

Device developers must use a profile. Any device that does not fall into the scope of one of the specialized profiles must use the Generic Device profile or a vendor-specific profile. What profile is used and which parts of it are implemented must be described in the user's device documentation.

Every profile consists of a set of objects - some required, some optional - and a behavior associated with that particular type of device. Most profiles also define one or several I/O data formats (Assemblies) that define the meaning of the individual bits and bytes of the I/O data. In addition to the publicly-defined object set and I/O data Assemblies, vendors can add objects and Assemblies of their own if their devices provide additional functionality. In addition, vendors can create profiles within the vendor-specific profile range. They are then free to define whatever behavior and objects are required for their device as long as they adhere to some general rules for profiles. Whenever additional functionality is used by multiple vendors, ODVA and ControlNet International encourage coordinating these new features through discussion in the Joint Special Interest Groups (JSIGs), which can then create new profiles and additions to existing profiles for everybody's use and for the benefit of the device users.

All open (ODVA/CI defined) profiles carry numbers in the 0x00 through 0x63 or 0x0100 through 0x02FF ranges, while vendor-specific profiles carry numbers in the 0x64 through 0xC7 or 0x0300 through 0x02FF ranges. All other profile numbers are reserved by CIP.

2.8 EDS (Electronic Data Sheet)

An EDS is a simple ASCII text file that can be generated on any ASCII editor. Since the CIP Specification lays down a set of rules for the overall design and syntax of an EDS which makes configuration of devices much easier. Specialized EDS editing tools, such as ODVA's EZ-EDS, can simplify the creation of EDS files. The main purpose of the EDS is to give information on several aspects of the device's capabilities, the most important ones being the I/O Connections it supports and what parameters for display or configuration exist within the device. It is highly recommended that an EDS describe all supported I/O Connections, as this makes the application of a device much easier. When it comes to parameters, it is up to the developer to decide which items to make accessible to the user.

Let's look at some details of the EDS. First, an EDS is structured into sections, each of which starts with a section name in square brackets []. The first two sections are mandatory for all EDSs.

[File]: Describes the contents and revision of the file.

- **[Device]:** Is equivalent to the Identity Object information and is used to match an EDS to a device.
- **[Device Classification]:** Describes what network the device can be connected to. This section is optional for DeviceNet, required for ControlNet and EtherNet/IP.
- **[Params]:** Identifies all configuration parameters in the device.
- **[Assembly]:** Describes the structure of data items.
- **[Connection Manager]:** Describes connections supported by the device. Typically used in ControlNet and EtherNet/IP.
- **[Capacity]:** Specifies the communication capacity of EtherNet/IP and ControlNet devices.

A tool with a collection of EDSs will first use the [Device] section to try to match an EDS with each device it finds on a network. Once this is done and a particular device is chosen, the tool can then display device properties and parameters and allows their modification (if necessary). A tool may also display what I/O Connections a device may allow and which of these are already in use. EDS-based tools are mainly used for slave or adapter devices, as scanner devices typically are too complex to be configured through EDSs. For those devices, the EDS is used primarily to identify the device, then guide the tool to call a matching configuration applet.

A particular strength of the EDS approach lies in the methodology of parameter configuration. A configuration tool typically takes all of the information supplied by an EDS and displays it in a user-friendly manner. In many cases, this enables the user to configure a device without needing a detailed manual, as the tool presentation of the parameter information, together with help texts, enables decisions making for a complete device configuration (provided, of course, the developer has supplied all required information).

3 Available CIP Classes in the Hilscher EtherNet/IP Stack

The following subsections describe all default CIP object classes that are available within the Hilscher EtherNet/IP stack.

Figure 8 gives an overview about the available CIP objects and their instances assuming a default configuration (assembly instances 100 and 101).

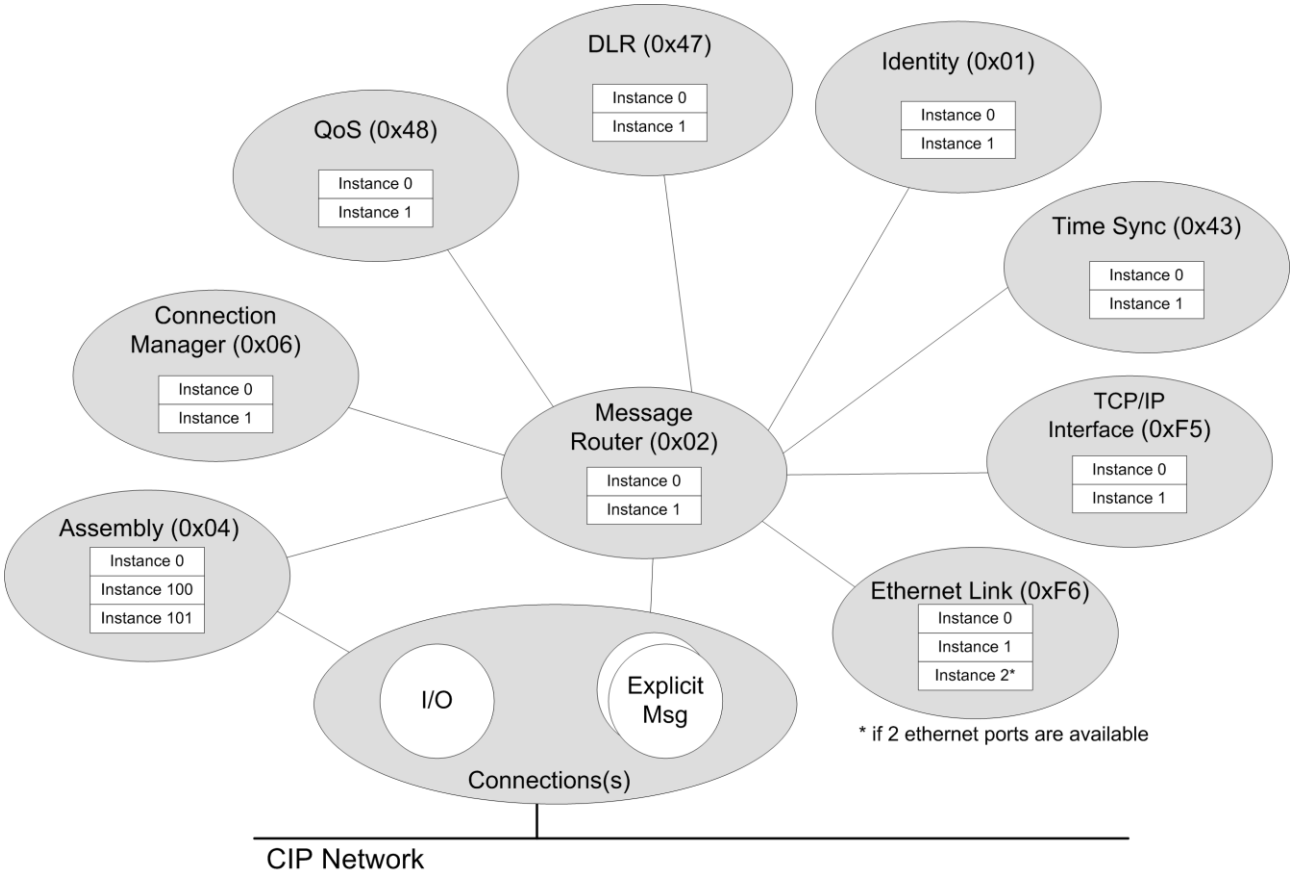

Figure 8: Default Hilscher Device Object Model

3.1 Introduction

Every CIP class is described using two tables. One table describes the class attributes and one describes the instance attributes.

A Class Attribute is an attribute whose scope is that of the class as a whole, rather than any one particular instance. Therefore, the list of Class Attributes is different than the list of Instance Attributes. CIP defines the Instance ID value zero (0) to designate the Class level versus a specific Instance within the Class. Class Attributes are defined using the following terms:

Class Attributes (Instance 0)

Attr ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
1	2	3	4	5	6	7	8
1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request							

Table 16: Class Attributes

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. The **Access Rule From Network** specifies how a requestor can access an attribute from the EtherNet/IP network. The definitions for access rules are:
   - Settable (Set) - The attribute can be accessed by at least on of the set services (Set_Attribute_Single/ Set_Attribute_All).
   - Gettable (Get) - The attribute can be accessed by at least one of the get services (Get_Attribute_Single/ Get_Attribute_All).
3. The **Access Rule From Host** specifies how the Host Application (running on the netX or on a host processor) can access an attribute using the packet API of the stack (see description of EIP_OBJECT_CIP_SERVICE_REQ/CNF – CIP Service Request).

   The definitions for access rules are:
   - Settable (Set) - The attribute can be accessed by at least one of the set services (Set_Attribute_Single/ Set_Attribute_All).
   - Gettable (Get) - The attribute can be accessed by at least one of the get services (Get_Attribute_Single/ Get_Attribute_All).
4. **NV** indicates whether an attribute values maintained through power cycles. This column is used in object definitions where non-volatile storage of attribute values is required. An entry of ‘NV’ indicates value shall be saved, ‘V’ means not saved.
5. **Name** refers to the attribute.
6. **Data Type** – See section *CIP Data Types* on page 36.
7. **Description of Attribute** provides general information about the attribute.
8. **Semantics of values** specifies the meaning of the value of the attribute.

An Instance Attribute is an attribute that is unique to an object instance and not shared by the object class. Instance Attributes are defined in the same terms as Class Attributes.

Instance Attributes (Instance 1, 2, …, n)

Att ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
1	2	3	4	5	6	7	8
1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request							

Table 17: Instance Attributes

3.2 Identity Object (Class Code: 0x01)

The Identity Object provides identification and general information about the device. The first and only instance identifies the whole device. It is used for electronic keying and by applications wishing to determine what devices are on the network.

3.2.1 Class Attributes

Attr ID	Access Rule		Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]				
1	Get	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is 2.
2	Get	Get	Max. Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
6	Get	Get	Maximum ID Number Class Attributes	UINT	The attribute ID number of the last class attribute of the class definition implemented in the device.	
7	Get	Get	Maximum ID Number Instance Attributes	UINT	The attribute ID number of the last instance attribute of the class definition implemented in the device.	
1) Related to API `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.						

Table 18: Identity Object - Class Attributes

3.2.2 Instance Attributes

Attr ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
1	Get	Get	NV	Vendor ID	UINT	Vendor Identification	
2	Get	Get	NV	Device Type	UINT	Indication of general type of product	
3	Get	Get	NV	Product Code	UINT	Identification of a particular product of an individual vendor	
4	Get	Get	NV	Revision	STRUCT of		
				Major Revision	USINT		
				Minor Revision	USINT		
5	Get	Get, Set [2)]	V	Status	WORD	Summary status of device	
6	Get	Get	NV	Serial Number	UDINT	Serial number of device	
7	Get	Get	NV	Product Name	SHORT _STRING	Human readable identification	
8	Get	Get Set [3)]	V	State	USINT	Present state of the device	0 = Non-existent 1 = Device Self Testing 2 = Standby 3 = Operational 4 = Major Recoverable Fault 5 = Major Unrecoverable Fault
9	Get	Get	NV	Conf. Consist. Value	UINT	Configuration Consistency Value	

1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.

2) Set service is possible, but only bit 0 (Owned), 2 (Configured) and bits 8 to 15 are settable. Do not overwrite other bits in this attribute. The host has to read, modify and write the attribute data. For more information about this attribute have a look at the definition of the identity object (Cip Volume 1).

3) Setting of the state attribute is not possible by default, but can be activated using the EIP_OBJECT_SET_PARAMETER_REQ (0x1AF2) (see flag EIP_OBJECT_PRM_APPLICATION_CONTROLS_IDENTITY_STATE_ATTRIBUTE in section 6.2.14).

Table 19: Identity Object - Instance Attributes

3.2.3 Supported Services

- Get_Attribute_Single (Service Code: 0x0E)
- Set_Attribute_Single (Service Code: 0x10)
- GetAttributeAll (Service Code: 0x01)
- Reset (Service Code: 0x05)
  - Reset Type 0 is supported by default
  - Additionally, the support of reset type 1 can be activated using API command `EIP_OBJECT_SET_PARAMETER_REQ/CNF` – Set Parameter

3.3 Message Router Object (Class Code: 0x02)

The Message Router Object provides a messaging connection point through which a client may address a service to any object class or instance residing in the physical device.

3.3.1 Supported Services

Since the message router (in the Hilscher Implementation) does not have any class or instance attributes, there are no services supported.

3.4 Assembly Object (Class Code: 0x04)

The Assembly Object binds attributes of multiple objects, which allows data to or from each object to be sent or received over a single connection. Assembly Objects can be used to bind produced data or consumed data.

3.4.1 Class Attributes

Attr ID	Access Rule		Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]				
1	Get	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is 2 (02).
1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.						

Table 20: Assembly Object - Class Attributes

3.4.2 Instance Attributes

Attr ID	Access Rule		Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]				
3	Get, Set [2)]	Get, Set [3)]	Data	ARRAY of BYTE		
4	Get	Get	Size	UINT	Number of bytes in Attribute 3	
1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request. 2) Set service only available for consuming assemblies that are not part of an active implicit connection 3) Set service is only available for configuration assembly instances (registered with the flag EIP_AS_FLAG_CONFIG, see Table 100)						

Table 21: Assembly Object - Instance Attributes

3.4.3 Supported Services

- Get_Attribute_Single (Service Code: 0x0E)
- Set_Attribute_Single (Service Code: 0x10)

3.5 Connection Manager Object (Class Code: 0x06)

The Connection Manager Class allocates and manages the internal resources associated with both I/ O and Explicit Messaging Connections.

3.5.1 Class Attributes

Attr ID	Access Rule		Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]				
1	Get	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is one (01).
2	Get	Get	Max. Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.						

Table 22: Assembly Object - Class Attributes

3.5.2 Supported Services

- Get_Attribute_Single (Service Code: 0x0E)
- Forward_Open (Service Code: 0x54)
- Forward_Close (Service Code: 0x4E)

3.6 TCP/IP Interface Object (Class Code: 0xF5)

The TCP/IP Interface Object provides the mechanism to configure a device's TCP/IP network interface. Examples of configurable items include 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.

3.6.1 Class Attributes

Attr ID	Access Rule		Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]				
1	Get	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is 4.
2	Get	Get	Max. Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.						

Table 23: TCP/IP Interface - Class Attributes

3.6.2 Instance Attributes

Attr ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
1	Get	Get, Set[6)]	V	Status	DWORD	Interface status	See section 3.6.2.1
2	Get	Get, Set	NV	Configuration Capability	DWORD	Interface capability flags	See section 3.6.2.2
3[5)]	Get, Set	Get	NV	Configuration Control	DWORD	Interface control flags	See section 3.6.2.3
4	Get	Get	NV	Physical Link Object	STRUCT of	Path to physical link object	See section 3.6.2.4
				Path size	UINT	Size of Path	Number of 16 bit words in Path
				Path	Padded EPATH	Logical segments identifying the physical link object	The path is restricted to one logical class segment and one logical instance segment. The maximum size is 12 bytes.
5[5)]	Get, Set[4)]	Get, Set [2)]	NV	Interface Configuration	STRUCT of		See section 3.6.2.5

Attr ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
				IP Address	UDINT	The device's IP address.	Value of 0 indicates no IP address has been configured. Otherwise, the IP address shall be set to a valid Class A, B, or C address and shall not be set to the loopback address (127.0.0.1).
				Network Mask	UDINT	The device's network mask	Value of 0 indicates no network mask address has been configured.
				Gateway Address	UDINT	Default gateway address	Value of 0 indicates no IP address has been configured. Otherwise, the IP address shall be set to a valid Class A, B, or C address and shall not be set to the loopback address (127.0.0.1).
				Name Server	UDINT	Primary name server	Value of 0 indicates no name server address has been configured. Otherwise, the name server address shall be set to a valid Class A, B, or C address.
				Name Server 2	UDINT	Secondary name server	Value of 0 indicates no secondary name server address has been configured. Otherwise, the name server address shall be set to a valid Class A, B, or C address.
				Domain Name	STRING	Default domain name	ASCII characters. Maximum length is 48 characters. Shall be padded to an even number of characters (pad not included in length). A length of 0 shall indicate no Domain Name has been configured.

Attr ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
6[5)]	Get, Set	Get, Set	NV	Host Name	STRING	The Host Name attribute contains the device's host name, which can be used for informational purposes.	ASCII characters. Maximum length is 64 characters. Shall be padded to an even number of characters (pad not included in length). A length of 0 shall indicate no Host Name has been configured.
7[3)]	Get	Get, Set		Safety Network Number	6 octets	See CIP Safety Specfication, Volume 5, Chapter 3	
8[3) 5)]	Get, Set	Get, Set	NV	TTL Value	USINT	TTL value for EtherNet/IP multicast packets	Time-to-Live value for IP multicast packets. Default value is 1. Minimum is 1; maximum is 255 See section 3.6.2.6
9[3) 5)]	Get, Set	Get, Set	NV	Mcast Config	STRUCT of	IP multicast address configuration	See section 3.6.2.7
				Alloc Control	USINT	Multicast address allocation control word. Determines how addresses are allocated.	See section 3.6.2.7 for details. Determines whether multicast addresses are generated via algorithm or are explicitly set.
				Reserved	USINT	Reserved for future use	Shall be 0.
				Num Mcast	UINT	Number of IP Multicast addresses to allocate for EtherNet/IP	The number of IP multicast addresses allocated, starting at "Mcast Start Addr". Maximum value is 128 (Hilscher specific).
				Mcast Start Addr	UDINT	Starting multicast address from which to begin allocation.	IP multicast address (Class D). A block of "Num Mcast" addresses is allocated starting with this address.
10[5)]	Get, Set	Get, Set	NV	SelectAcd	BOOL	Activates the use of ACD	Enable ACD (1, default), Disable ACD (0). See section 3.6.2.8

Attr ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
11[5)]	Get, Set	Get, Set	NV when Configuration Method is 0. V when obtained via BOOTP or DHCP	LastConflictDetected	STRUCT of:	Structure containing information related to the last conflict detected	ACD Diagnostic Parameters. See section 3.6.2.9
				AcdActivity	USINT	State of ACD activity when last conflict detected	ACD activity Default = 0
				Remote MAC	Array of 6 USINT	MAC address of remote node from the ARP PDU in which a conflict was detected	MAC Entry from Ethernet Frame Header Default = 0
				ArpPdu	ARRAY of 28 USINT	Copy of the raw ARP PDU in which a conflict was detected.	ARP PDU Default = 0
12 [3) 5)]	Get, Set	Get, Set	NV	EtherNet/IP Quick Connect	BOOL	Enable/Disable of Quick Connect feature	0 = Disable (default) 1 = Enable See section 9.4 “Quick Connect”
13[5)]	Get,Set	Get, Set	NV	Encapsulation Inactivity Timeout	UINT	Number of seconds of inactivity before TCP connection is closed	0 = Disable 1-3600 = timeout in seconds Default = 120 See section 3.6.2.10

1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.

2) All entries are settable except: IP, Gateway, and subnet mask. These must be set by either of the following API commands:
`EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ/CNF` – Configure the Device with Configuration Parameter `TCPIP_IP_CMD_SET_CONFIG_REQ/CNF (0x00000200)` - TcpIp Stack (see reference [3])

3) Attribute is not available in the EtherNet/IP stack per default. If the attribute shall be activated use API command `EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ/CNF` – CIP Object Attribute Activate Request

4) This attribute is only settable from the network if attribute 3 of this object (configuration control) has value 0 (STATIC). Otherwise, the set request will be rejected with error code 0x0C (“Object State Conflict”)

5) If the attribute value is changed, the host application is notified via the indication `EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES` – CIP Object Change Indication (see section 6.2.18 on page 201)

6) The host application is allowed to set this attribute, but only the lower 4bits (Interface Configuration Status). All other bits are managed by the EtherNet/IP protocol stack itself.

Table 24: TCP/IP Interface - Instance Attributes

3.6.2.1 Status

The Status attribute is a bitmap that shall indicate the status of the TCP/IP network interface.

Bit(s)	Name	Definition	
0-3	Interface Configuration Status	Indicates the status of the Interface Configuration attribute.	0 = The Interface Configuration attribute has not been configured. 1 = The Interface Configuration attribute contains configuration obtained from BOOTP, DHCP or nonvolatile storage. 2 = The IP address member of the Interface Configuration attribute contains configuration, obtained from hardware settings (e.g.: pushwheel, thumbwheel, etc.) 3-15 = Reserved for future use.
4	Mcast Pending	Indicates a pending configuration change in the TTL Value and/or Mcast Config attributes. This bit shall be set when either the TTL Value or Mcast Config attribute is set, and shall be cleared the next time the device starts.	
5	Interface Configuration Pending	Indicates a pending configuration change in the Interface Configuration attribute. This bit shall be 1 (TRUE) when Interface Configuration attribute are set and the device requires a reset in order for the configuration change to take effect (as indicated in the Configuration Capability attribute). The intent of the Interface Config Pending bit is to allow client software to detect that a device's IP configuration has changed, but will not take effect until the device is reset.	
6	AcdStatus	Indicates when an IP address conflict has been detected by ACD. This bit shall default to 0 (FALSE) on startup. If ACD is supported and enabled, then this bit shall be set to 1 (TRUE) any time an address conflict is detected as defined by the [ConflictDetected] transitions in Figure F-1.1 ACD Behavior.	
7	Acd Fault	Indicates when an IP address conflict has been detected by ACD or the defense failed, and that the current Interface Configuration cannot be used due to this conflict. This bit SHALL be 1 (TRUE) if an address conflict has been detected and this interface is currently in the Notification & FaultAction or AcquireNewIpv4Parameters ACD state as defined in Appendix F, and SHALL be 0 (FALSE) otherwise. Notice that when this bit is set, then this CIP port will not be usable. However, for devices with multiple ports, this bit provides a way of determining if the port has an ACD fault and thus cannot be used.	
8-31	Reserved	Reserved for future use. Is set to zero.	

Table 25: TCP/IP Interface - Instance Attribute 1 - Status

3.6.2.2 Configuration Capability

The Configuration Capability attribute is a bitmap that indicates the device's support for optional network configuration capability. Devices are not required to support any one particular item, however must support at least one method of obtaining an initial IP address.

Bit(s)	Name	Definition
0	BOOTP Client	1 (TRUE) shall indicate the device is capable of obtaining its network configuration via BOOTP.
1	DNS Client	1 (TRUE) shall indicate the device is capable of resolving host names by querying a DNS server.
2	DHCP Client	1 (TRUE) shall indicate the device is capable of obtaining its network configuration via DHCP.
3	DHCP-DNS Update (not supported)	Shall be 0, behavior to be defined in a future specification edition.
4	Configuration Settable	1 (TRUE) shall indicate the Interface Configuration attribute is settable.
5	Hardware Configurable	1 (TRUE) shall indicate the IP Address member of the Interface Configuration attribute can be obtained from hardware settings (e.g., pushwheel, thumbwheel, etc.). If this bit is FALSE the Status Instance Attribute (1), Interface Configuration Status field value shall never be 2 (The Interface Configuration attribute contains valid configuration, obtained from hardware settings). This bit can be configured by the host application using the packet `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.
6	Interface Configuration Change Requires Reset	1 (TRUE) shall indicate that the device requires a restart in order for a change to the Interface Configuration attribute to take effect. If this bit is FALSE a change in the Interface Configuration attribute will take effect immediately.
7	AcdCapable	(1) TRUE shall indicate that the device is capable of ACD.
8-31	Reserved	Reserved for future use. Is set to zero.

Table 26: TCP/IP Interface - Instance Attribute 2 – Configuration Capability

3.6.2.3 Configuration Control

The Configuration Control attribute is a bitmap used to control network configuration options.

Bit(s)	Name	Definition	
0-3	Configuration Method	Determines how the device shall obtain its IP-related configuration	0 = The device shall use statically-assigned IP configuration values. 1 = The device shall obtain its interface configuration values via BOOTP. 2 = The device shall obtain its interface configuration values via DHCP. 3-15 = Reserved for future use.
4	DNS Enable (not supported)	If 1 (TRUE), the device shall resolve host names by querying a DNS server.	
5-31	Reserved	Reserved for future use. Is set to zero.	

Table 27: TCP/IP Interface - Instance Attribute 3 – Configuration Control

Configuration Method:

The Configuration Method determines how a device shall obtain its IP-related configuration:

- If the Configuration Method is 0, the device shall use statically-assigned IP configuration contained in the Interface Configuration attribute (or assigned via non-CIP methods, as noted below).
- If the Configuration Method is 1, the device shall obtain its IP configuration via BOOTP.
- If the Configuration Method is 2, the device shall obtain its IP configuration via DHCP.
- Devices that optionally provide hardware means (e.g., rotary switch) to configure IP addressing behavior shall set the Configuration Method to reflect the configuration set via hardware: 0 if a static IP address has been configured, 1 if BOOTP has been configured, 2 if DHCP has been configured.

If a device has been configured to obtain its configuration via BOOTP or DHCP it will continue sending requests until a response from the server is received. Devices that elect to use default IP configuration in the event of no response from the server shall continue issuing requests until a response is received, or until the Configuration Method is changed to static.

Once the device receives a response from the server it stops sending the BOOTP/DHCP client requests (DHCP clients shall follow the lease renewal behavior per the RFC).

Setting the Configuration Method to 0 (static address) causes the Interface Configuration to be saved to NV storage.

Note: Usually the host application of the EtherNet/IP stack is responsible for storing the new Interface Configuration values. See also section 6.2.18 *EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES – CIP Object Change Indication*.

Setting the Configuration Method to 1 (BOOTP) or 2 (DHCP) causes the device right away to start the BOOTP / DHCP client to obtain new IP address configuration. The device does not require a reset in order to start the BOOTP / DHCP client.

Note: This behavior must be implemented by the host application.

3.6.2.4 Physical Link

This attribute identifies the object associated with the underlying physical communications interface (e.g., an 802.3 interface). There are two components to the attribute: a Path Size (in UINTs) and a Path. The Path shall contain a Logical Segment, type Class, and a Logical Segment, type Instance that identifies the physical link object. The maximum Path Size is 6 (assuming a 32 bit logical segment for each of the class and instance).

The physical link object itself typically maintains link-specific counters as well as any link specific configuration attributes. If the CIP port associated with the TCP/IP Interface Object has an Ethernet physical layer, this attribute shall point to an instance of the Ethernet Link Object (class code = 0xF6). When there are multiple physical interfaces that correspond to the TCP/IP interface, this attribute shall either contain a Path Size of 0, or shall contain a path to the object representing an internal communications interface (often used in the case of an embedded switch).

For example, the path could be as follows:

Path	Meaning
[20][F6][24][01]	[20] = 8 bit class segment type; [F6] = Ethernet Link Object class; [24] = 8 bit instance segment type; [01] = instance 1.

Table 28: TCP/IP Interface - Instance Attribute 4 – Physical Link

3.6.2.5 Interface Configuration

The Interface Configuration attribute contains the configuration parameters required for a device to operate as a TCP/IP node. The contents of the Interface Configuration attribute shall depend upon how the device has been configured to obtain its IP parameters:

- If configured to use a static IP address (Configuration Method value is 0), the Interface Configuration values shall be those which have been statically assigned and stored in NV storage.
- If configured to use BOOTP or DHCP (Configuration Method value is 1 or 2), the Interface Configuration values shall contain the configuration obtained from the BOOTP or DHCP server. The Interface Configuration attribute shall be 0 until the BOOTP/DHCP reply is received.
- Some devices optionally provide additional, non-CIP mechanisms for setting IP-related configuration (e.g., a web server interface, rotary switch for configuring IP address, etc.). When such a mechanism is used, the Interface Configuration attribute shall reflect the IP configuration values in use.

Name	Meaning
IP Address	The device's IP address.
Network mask	The device's network mask. The network mask is used when the IP network has been partitioned into subnets. The network mask is used to determine whether an IP address is located on another subnet.
Gateway address	The IP address of the device's default gateway. When a destination IP address is on a different subnet, packets are forwarded to the default gateway for routing to the destination subnet.
Name server	The IP address of the primary name server. The name server is used to resolve host names. For example, that might be contained in a CIP connection path. **Note:** The name server functionality is not supported by the Hilscher Ethernet/IP stack
Name server 2	The IP address of the secondary name server. The secondary name server is used when the primary name server is not available, or is unable to resolve a host name. **Note:** The name server functionality is not supported by the Hilscher Ethernet/IP stack
Domain name	The default domain name. The default domain name is used when resolving host names that are not fully qualified. For example, if the default domain name is "odva.org", and the device needs to resolve a host name of "plc", then the device will attempt to resolve the host name as "plc.odva.org". **Note:** The domain name functionality is not supported by the Hilscher Ethernet/IP stack

Table 29: TCP/IP Interface - Instance Attribute 5 – Interface Control

Set Behavior

In order to prevent incomplete or incompatible configuration, the parameters making up the Interface Configuration attribute cannot be set individually. To modify the Interface Configuration attribute, client software should first Get the Interface Configuration attribute, change the desired parameters, and then Set the attribute.

An attempt to set any of the parameters of the Interface Configuration attribute to invalid values will result in an error response with status code 0x09 'Invalid Attribute Value' to be returned. In this scenario, all of the parameters of the Interface Configuration attribute retain the values that existed prior to the invocation of the set service.

When the value of the Configuration Method (Configuration Control attribute) is 0, the set attribute service will store the new Interface Configuration values in non-volatile memory.

Note: Usually the host application of the EtherNet/IP stack is responsible for storing the new Interface Configuration values. See also section 6.2.18 "`EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES` – CIP Object Change Indication". Although the Name Server, Name Server 2 and Domain Name parameters are not supported by the Hilscher EtherNet/IP stack, they need to be stored along with the other parameters.

Changing the IP setting causes the device right away to apply the new address configuration. The device does not require a reset.

Note: This behavior must be implemented by the host application.

3.6.2.6 TTL Value

TTL Value is value the device shall use for the IP header Time-to-Live field when sending EtherNet/IP packets via IP multicast. By default, TTL Value shall be 1. The maximum value for TTL is 255.

When set, the TTL Value attribute shall be saved in non-volatile memory.

Note: Usually the host application of the EtherNet/IP stack is responsible for storing the new Interface Configuration values. See also section 6.2.18 "EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES – CIP Object Change Indication".

If the TTL Value is set, the Hilscher EtherNet/IP Stack automatically sets the Mcast Pending bit in the Interface Status attribute. This indicates that there is a pending configuration. The device then needs to be reset in order for the new configuration to be applied. The Mcast Pending bit will be cleared automatically the next time the device starts.

When a new TTL Value is pending, Get_Attribute_Single or Get_Attributes_All requests will return the pending value.

Note: Users should exercise caution when setting the TTL Value greater than 1, to prevent unwanted multicast traffic from propagating through the network.

3.6.2.7 Mcast Config

The Mcast Config attribute contains the configuration of the device's IP multicast addresses to be used for EtherNet/IP multicast packets. There are three elements to the Mcast Config structure: **Alloc Control, Num Mcast, and Mcast Start Addr**.

Alloc Control determines how the device shall allocate IP multicast addresses (e.g., whether by algorithm, whether they are explicitly set, etc.). Table 30 shows the details for Alloc Control.

Value	Definition
0	Multicast addresses shall be generated using the default allocation algorithm (automatically done by the Hilscher EtherNet/IP stack). When this value is specified on a set-attribute or set-attributes-all, the values of Num Mcast and Mcast Start Addr in the set-attribute request must be 0.
1	Multicast addresses shall be allocated according to the values specified in Num Mcast and Mcast Start Addr.
2	Reserved

Table 30: TCP/IP Interface - Instance Attribute 9 – Mcast Config (Alloc Control Values)

Num Mcast is the number of IP multicast addresses allocated. The maximum number of multicast addresses is 128 (Hilscher specific).

Mcast Start Addr is the starting multicast address from which Num Mcast addresses are allocated.

When set, the Mcast Config attribute must be saved in non-volatile memory.

Note: Usually the host application of the EtherNet/IP stack is responsible for storing the new Interface Configuration values. See also section 6.2.18 "EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES – CIP Object Change Indication".

If the Mcast Config is set, the Hilscher EtherNet/IP Stack automatically sets the Mcast Pending bit in the Interface Status attribute. This indicates that there is a pending configuration.

When a new Mcast Config value is pending, Get_Attribute_Single or Get_Attributes_All requests will return the pending value. The Mcast Pending bit will be cleared the next time the device starts.

When the multicast addresses are generated using the default algorithm, Num Mcast and Mcast Start Addr will report the values generated by the algorithm.

3.6.2.8 Select ACD

SelectAcd is an attribute used to Enable/Disable ACD.

If SelectAcd is 0 then ACD is disabled. If SelectAcd is 1 then ACD is enabled (default value is 1).

When the value of SelectAcd is changed by a Set_Attribute service, the new value of SelectAcd will not be applied until the device executes a restart.

3.6.2.9 Last Conflict Detected

The LastConflictDetected attribute is a diagnostic attribute presenting information about the ACD state when the last IP Address conflict was detected. This attribute will be updated by the device whenever an incoming ARP packet is received that represents a conflict with the device's IP address as described in IETF RFC 5227.

To reset this attribute the Set_Attribute_Single service must be invoked with an attribute value of all 0. If the Set_Attribute_Single service is received from an EtherNet/IP Scanner, values other than 0 will result in an error response: status code 0x09, Invalid Attribute Value. If this attribute is set from the host application e.g. using Set_Attribute_Single service with command EIP_OBJECT_CIP_SERVICE_REQ, any data is valid.

AcdActivity – The ACD contains the state of the ACD algorithm when the last IP address conflict was detected. The ACD activities are defined in the following table.

Value	AcdMode	Description
0	NoConflictDetected (Default)	No conflict has been detected since this attribute was last cleared.
1	ProbeIpv4Address	Last conflict detected during ProbeIpv4Address state.
2	OngoingDetection	Last conflict detected during OngoingDetection state or subsequent DefendWithPolicyB state.
3	SemiActiveProbe	Last conflict detected during SemiActiveProbe state or subsequent DefendWithPolicyB state.

Table 31: TCP/IP Interface - Instance Attribute 11 – Last Conflict Detected (Acd Activity)

RemoteMac - The IEEE 802.3 source MAC address from the header of the received Ethernet packet which was sent by a device reporting a conflict.

ArpPdu – The ARP Response PDU in binary format.

The ArpPdu is a copy of the ARP message that caused the address conflict. It is a raw copy of the ARP message as it appears on the Ethernet network, i.e.: ArpPdu[1] contains the first byte of the ArpPdu received.

Field Size	Field Description	Field Value
2	Hardware Address Type	1 for Ethernet H/W
2	Protocol Address Type	0x800 for IP
1	HADDR LEN	6 for Ethernet h/w
1	PADDR LEN	4 for IP
2	OPERATION	1 for Req or 2 for Rsp
6	SENDER HADDR	Sender’s h/w addr (MAC address)
4	SENDER PADDR	Sender’s proto addr (IP address)
6	TARGET HADDR	Target’s h/w addr (MAC address)
4	TARGET PADDR	Target’s proto addr (IP address)

Table 32: TCP/IP Interface - Instance Attribute 11 – Last Conflict Detected (Arp PDU)

3.6.2.10 Encapsulation Inactivity Timeout

The Encapsulation Inactivity Timeout attribute is used to enable TCP socket cleanup (closing) when the defined number of seconds have elapsed with no Encapsulation activity.

3.6.3 Supported Services

- Get_Attribute_Single (Service Code: 0x0E)
- Set_Attribute_Single (Service Code: 0x10)
- GetAttributeAll (Service Code: 0x01)

3.7 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 2-port switch, instance 1 is assigned Ethernet port 0 and instance 2 is assigned Ethernet port 1.

3.7.1 Class Attributes

Attribute ID	Access Rule		Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]				
1	Get	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is four (04).
2	Get	Get	Max. Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
3	Get	Get	Number of Instances	UINT	Number of object instances currently created at this class level of the device	The number of object instances at this class hierarchy level. This basically relates to the number of ethernet ports the device supports.
1) Related to API `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.						

Table 33: Ethernet Link - Class Attributes

3.7.2 Instance Attributes

Att ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
1	Get	Get	V	Interface Speed	UDINT	Interface speed currently in use	Speed in Mbps (e.g., 0, 10, 100, 1000, etc.)
2	Get	Get	V	Interface Flags	DWORD	Interface status flags	Bit map of interface flags. See section 3.7.2.2
3	Get	Get	NV	Physical Address	ARRAY of 6 USINTs	MAC layer address	See section 3.7.2.3
4	Get	Get	V	Interface Counters	STRUCT of:		See section 3.7.2.4
				In Octets	UDINT	Octets received on the interface	
				IN Ucast Packets	UDINT	Unicast packets received on the interface	
				In NUcast Packets	UDINT	Non-unicast packets received on the interface	
				In Discards	UDINT	Inbound packets received on the interface but discarded	
				In Errors	UDINT	Inbound packets that contain errors (does not include In Discards)	
				In Unknown Protos	UDINT	Inbound packets with unknown protocol	

Att ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
				Out Octets	UDINT	Octets sent on the interface	
				Out Ucast Packets	UDINT	Unicast packets sent on the interface	
				Out NUcast Packets	UDINT	Non-unicast packets sent on the interface	
				Out Discards	UDINT	Outbound packets discarded	
				Out Errors	UDINT	Outbound packets that contain errors	
5	Get	Get	V	Media Counters	STRUCT of:	Media-specific counters	See section 3.7.2.5
				Alignment Errors	UDINT	Frames received that are not an integral number of octets in length	
				FCS Errors	UDINT	Frames received that do not pass the FCS check	
				Single Collisions	UDINT	Successfully transmitted frames which experienced exactly one collision	
				Multiple Collisions	UDINT	Successfully transmitted frames which experienced more than one collision	
				SQE Test Errors	UDINT	Number of times SQE test error message is generated	
				Deferred Transmissions	UDINT	Frames for which first transmission attempt is delayed because the medium is busy	
				Late Collisions	UDINT	Number of times a collision is detected later than 512 bit-times into the transmission of a packet	
				Excessive Collisions	UDINT	Frames for which transmission fails due to excessive collisions	
				MAC Transmit Errors	UDINT	Frames for which transmission fails due to an internal MAC sub layer transmit error	
				Carrier Sense Errors	UDINT	Times that the carrier sense condition was lost or never asserted when attempting to transmit a frame	
				Frame Too Long	UDINT	Frames received that exceed the maximum permitted frame size	
				MAC Receive Errors	UDINT	Frames for which reception on an interface fails due to an internal MAC sub layer receive error	

Att ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
6[2)]	Get, Set	Get	NV	Interface Control	STRUCT of	Configuration for physical interface	See section 3.7.2.6
				Control Bits	WORD	Interface Control Bits	
				Forced Interface Speed	UINT	Speed at which the interface shall be forced to operate	
7	Get	Get	NV[3)]	Interface Type	USINT	Type of interface: twisted pair, fiber, internal, etc	See section 3.7.2.7
8	Get	Get	V	Interface State	USINT	Current state of the interface: operational, disabled, etc	See section 3.7.2.8
9	Get, Set	Get, Set	NV	Admin State	USINT	Administrative state: enable, disable	See section 3.7.2.9
10	Get	Get, Set	NV	Interface Label	SHORT_ STRING	Human readable identification	See section 3.7.2.10
11	Get	Get	NV	Interface Capability	STRUCT of	Indication of capabilities of the interface	See section 0
				Capability Bits	WORD	Interface capabilities, other than speed/duplex	Bit map
				Speed/Duplex Options	STRUCT of	Indicates speed/duplex pairs supported in the Interface Control attribute	
					USINT	Speed/Duplex Array Count	Number of elements
					ARRAY of STRUCT of	Speed/Duplex Array	
					UINT	Interface Speed	Semantics are the same as the Forced Interface Speed in the Interface Control attribute: speed in Mbps
					USINT	Interface Duplex Mode	0=half duplex 1=full duplex 2-255=Reserved

1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.

2) If the attribute value is changed from the network side, the host application is notified via the indication `EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES` – CIP Object Change Indication (see section 6.2.18 on page 201)

3) Although this attribute is of type NV (non-volatile), it does not need to be stored in remanent memory by the application, since there is only one interface type (twisted pair) supported at this time.

Table 34: Ethernet Link - Instance Attributes

3.7.2.1 Interface Speed

The Interface Speed attribute indicates the speed at which the interface is currently running (e.g., 10 Mbps, 100 Mbps) A value of 0 is used to indicate that the speed of the interface is indeterminate. The scale of the attribute is in Mbps, so if the interface is running at 100 Mbps then the value of Interface Speed attribute is 100. The Interface Speed is intended to represent the media bandwidth; the attribute is not doubled if the interface is running in full-duplex mode.

3.7.2.2 Interface Status Flags

The Interface Flags attribute contains status and configuration information about the physical interface and shall be as follows:

Bit(s)	Name	Definition
0	Link Status	Indicates whether or not the IEEE 802.3 communications interface is connected to an active network. 0 indicates an inactive link. 1 indicates an active link.
1	Half/Full Duplex	Indicates the duplex mode currently in use. 0 indicates the interface is running half duplex 1 indicates full duplex. **Note**: If the Link Status flag is 0, then the value of the Half/Full Duplex flag is indeterminate.
2-4	Negotiation Status	Indicates the status of link auto-negotiation 0 = Auto-negotiation in progress 1 = Auto-negotiation and speed detection failed. Using default values for speed and duplex (defaults are 10Mbps and half duplex). 2 = Auto negotiation failed but detected speed. Duplex was defaulted (default is half duplex). 3 = Successfully negotiated speed and duplex. 4 = Auto-negotiation not attempted. Forced speed and duplex.
5	Manual Setting Requires Reset	0 indicates the interface can activate changes to link parameters (auto-negotiate, duplex mode, interface speed) automatically. 1 indicates the device requires a Reset service be issued to its Identity Object in order for the changes to take effect. **Note:** The Hilscher EtherNet/IP stack always requires a reset to the identity object in order for the configuration to take affect.
6	Local Hardware Fault	0 indicates the interface detects no local hardware fault; 1 indicates a local hardware fault is detected. The meaning of this is product-specific. Examples are an AUI/MII interface detects no transceiver attached or a radio modem detects no antennae attached. In contrast to the soft, possible self-correcting nature of the Link Status being inactive, this is assumed a hard-fault requiring user intervention. **Note:** The Hilscher EtherNet/IP stack never sets this hardware Fault flag.
7-31	Reserved	Is set to zero

Table 35: Ethernet Link - Instance Attribute 2 – Interface Status Flags

3.7.2.3 Physical Address

The Physical Address attribute contains the interface's MAC layer address. The Physical Address is an array of octets. Note that the Physical Address is not a settable attribute. The Ethernet address must be assigned by the manufacturer, and must be unique per IEEE 802.3 requirements. Devices with multiple ports but a single MAC interface (e.g., a device with an embedded switch technology) may use the same value for this attribute in each instance of the Ethernet Link Object. The general requirement is that the value of this attribute must be the MAC address used for packets to and from the device's own MAC interface over this physical port.

3.7.2.4 Interface Counters

The Interface Counters attribute contains counters relevant to the receipt of packets on the interface.

3.7.2.5 Media Counters

The Media Counters attribute contains counters specific to Ethernet media.

3.7.2.6 Interface Control

The Interface Control attribute is a structure consisting of Control Bits and Forced Interface Speed and shall be as follows:

Control Bits

Bit(s)	Name	Definition
0	Auto-negotiate	0 indicates 802.3 link auto-negotiation is disabled. 1 indicates auto-negotiation is enabled. If auto-negotiation is disabled, then the device shall use the settings indicated by the Forced Duplex Mode and Forced Interface Speed bits.
1	Forced Duplex Mode	If the Auto-negotiate bit is 0, the Forced Duplex Mode bit indicates whether the interface shall operate in full or half duplex mode. 0 indicates the interface duplex should be half duplex. 1 indicates the interface duplex should be full duplex. If auto-negotiation is enabled, attempting to set the Forced Duplex Mode bits results in a GRC hex 0x0C (Object State Conflict).
2-15	Reserved	Is set to zero

Table 36: Ethernet Link - Instance Attribute 6 – Interface Control (Control Bits)

Forced Interface Speed

If the Auto-negotiate bit is 0, the Forced Interface Speed bits indicate the speed at which the interface shall operate. Speed is specified in megabits per second (e.g., for 10 Mbps Ethernet, the Interface Speed shall be 10). If a requested speed is not supported by the Interface, the device returns a GRC hex 0x09 (Invalid Attribute Value).

If auto-negotiation is enabled, attempting to set the Forced Interface Speed results in a GRC hex 0x0C (Object State Conflict).

3.7.2.7 Interface Type

The Interface Type attribute indicates the type of the physical interface. Table 37 shows the Interface Type values.

Bit(s)	Type of interface
0	Unknown interface type
1	The interface is internal to the device, for example, in the case of an embedded switch.
2	Twisted-pair (e.g., 10Base-T, 100Base-TX, 1000Base-T, etc.)
3	Optical fiber (e.g., 100Base-FX)
4-255	Reserved

Table 37: Ethernet Link - Instance Attribute 7 – Interface Types

3.7.2.8 Interface State

The Interface State attribute shall indicate the current operational state of the interface. Table 38 shows the Interface State values.

Bit(s)	Interface State
0	Unknown interface state
1	The interface is enabled and is ready to send and receive data
2	The interface is disabled
3	The interface is testing
4-255	Reserved

Table 38: Ethernet Link - Instance Attribute 8 – Interface State

3.7.2.9 Admin State

The Admin State attribute shall allow administrative setting of the interface state. Table 39 shows the Admin State values. This attribute shall be stored in non-volatile memory.

Bit(s)	Admin State
0	Reserved
1	Enable the interface
2	Disable the interface
3-255	Reserved

Table 39: Ethernet Link - Instance Attribute 9 – Admin State

3.7.2.10 Interface Label

The Interface Label attribute is a text string that describes the interface. The content of the string is vendor specific. The maximum number of characters in this string is 64. This attribute shall be stored in non-volatile memory.

Note:

1. The default Interface Label values in the Hilscher EtherNet/IP stack for Ethernet port 0 and port 1 (Instances 1 and 2) are “port1” and “port2”, respectively.

   The default values can be changed using the packet command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.

2. The Interface Label values for instance 1 and instance 2 should correspond to the port labels that are present on the devices hardware ports.

3. The Interface Label values for instance 1 and instance 2 must correspond to the Interface Label entries in the EDS file (section “[Ethernet Link Class]”).

3.7.2.11 Interface Capability

The Interface Capability attribute indicates the set of capabilities for the interface. The attribute is a structure with two main elements: Capability bits and Speed/Duplex options. Capability bits contains an array of bits that indicate whether the interface supports capabilities such as auto-negotiation and auto-MDIX. Table 40 specifies the capability bits.

Bit(s)	Called	Definition
0	Manual Setting Requires Reset	Indicates whether or not the device requires a reset to apply changes made to the Interface Control attribute (#6). 0 = Indicates that the device automatically applies changes made to the Interface Control attribute (#6) and, therefore, does not require a reset in order for changes to take effect. This is the value this bit shall have when the Interface Control attribute (#6) is not implemented. 1 = Indicates that the device does not automatically apply changes made to the Interface Control attribute (#6) and, therefore, will require a reset in order for changes to take effect. **Note:** this bit shall also be replicated in the Interface Flags attribute (#2) in order to retain backwards compatibility with previous object revisions.
1	Auto-negotiate	0 = Indicates that the interface does not support link auto-negotiation 1 = Indicates that the interface supports link auto-negotiation
2	Auto-MDIX	0 = Indicates that the interface does not support auto MDIX operation 1 = Indicates that the interface supports auto MDIX operation
3	Manual Speed/Duplex	0 = Indicates that the interface does not support manual setting of speed/duplex. The Interface Control attribute (#6) shall not be supported. 1 = Indicates that the interface supports manual setting of speed/duplex via the Interface Control attribute (#6)
4-31	Reserved	Shall be set to 0

Table 40: Ethernet Link - Instance Attribute 11 – Capability Bits

The Speed/Duplex Options element holds an array that indicates the speed/duplex pairs that may be set via the Interface Control instance attribute (#6). One speed/duplex pair (e.g., 10 Mbps-half duplex, 100 Mbps-full duplex, etc.) shall be returned for each combination supported by the interface.

3.7.3 Supported Services

- Get_Attribute_Single (Service Code: 0x0E)
- Set_Attribute_Single (Service Code: 0x10)
- Get_and_Clear (Service Code: 0x4C)

3.8 Time Sync Object (Class Code: 0x43)

A detailed description of CIP Sync and the Time Sync object (class ID 0x43) can be found in reference [9].

3.9 DLR Object (Class Code: 0x47)

The Device Level Ring (DLR) Object provides status information interface for the DLR protocol. The DLR protocol is a layer 2 protocol that enables the use of an Ethernet ring topology. For further information regarding DLR see section *DLR* on page 246.

3.9.1 Class Attributes

Attribute ID	Access Rule		Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]				
1	Get	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is three (03).
2	Get	Get	Max. Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level. The current value assigned to this attribute is one (01).

1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.

Table 41: DLR - Class Attributes

3.9.2 Instance Attributes

Att ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
1	Get	Get	V	Network Topology	USINT	Current network topology mode	0 indicates “Linear” 1 indicates “Ring” See section 3.9.2.1
2	Get	Get	V	Network Status	USINT	Current status of network	0 indicates “Normal” 1 indicates “Ring Fault” 2 indicates “Unexpected Loop Detected” 3 indicates “Partial Network Fault” 4 indicates “Rapid Fault/Restore Cycle” See section 3.9.2.2
10	Get	Get	V	Active Supervisor Address	STRUCT of:	IP and/or MAC address of the active ring supervisor	See section 3.9.2.3
					UDINT	Supervisor IP Address	A Value of 0 indicates no IP Address has been configured for the device
					ARRAY of 6 USINTs	Supervisor MAC Address	Ethernet MAC address
12	Get	Get	NV	Capability Flags	DWORD	Describes the DLR capabilities of the device	See section 3.9.2.4

1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.

Table 42: DLR - Instance Attributes

3.9.2.1 Network Topology

The Network Topology attribute indicates the current network topology mode. A value of 0 shall indicate “Linear” topology. A value of 1 shall indicate “Ring” topology.

3.9.2.2 Network Status

The Network Status attribute provides current status of the network based the device’s view of the network, as specified in the DLR behavior in Chapter 9. Table 5-5.3 shows the possible values:

Bit(s)	Definition
0	Normal operation in both Ring and Linear Network Topology modes.
1	Ring Fault. A ring fault has been detected. Valid only when Network Topology is Ring.
2	Unexpected Loop Detected. A loop has been detected in the network. Valid only when the Network Topology is Linear.
3	Partial Network Fault. A network fault has been detected in one direction only. Valid only when Network Topology is Ring and the node is the active ring supervisor (Ring Supervisor not supported by Hilscher EtherNet/IP stack).
4	Rapid Fault/Restore Cycle. A series of rapid ring fault/restore cycles has been detected (DLR Supervisor only).

Table 43: DLR - Instance Attribute 2 – Network Status

3.9.2.3 Active Supervisor Address

This attribute contains the IP address and/or Ethernet MAC address of the active ring supervisor. The initial values of IP address and Ethernet MAC address is 0, until the active ring supervisor is determined.

3.9.2.4 Capability Flags

The Capability Flags describe the DLR capabilities of the device.

Bit(s)	Name	Definition
0	Announce-based Ring Node[1)]	Set if device’s ring node implementation is based on processing of Announce frames. (The Hilscher implementation is Beacon-based; see definition of next bit)
1	Beacon-based Ring Node1[1)]	Set if device’s ring node implementation is based on processing of Beacon frames. (This is the Hilscher Implementation)
2-4	Reserved	Is set to zero.
5	Supervisor Capable	Set if device is capable of providing the supervisor function (not supported by the Hilscher EtherNet/IP stack).
6	Redundant Gateway Capable	Set if device is capable of providing the redundant gateway function. (not supported by the Hilscher EtherNet/IP stack)
7	Flush_Table frame Capable	Set if device is capable of supporting the Flush_Tables frame.
8-31	Reserved	Is set to zero.
1) Bits 0 and 1 are mutually exclusive. Exactly one of these bits shall be set in the attribute value that a device reports.		

Table 44: DLR - Instance Attribute 12 – Capability Flags

3.9.3 Supported Services

- Get_Attribute_Single (Service Code: 0x0E) is supported for class and instance attributes.
- Get_Attribute_All (Service Code: 0x01) is supported for instance attributes only.

3.10 Quality of Service Object (Class Code: 0x48)

Quality of Service (QoS) is a general term that is applied to mechanisms used to treat traffic streams with different relative priorities or other delivery characteristics. Standard QoS mechanisms include IEEE 802.1D/Q (Ethernet frame priority) and Differentiated Services (DiffServ) in the TCP/IP protocol suite.

The QoS Object provides a means to configure certain QoS-related behaviors in EtherNet/IP devices.

The QoS Object is required for devices that support sending EtherNet/IP messages with nonzero DiffServ code points (DSCP), or sending EtherNet/IP messages in 802.1Q tagged frames or devices that support the DLR functionality.

3.10.1 Class Attributes

Attribute ID	Access Rule		Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]				
1	Get	Get	Revision	UINT	Revision of this object	The current value assigned to this attribute is 1.
2	Get	Get	Max. Instance	UINT	Maximum instance number of an object currently created in this class level of the device.	The largest instance number of a created object at this class hierarchy level.
1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.						

Table 45: QoS - Class Attributes

3.10.2 Instance Attributes

Att ID	Access Rule		NV	Name	Data Type	Description of Attribute	Semantics of Values
	From Network	From Host[1)]					
1[2)]	Get, Set	Get	NV	802.1Q Tag Enable	USINT	Enables or disables sending 802.1Q frames on CIP and IEEE 1588 messages	A value of 0 indicates tagged frames disabled. A value of 1 indicates tagged frames enabled. The default value shall be 0.
2[3)]	Get, Set	Get	NV	DSCP PTP Event	USINT	DSCP value for PTP (IEEE 1588) event messages	
3[3)]	Get, Set	Get	NV	DSCP PTP General	USINT	DSCP value for PTP (IEEE 1588) general messages	
4[2)]	Get, Set	Get	NV	DSCP Urgent	USINT	DSCP value for CIP transport class 0/1 Urgent priority messages	
5[2)]	Get, Set	Get	NV	DSCP Scheduled	USINT	DSCP value for CIP transport class 0/1 Scheduled priority messages	
6[2)]	Get, Set	Get	NV	DSCP High	USINT	DSCP value for CIP transport class 0/1 High priority messages	
7[2)]	Get, Set	Get	NV	DSCP Low	USINT	DSCP value for CIP transport class 0/1 low priority messages	
8[2)]	Get, Set	Get	NV	DSCP Explicit	USINT	DSCP value for CIP explicit messages (transport class 2/3 and UCMM) and all other EtherNet/IP encapsulation messages	

1) Related to API command `EIP_OBJECT_CIP_SERVICE_REQ/CNF` – CIP Service Request.

2) If the attribute value is changed from the network side, the host application is notified via the indication `EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES` – CIP Object Change Indication (see section 6.2.18 on page 201)

3) This attribute is only available when the CIP Time Sync object is used.

Table 46: QoS - Instance Attributes

3.10.2.1 802.1Q Tag Enable

The 802.1Q Tag Enable attribute enables or disables sending 802.1Q frames on CIP. When the attribute is enabled, the device sends 802.1Q frames for all CIP.

A value of 1 indicates enabled. A value of 0 indicates disabled. The default value for the attribute is 0. A change to the value of the attribute takes effect the next time the device restarts.

Note: Devices always use the corresponding DSCP values regardless of whether 802.1Q frames are enabled or disabled.

3.10.2.2 DSCP Value Attributes

Attributes 4 through 8 contain the DSCP values that are used for the different types of EtherNet/IP traffic.

The valid range of values for these attributes is 0-63. Table 47 shows the default DSCP values and traffic usages.

Attr ID	Name	Traffic Type Usage	Default DSCP		
			dec	bin	hex
2	DSCP PTP Event (not supported)	PTP (IEEE 1588) event messages	59	111011	3B
3	DSCP PTP General (not supported)	PTP (IEEE 1588) general messages	47	101111	2F
4	DSCP Urgent	CIP transport class 0/1 messages with Urgent priority	55	110111	37
5	DSCP Scheduled	CIP transport class 0/1 messages with Scheduled priority	47	101111	2F
6	DSCP High	CIP transport class 0/1 messages with High priority	43	101011	2B
7	DSCP Low	CIP transport class 0/1 messages with Low priority	31	011111	1F
8	DSCP Explicit	CIP UCMM CIP transport class 2/3 All other EtherNet/IP encapsulation messages	27	011011	1B

Table 47: QoS - Instance Attribute 4-8 – DSCP Values

A change to the value of the above attributes will take effect the next time the device restarts.

3.10.3 Supported Services

- Get_Attribute_Single (Service Code: 0x0E)
- Set_Attribute_Single (Service Code: 0x10)

4 Getting Started/Configuration

4.1 Task Structure of the EtherNet/IP Adapter Stack

The figure below displays the internal structure of the tasks which together represent the EtherNet/IP Adapter Stack:

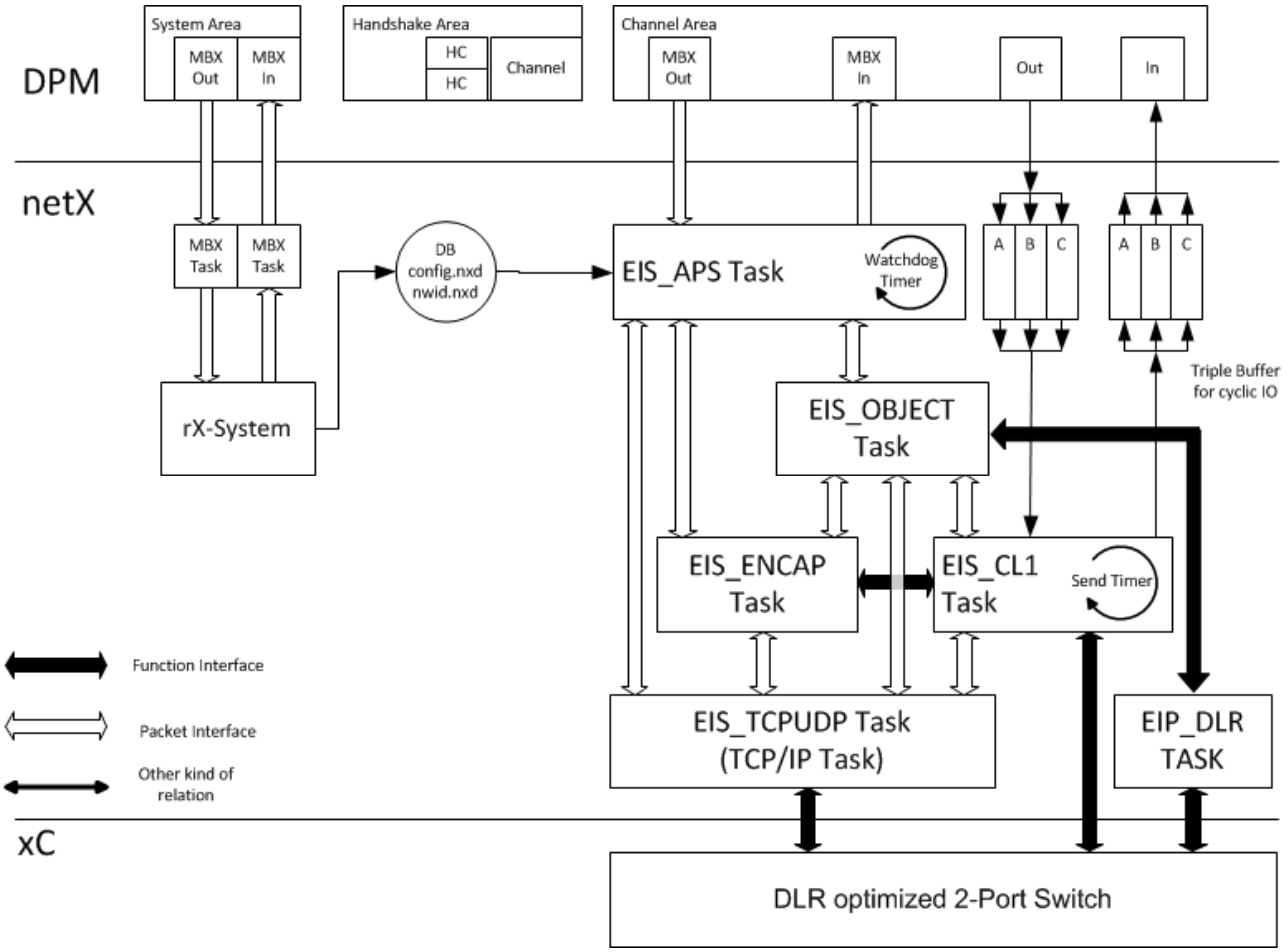

Figure 9: Task Structure of the EtherNet/IP Adapter Stack

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

The user application only accesses the task located in the highest layer namely the EIS_APS-Task which constitute the application interface of the EtherNet/IP Adapter Stack.

The EIS_OBJECT task, EIS_ENCAP task and EIS_CL1 task represent the core of the EtherNet/IP Adapter Stack.

The TCP/IP task represents the TCP/IP Stack, which is used by the EtherNet/IP Adapter.

4.1.1 EIS_APS task

The EIS_APS task provides the interface to the user application and the control of the stack. It also completely handles the Dual Port Memory interface of the communication channel. In detail, it is responsible for the following:

- Handling the communication channels DPM-interface
  - Process data exchange
  - channel mailboxes
  - Watchdog
  - Provides Status and diagnostic
- Handling applications packets (all packets described in Protocol Interface Manual)
  - Configuration packets
  - Packet Routing
- Handling stacks indication packets
- Provide information about state of every Connection contained in configuration
- Evaluation of data base files
- Preparation of configuration data

4.1.2 EIS_OBJECT task

The EIP_OBJECT task is the main part of the EtherNet/IP Stack. The task is responsible for the following items:

- CIP object directory
- Connection establishment
- Explicit messaging
- Connection management

4.1.3 EIS_ENCAP task

The EIS_ENCAP task implements the encapsulation layer of the EtherNet/IP. It handles the interface to the TCP/IP Stack and manages all TCP connections.

4.1.4 EIS_CL1 task

The EIS_CL1 task has the highest priority. The Task is responsible for the implicit messaging. The Task has an interface to the EDD and manages the handling of the cyclic communication.

4.1.5 EIP_DLR task

The EIS_DLR task provides support for the DLR technology for creating a single ring topology with media redundancy. For more information see next section.

4.1.6 TCP/IP task

The TCP/IP task coordinates the EtherNet/IP stack with the underlying TCP/IP stack. It provides services required by the EIS_ENCAP task.

4.2 Configuration Procedures

The following ways are available to configure the EtherNet/IP Adapter:

- Using the Packet API of the EtherNet/IP Protocol Stack
- By netX configuration and diagnostic utility
- Using the Configuration Tool SYCON.net

4.2.1 Using the Packet API of the EtherNet/IP Protocol Stack

Depending of the interface the host application has to the EtherNet/IP stack, there are different possibilities of how configuration can be performed.

For more information how to accomplish this, please see section 4.3 “Configuration Using the Packet API”.

4.2.2 Using the Configuration Tool SYCON.net

The easiest way to configure the EtherNet/IP Adapter is using Hilscher’s configuration tool SYCON.net. This tool is described in a separate documentation.

4.3 Configuration Using the Packet API

In section 3 “*Available CIP Classes in the Hilscher EtherNet/IP Stack*” the default Hilscher CIP Object Model is displayed. This section explains how these objects can be configured using the Packet API of the EtherNet/IP stack.

In order to determine what packets you should use first you need to select one of the following scenarios the EtherNet/IP Protocol Stack can be run with.

- **Scenario: Loadable Firmware (LFW)**

  The host application and the EtherNet/IP Adapter Protocol Stack run on different processors. While the host application runs on a separate CPU the EtherNet/IP Adapter Protocol Stack runs on the netX processor together with a connecting software layer, the AP task.

  The connection of host application and Protocol Stack is accomplished via a driver (Hilscher cifX Driver, Hilscher netX Driver) as software layer on the host side and the AP task as software layer on the netX side. Both communicate via a dual port memory as shown in Figure 10.

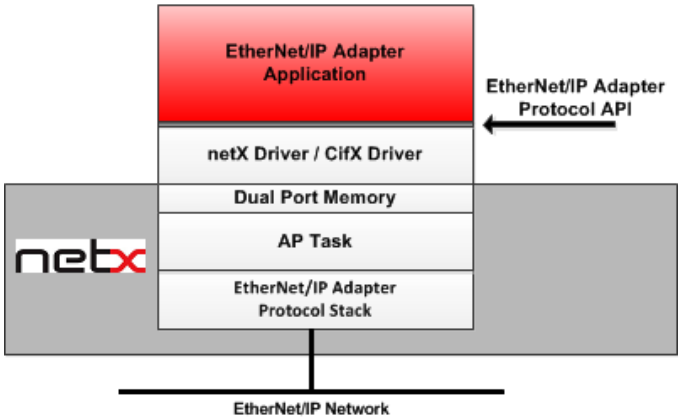

Figure 10: Loadable Firmware Scenario

- **Scenario: Linkable Object Module (LOM)**

  Both the host application and the EtherNet/IP Adapter Protocol Stack run on the same processor, the netX as shown in Figure 11. There is no need for drivers or a stack-specific AP task. Application and Protocol Stack are statically linked.

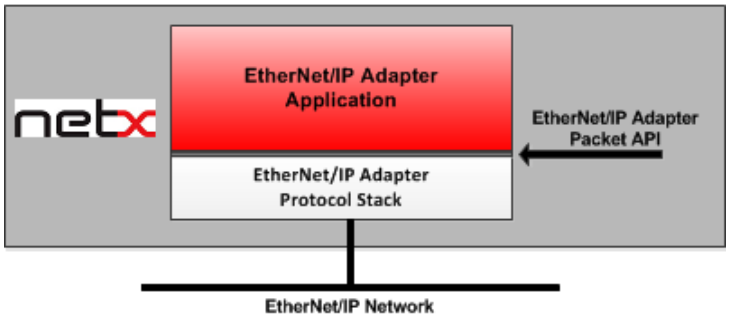

Figure 11: Linkable Object Modules Scenario

After making the scenario decision there are some Packet Sets available. The Packet Set must be chosen depending on the requirements for the device you want to develop and on the CIP Object Model you want the device to have.

Table 48: Packet Sets shows the available sets and describes the general functionalities that come with the corresponding set.

Scenario	Name of Packet Set	Description
Loadable Firmware	**Basic** (see section 4.3.1 “Basic Packet Set” for a detailed packet list)	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) ■ Support of Quick Connect Using this configuration the device’s CIP object model will look like the one that is illustrated in *Figure 8*. **Note:** If your application/device needs a special functionality that is not covered by the basic Packet Set, please use the Extended Packet Set described below.
	Extended (see section 4.3.2 “Extended Packet Set” for a detailed packet list)	Using this Configuration Set, the host application is free to design the device’s CIP object model in all aspects. In addition to the functionalities that come with the Basic Configuration Set, this set provides the following: ■ Up to 32 assembly instances possible. ■ Additional configuration assembly possible (necessary if the device needs configuration parameters from the Scanner/Master/PLC before going into cyclic communication). ■ Use additional CIP objects (that might be necessary when using a special CIP Profile (see section 2.7)). These objects are also accessible via acyclic/explicit messages. This Configuration Set can, of course, also be used if only a basic configuration is desired.
Linkable object module	**Stack** (see section 4.3.3 “Stack Configuration Set” for a detailed packet list)	This Configuration Set corresponds basically to the Extended Configuration Set of the Loadable Firmware. There are only some differences in the packet handling independent of the configuration.

Table 48: Packet Sets

4.3.1 Basic Packet Set

4.3.1.1 Configuration Packets

To configure the EtherNet/IP Stack's default CIP objects the following packets are necessary:

Section	Packet Name	Command Code (REQ/CNF)	Page
6.1.1	`EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ/CNF` – Configure the Device with Configuration Parameter	0x3612/ 0x3613	92
-	`RCX_REGISTER_APP_REQ` – Register the Application at the stack in order to receive indications (see reference [2])	0x2F10/ 0x2F11	
-	`RCX_CHANNEL_INIT_REQ` – Perform channel initialization (see reference [2])	0x2F80/ 0x2F81	

Table 49: Basic Packet Set - Configuration Packets

The packets of Packet Set "Basic" (Table 49) should be sent in the order that is illustrated in Figure 12.

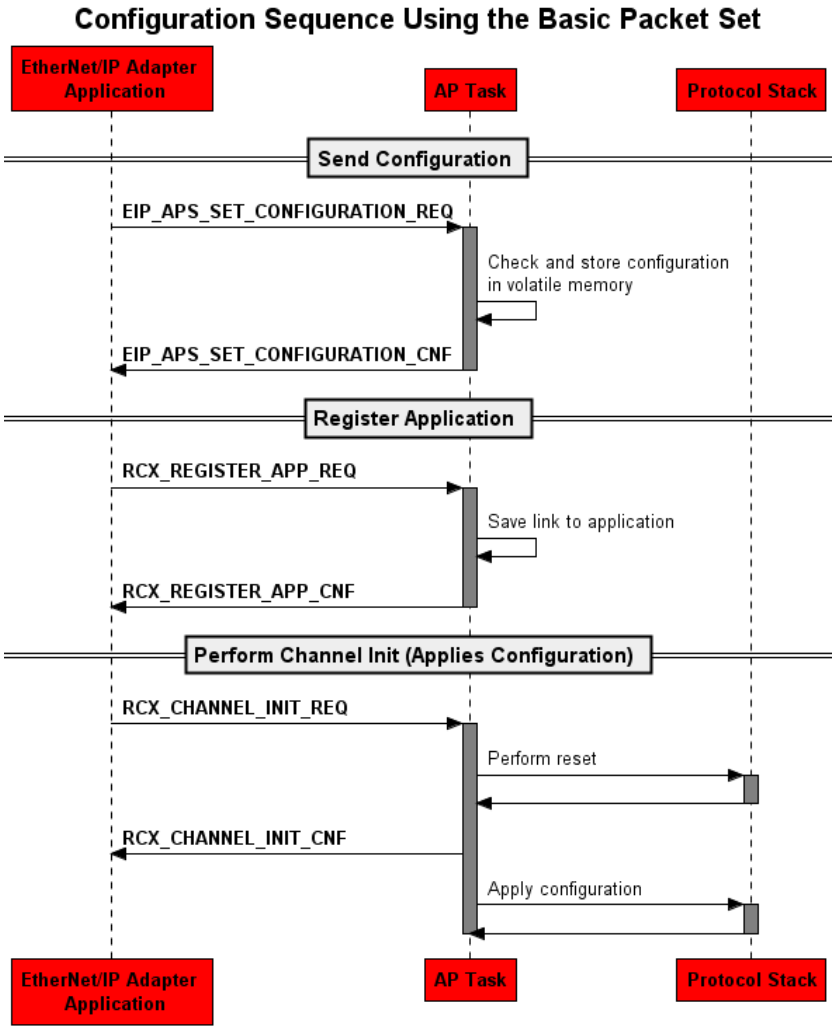

Figure 12: Configuration Sequence Using the Basic Packet Set

4.3.1.2 Optional Request Packets

In addition to the request packets related to configuration, there are some more request packets the application can use:

No. of section	Packet Name	Command code (IND/RES)	Page
6.1.5	EIP_APS_GET_MS_NS_REQ/CNF – Get Module Status/Network Status	0x360E/ 0x360F	113
-	RCX_UNREGISTER_APP_REQ – Unregister the Application (see reference [2])	0x2F12/ 0x2F13	
6.1.2	EIP_APS_CLEAR_WATCHDOG_REQ/CNF – Clear Watchdog error	0x3602/ 0x3603	104

Table 50: Additional Request Packets Using the Basic Packet Set

4.3.1.3 Indication Packets the Host Application Needs to Handle

In addition to the request packets, there are some indication packets the application needs to handle:

No. of section	Packet Name	Command code (IND/RES)	Page
6.2.18	EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES – CIP Object Change Indication	0x1AFA/ 0x1AFB	201
6.2.8	EIP_OBJECT_RESET_IND/RES – Indication of a Reset Request from	0x1A24/ 0x1A25	157
6.2.2	EIP_OBJECT_CONNECTION_IND/RES – Connection State Change Indication	0x1A2E/ 0x1A2F	122
6.2.1	EIP_OBJECT_FAULT_IND/RES – Fault Indication	0x1A30/ 0x1A31	119
6.2.20	RCX_LINK_STATUS_CHANGE_IND/RES – Link Status Change	0x2F8A/ 0x2F8B	209

Table 51: Indication Packets Using the Basic Packet Set

4.3.2 Extended Packet Set

4.3.2.1 Configuration Packets

When using the Extended Packet Set the packets listed in Table 52 "*Extended Packet Set - Configuration Packets*" are available. Please note, that there are required and optional packets depending on the desired functionalities your device shall support.

Affects	No. of section	Packet Name	Command Code REQ/ CNF	Page	Required /Optional
General Configuration		RCX_REGISTER_APP_REQ – Register Application (see DPM Manual for more information) Registers the EtherNet/IP Adapter application at the AP-Task. All necessary indication packets can now be received by the application.	0x2F10/ 0x2F11	See reference [1]	Required
Identity Object (0x01)	6.2.6	EIP_OBJECT_ID_SETDEVICEINFO_REQ/CNF – Set the Device's Identity Information Setting all necessary attributes of the CIP Identity Object.	0x1A16/ 0x1A17	148	Required
Addressed CIP Object	6.2.17	EIP_OBJECT_CIP_SERVICE_REQ/CNF – CIP Service Request Used to set attribute data of stack's internal CIP Objects	0x1AF8/ 0x1AF9	196	Required
Assembly Object (0x04) Cyclic Communication/ Implicit Messaging	6.2.5	EIP_OBJECT_AS_REGISTER_REQ/CNF – Register a new Assembly Instance Register an assembly instance as output, input or configuration assembly.	0x1A0C/ 0x1A0D	141	Optional[1]
Device's general CIP Object Model	6.2.3	EIP_OBJECT_MR_REGISTER_REQ/CNF – Register an additional Object Class at the Message Router Registers an additional CIP object class at the Message Router Object. Additional CIP Objects may be necessary when the device shall use a specific CIP Profile (see section 2.7 "CIP Device Profiles")	0x1A02/ 0x1A03	130	Optional
	6.2.11	EIP_OBJECT_REGISTER_SERVICE_REQ/CNF – Register Service Register an additional CIP service.	0x1A44/ 0x1A45	168	Optional
QoS Object (0x48)	6.2.16	EIP_OBJECT_CFG_QOS_REQ/CNF – Configure the QoS Object Configures the QoS (Quality of Service) Object	0x1A42/ 0x1A43	192	Optional[2]
Device's general CIP Object Model	6.2.19	EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ/CNF – CIP Object Attribute Activate Request	0x1AFC/ 0x1AFD	205	Optional

	6.2.14	EIP_OBJECT_SET_PARAMETER_REQ/CNF – Set Parameter Enable/disable specific functionalities within the EtherNet/IP Stack. (Please have a look at the packet description for further details)	0x1AF2/ 0x1AF3	181	Optional
	6.1.3	EIP_APS_SET_PARAMETER_REQ/CNF – Set Parameter Flags Enable/disable specific functionalities within the AP-Task. (Please have a look at the packet description for further details)	0x360A/ 0x360B	107	Optional
TCP/IP Interface Object (0xF5) Ethernet Link Object	See reference [3]	TCPIP_IP_CMD_SET_CONFIG_REQ – Set the TCP/IP Configuration Sets TCP/IP Parameters and Ethernet Port Configuration	0x200/ 0x201	See reference [3]	Required
		RCX_START_STOP_COMM_REQ (see DPM Manual for more information) Starts or stops the network communication, i.e. used to set or clear the netX's BUS_ON signal, according to the contained parameter	0x2F30/ 0x2F31	See reference [2]	Required

[1] Required if implicit messaging (cyclic I/O data exchange) shall be supported

[2] Required if DLR (Device Level Ring) shall be supported

Table 52: Extended Packet Set - Configuration Packets

The following Figure 13 illustrates an example packet sequence using the Extended Packet Set. Using the shown sequence and packets will basically give you a configuration that is equal to the configuration you get when using the Basic Packet Set. Of course, you can use additionally packets to further extend your Device's object model or activate additional functionalities.

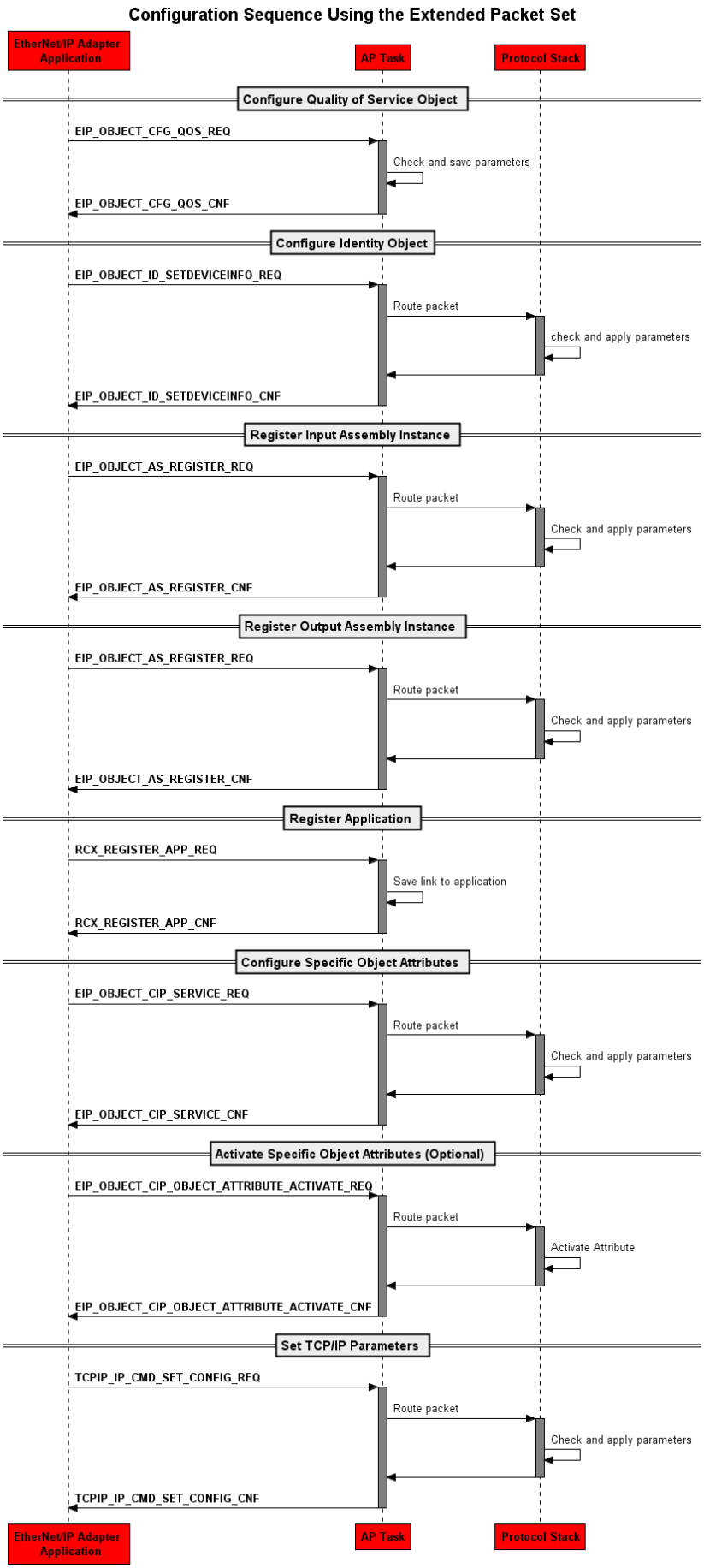

Figure 13: Configuration Sequence Using the Extended Packet Set

4.3.2.2 Optional Request Packets

In addition to the request packets related to configuration, there are some more request packets the application can use during runtime:

No. of section	Packet Name	Command code (IND/RES)	Page
6.1.5	EIP_APS_GET_MS_NS_REQ/CNF – Get Module Status/Network Status	0x360E/ 0x360F	113
	RCX_UNREGISTER_APP_REQ – Unregister the Application (see reference [2])	0x2F12/ 0x2F13	
6.1.2	EIP_APS_CLEAR_WATCHDOG_REQ/CNF – Clear Watchdog error	0x3602/ 0x3603	104

Table 53: Additional Request Packets Using the Extended Packet Set

4.3.2.3 Indication Packets the Host Application Needs to Handle

In addition to the request packets, there are some indication packets the application needs to handle:

No. of section	Packet Name	Command code (IND/RES)	Page	Required/Optional
6.2.18	EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES – CIP Object Change Indication	0x1AFA/ 0x1AFB	201	Required
6.2.8	EIP_OBJECT_RESET_IND/RES – Indication of a Reset Request from	0x1A24/ 0x1A25	157	Required
6.2.2	EIP_OBJECT_CONNECTION_IND/RES – Connection State Change Indication	0x1A2E/ 0x1A2F	122	Required
6.2.12	EIP_OBJECT_CONNECTION_CONFIG_IND/RES – Indication of Configuration Data received during Connection Establishment	0x1A40/ 0x1A41	171	Conditional[1]
6.2.1	EIP_OBJECT_FAULT_IND/RES – Fault Indication	0x1A30/ 0x1A31	119	Required
6.2.20	RCX_LINK_STATUS_CHANGE_IND/RES – Link Status Change	0x2F8A/ 0x2F8B	209	Required
6.2.4	EIP_OBJECT_CL3_SERVICE_IND/RES - Indication of acyclic Data Transfer	0x1A3E/ 0x1A3F	134	Conditional[2]
6.1.4	EIP_APS_MS_NS_CHANGE_IND/RES – Module Status/ Network Status Change Indication	0x360C/ 0x360D	110	Conditional[3]

No. of section	Packet Name	Command code (IND/RES)	Page	Required/Optional
[1] Only necessary if configuration assembly has been registered using command `EIP_OBJECT_AS_REGISTER_REQ (0x1A0C)` [2] Only necessary if additional service or CIP object has been registered using command `EIP_OBJECT_REGISTER_SERVICE_REQ (0x1A44) or EIP_OBJECT_MR_REGISTER_REQ (0x1A02)` [3] Only necessary if functionality has been activated using command `EIP_APS_SET_PARAMETER_REQ (0x360A)`				

Table 54: Indication Packets Using the Extended Packet Set

4.3.3 Stack Configuration Set

4.3.3.1 Configuration Packets

When using the Stack Packet Set the packets listed in Table 55 “Stack Packet Set - Configuration Packets” are available. Please note, that there are required and optional packets depending on the desired functionalities your device shall support.

Affects	No. of section	Packet Name	Command Code REQ/CNF	Page	Required/ Optional
Identity Object (0x01)	6.2.6	EIP_OBJECT_ID_SETDEVICEINFO_REQ/CNF – Set the Device’s Identity Information Setting all necessary attributes of the CIP Identity Object.	0x1A16/ 0x1A17	148	Required
Addressed CIP Object	6.2.17	EIP_OBJECT_CIP_SERVICE_REQ/CNF – CIP Service Request Used to set attribute data of stack’s internal CIP Objects	0x1AF8/ 0x1AF9	196	Required
Assembly Object (0x04) Cyclic Communication/ Implicit Messaging	6.2.5	EIP_OBJECT_AS_REGISTER_REQ/CNF – Register a new Assembly Instance Register an assembly instance as output, input or configuration assembly.	0x1A0C/ 0x1A0D	141	Optional[1]
Device’s general CIP Object Model	6.2.3	EIP_OBJECT_MR_REGISTER_REQ/CNF – Register an additional Object Class at the Message Router Registers an additional CIP object class at the Message Router Object. Additional CIP Objects may be necessary when the device shall use a specific CIP Profile (see section 2.7 “CIP Device Profiles”)	0x1A02/ 0x1A03	130	Optional
	6.2.11	EIP_OBJECT_REGISTER_SERVICE_REQ/CNF – Register Service Register an additional CIP service.	0x1A44/ 0x1A45	168	Optional
QoS Object (0x48)	6.2.16	EIP_OBJECT_CFG_QOS_REQ/CNF – Configure the QoS Object Configures the QoS (Quality of Service) Object	0x1A42/ 0x1A43	192	Optional[2]
Device’s general CIP Object Model	6.2.19	EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ/CNF – CIP Object Attribute Activate Request	0x1AFC/ 0x1AFD	205	Optional
	6.2.14	EIP_OBJECT_SET_PARAMETER_REQ/CNF – Set Parameter Enable/disable specific functionalities within the EtherNet/IP Stack. (Please have a look at the packet description for further details)	0x1AF2/ 0x1AF3	181	Optional

TCP/IP Interface Object (0xF5) Ethernet Link Object	See reference [3]	TCPIP_IP_CMD_SET_CONFIG_REQ – Set the TCP/IP Configuration Sets TCP/IP Parameters and Ethernet Port Configuration	0x200/ 0x201	See reference [3]	Required
Cyclic Communication/ Implicit Messaging	6.2.10	EIP_OBJECT_READY_REQ/CNF – Set Ready and Run/Idle State		165	Required
[1] Required if implicit messaging (cyclic I/O data exchange) shall be supported [2] Required if DLR (Device Level Ring) shall be supported					

Table 55: Stack Packet Set - Configuration Packets

The following Figure 14 illustrates an example packet sequence using the Stack Packet Set. Using the shown sequence and packets will basically give you a configuration that is equal to the configuration you get when using the Basic Packet Set. Of course, you can use additionally packets to further extend your Device's object model or activate additional functionalities.

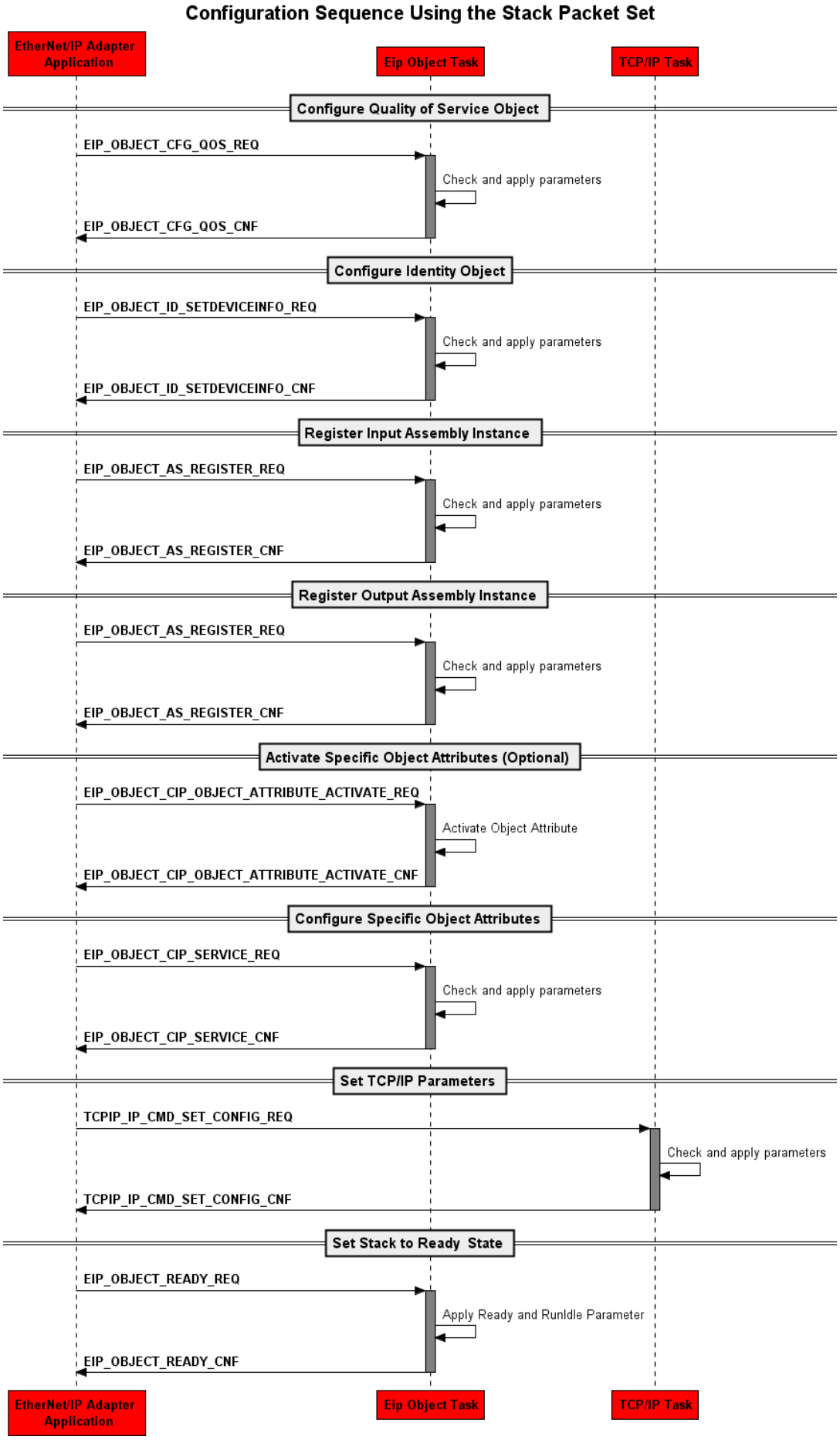

Figure 14: Configuration Sequence Using the Stack Packet Set

4.3.3.2 Indication Packets the Host Application Needs to Handle

In addition to the request packets, there are some indication packets the application needs to handle:

No. of section	Packet Name	Command code (IND/RES)	Page	Required /Optional
6.2.18	EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES – CIP Object Change Indication	0x1AFA/ 0x1AFB	201	Required
6.2.8	EIP_OBJECT_RESET_IND/RES – Indication of a Reset Request from	0x1A24/ 0x1A25	157	Required
6.2.2	EIP_OBJECT_CONNECTION_IND/RES – Connection State Change Indication	0x1A2E/ 0x1A2F	122	Required
6.2.12	EIP_OBJECT_CONNECTION_CONFIG_IND/RES – Indication of Configuration Data received during Connection Establishment	0x1A40/ 0x1A41	171	Conditional[1]
6.2.1	EIP_OBJECT_FAULT_IND/RES – Fault Indication	0x1A30/ 0x1A31	119	Required
6.2.4	EIP_OBJECT_CL3_SERVICE_IND/RES - Indication of acyclic Data Transfer	0x1A3E/ 0x1A3F	134	Conditional[2]

[1] Only necessary if configuration assembly has been registered using command EIP_OBJECT_AS_REGISTER_REQ/CNF – Register a new Assembly Instance

[2] Only necessary if additional service or CIP object has been registered using command EIP_OBJECT_REGISTER_SERVICE_REQ (0x1A44) or EIP_OBJECT_MR_REGISTER_REQ (0x1A02)

Table 56: Indication Packets Using the Stack Packet Set

5 Status information

The EtherNet/IP Adapter provides status information in the dual-port memory. The status information has a common block (protocol-independent) and a protocol-specific block (extended status). For the EtherNet/IP Adapter protocol implementation, the extended status is not used.

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

6 The Application Interface

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

The following send and receive packets are exchanged with the task via its queues in the structure like it is described in the netX DPM Interface manual. All packets should be exchanged with the APS-Task queue.

The structures of these packets and their values are described in the sections below.

In order to know what packets are needed to configure the stack please read section 4.3 “Configuration Using the Packet API”.

6.1 The EIS_APS-Task

The EIS_APS-Task is the interface between dual port memory and the EtherNet/IP-Adapter stack. All services should be sent to this task. For addressing a packet to the EIS_APS-Task the destination address 0x20 is used.

In detail, the following functionality is provided by the EIS_APS-Task:

Overview over the Packets of the APS-Task			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
6.1.1	EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ/CNF – Configure the Device with Configuration Parameter	0x3612/ 0x3613	92
	RCX_REGISTER_APP_REQ – Register the Application at the stack in order to receive indications (see reference [2])	0x2F10/ 0x2F11	
	RCX_UNREGISTER_APP_REQ – Unregister the Application (see reference [2])	0x2F12/ 0x2F13	
6.1.2	EIP_APS_CLEAR_WATCHDOG_REQ/CNF – Clear Watchdog error	0x3602/ 0x3603	104
6.1.3	EIP_APS_SET_PARAMETER_REQ/CNF	0x360A/ 0x360B	107
6.1.4	EIP_APS_MS_NS_CHANGE_IND/RES	0x360C/ 0x360D	110
6.1.5	EIP_APS_GET_MS_NS_REQ/CNF	0x360E 0x360F	113
6.1.7	Modify Configuration Parameters	0x2F86/ 0x2F87	117

Table 57: Overview over the Packets of the EIS_APS-Task of the EtherNet/IP-Adapter Protocol Stack

6.1.1 EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ/CNF – Configure the Device with Configuration Parameter

Note: This packet replaces the packet EIP_APS_SET_CONFIGURATION_REQ(cmd:0x3608). For compatibility reasons this packet is still supported. However, for new developments only the packet EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ (cmd: 0x3612) shall be used.

This service can be used by the host application in order to configure the device with configuration parameters. This packet is part of the basic packet 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 that can be used implicit/cyclic communication. The I/O data of theses instances will start at offset 0 at the dual port memory (relative offset to the input and output areas of the DPM).

Note: If you set usVendId, usProductType and usProductCode to zero, Hilscher's firmware standard values will be applied for the according variables.

The following rules apply for the behavior of the EtherNet/IP Adapter Stack when receiving a set configuration command:

- The configuration data is checked for consistency and integrity.
- In case of failure no data is accepted.
- In case of success the configuration parameters are stored internally (within the RAM).
- The parameterized data will be activated only after a channel init (RCX_CHANNEL_INIT_REQ).
- This packet does not perform any registration at the stack automatically. Registering must be performed with a separate packet such as the registration packet described in the netX Dual-Port-Memory Manual (RCX_REGISTER_APP_REQ, code 0x2F10).
- This request will be denied if the "configuration locked" flag is set in the DPM (for more information see reference [1]).

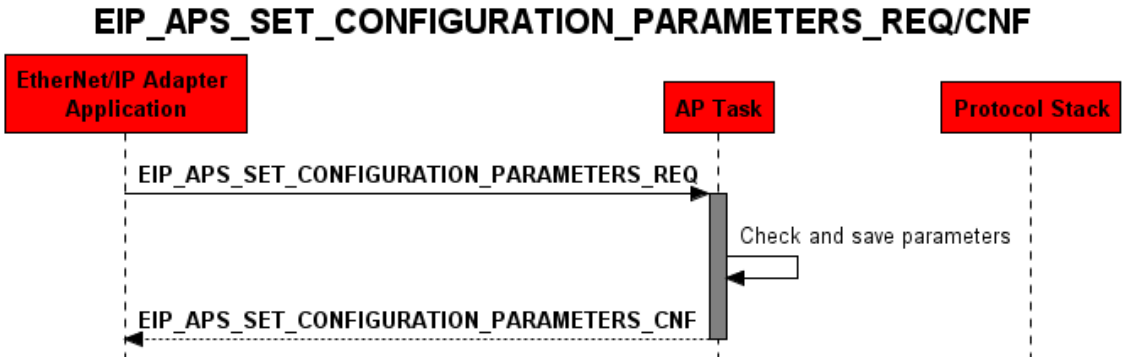

Figure 15: Sequence Diagram for the EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ/CNF Packet

Packet Structure Reference

```
typedef struct EIP DPMINTF QOS CONFIG Ttag
{
  TLR UINT32    ulQoSFlags;
  TLR UINT8     bTag802Enable;
  TLR UINT8     bDSCP PTP Event;
  TLR UINT8     bDSCP PTP General;
  TLR UINT8     bDSCP Urgent;
  TLR_UINT8     bDSCP_Scheduled;
  TLR UINT8     bDSCP High;
  TLR UINT8     bDSCP Low;
  TLR UINT8     bDSCP Explicit;
} EIP DPMINTF QOS CONFIG T;

typedef struct EIP_DPMINTF_TI_ACD_LAST_CONFLICT_Ttag
{
  TLR UINT8    bAcdActivity;      /*!< State of ACD activity when last
                                       conflict detected */

  TLR_UINT8    abRemoteMac[6];    /*!< MAC address of remote node from
                                       the ARP PDU in which a conflict was
                                       detected */

  TLR UINT8    abArpPdu[28];      /*!< Copy of the raw ARP PDU in which
                                       a conflict was detected. */
} EIP_DPMINTF_TI_ACD_LAST_CONFLICT_T;

typedef struct  APS CONFIGURATION PARAMETER SET V3 T tag
{
  TLR UINT32  ulSystemFlags;
  TLR_UINT32  ulWdgTime;
  TLR_UINT32  ulInputLen;
  TLR UINT32  ulOutputLen;
  TLR UINT32  ulTcpFlag;
  TLR UINT32  ulIpAddr;
  TLR UINT32  ulNetMask;
  TLR_UINT32  ulGateway;
  TLR_UINT16  usVendId;
  TLR UINT16  usProductType;
  TLR UINT16  usProductCode;
  TLR_UINT32  ulSerialNumber;
  TLR_UINT8   bMinorRev;
  TLR_UINT8   bMajorRev;
  TLR UINT8   abDeviceName[32];
  TLR UINT32  ulInputAssInstance;
  TLR UINT32  ulInputAssFlags;
  TLR_UINT32  ulOutputAssInstance;
  TLR_UINT32  ulOutputAssFlags;
  EIP_DPMINTF_QOS_CONFIG_T tQoS_Config;
  TLR UINT32   ulNameServer;
  TLR UINT32   ulNameServer 2;
  TLR UINT8    abDomainName[48 + 2];
  TLR_UINT8    abHostName[64+2];
  TLR_UINT8    bSelectAcd;
  EIP DPMINTF TI ACD LAST CONFLICT T  tLastConflictDetected;
  TLR UINT8    bQuickConnectFlags;
  TLR UINT8    abAdminState[2]
  TLR UINT8    abReserved[9];
  TLR_UINT16   usEncapInactivityTimeout;
} EIP APS CONFIGURATION PARAMETER SET V3 T;

typedef struct EIP APS SET CONFIGURATION PARAMETERS REQ Ttag
{
  TLR UINT32  ulParameterVersion;  /*!< Version related to the following configuration union */

  union
  {
    EIP APS CONFIGURATION PARAMETER SET V1 T tV1;
    EIP APS CONFIGURATION PARAMETER SET V2 T tV2;
    EIP_APS_CONFIGURATION_PARAMETER_SET_V2_T tV3;

  } unConfig;

} EIP APS SET CONFIGURATION PARAMETERS REQ T;

typedef struct EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ_Ttag
{
  TLR_PACKET_HEADER_T                      tHead;
```

```
  EIP APS SET CONFIGURATION PARAMETERS REQ T tData;
}EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ_T;
```

Packet Description

structure EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue-Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	282	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x3612	EIP_APS_SET_CONFIGURATION_PARAMETERS_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_APS_SET_CONFIGURATION_ PARAMETERS_REQ_T			
ulParameterVersion	UINT32	3 (latest version)	Version of the following parameter structure
unConfig.tV3	UNION		For parameter set version 2 the structure in Table 59 must be used.

Table 58: EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ – Set Configuration Parameters Request

Structure EIP_APS_CONFIGURATION_PARAMETER_SET_V3_T			
ulSystemFlags	UINT32 (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 (see section 3.2.5.1 of the netX DPM Interface Manual). Communication with controller is allowed only with the BUS_ON flag. For more information concerning this topic see section 4.4.1 “Controlled or Automatic Start” of the netX DPM Interface Manual.
ulWdgTime	UINT32	0, 20..65535	Watchdog time (in milliseconds). 0 = Watchdog timer has been switched off Default value: 1000
ulInputLen	UINT32	0..504	Length of Input data (O→T direction, data the device receives from a Scanner)
ulOutputLen	UINT32	0..504	Length of Output data (T→O direction, data the device sends to a Scanner)
ulTcpFlag	UINT32	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 63 on page 101. Default value: 0x00000410 (both ports set to DHCP + Autoneg)
ulIPAddr	UINT32	All valid IP-addresses	IP Address See detailed explanation in the corresponding TCP/IP Manual (reference [3])
ulNetMask	UINT32	All valid masks	Network Mask See detailed explanation in the corresponding TCP/IP Manual (reference [3])
ulGateway	UINT32	All valid IP-addresses	Gateway Address See detailed explanation in the corresponding TCP/IP Manual (reference [3])
usVendorID	UINT16	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 value zero is not valid. Default value: 283 (Hilscher)

usProductType	UINT16	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) The value 0 is not a valid Product Type. However, when using value 0 here, the stack automatically chooses the default Product Type (0x0C).
usProductCode	UINT16	0..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. The value 0 is not a valid Product Code. However, when using value 0 here, the stack automatically chooses the default Product Code dependent on the chip type (netX50/100 etc.) that is used.
ulSerialNumber	UINT32	0..0xFFFFFFFF	Serial Number of the device This parameter is a number used in conjunction with the Vendor ID to form 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. Usually, this number will be set automatically by the firmware, if a security memory is available. In this case leave this parameter at value 0.
bMinorRev	UINT8	1..255	Major revision Value 0 is not a valid major revision number. If major revision and minor revision both are set to 0, the stack uses the default value predefined in the firmware.
bMajorRev	UINT8	1..127	Minor revision Value 0 is not a valid minor revision number. If major revision and minor revision both are set to 0, the stack uses the default value predefined in the firmware.

abDeviceName	UINT8[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” Note: If an empty device name (“”) is configured, the firmware will use the default device name. For an overview of default names see Table 60.
ulInputAssInstance	UINT32	0 (no input assembly) 1- 0xFFFFFFFF	Instance number of input assembly (O→T direction) See Table 98 “Assembly Instance Number Ranges” Default: 100 **Note**: If instance number is != 0: the value of ulInputAssInstance must differ from the value of ulOutputAssInstance.
ulInputAssFlags	UINT32	Bit mask	Input assembly (O→T) flags See Table 64 “Input Assembly Flags/ Output Assembly Flags”
ulOutputwAssInstance	UINT32	0 (no output assembly) 1- 0xFFFFFFFF	Instance number of output assembly (T→O direction) See Table 98 “Assembly Instance Number Ranges” Default: 101 **Note**: If instance number is != 0: the value of ulInputAssInstance must differ from the value of ulOutputAssInstance.
ulOutputAssFlags	UINT32	Bit mask	Output assembly (T→O) flags See Table 64 “Input Assembly Flags/ Output 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) For a detailed description of the parameters see command EIP_OBJECT_CFG_QOS_REQ/CNF – Configure the QoS Object
ulNameServer	UINT32	See section 3.6	Name Server 1 This parameter configures the NameServer element of attribute 5 of the TCP/IP Interface Object. See section 3.6 “TCP/IP Interface Object (Class Code: 0xF5) for more information. Default: 0.0.0.0
ulNameServer_2	UINT32	See section 3.6	Name Server 2 This parameter configures the NameServer2 element of attribute 5 of the TCP/IP Interface Object. See section 3.6 “TCP/IP Interface Object (Class Code: 0xF5) for more information.
abDomainName[48 + 2]	UINT8[]	See section 3.6	Domain Name This parameter configures the DomainName element of attribute 5 of the TCP/IP Interface Object. See section 3.6 “TCP/IP Interface Object (Class Code: 0xF5) for more information.

abHostName[64+2]	UINT8[]	See section 3.6	Host Name This parameter configures attribute 6 of the TCP/IP Interface Object. See section 3.6 “TCP/IP Interface Object (Class Code: 0xF5) for more information.
bSelectAcd	UINT8	See section 3.6	Select ACD This parameter configures attribute 10 of the TCP/IP Interface Object. See section 3.6 “TCP/IP Interface Object (Class Code: 0xF5) for more information.
tLastConflictDetected	EIP_DPMINTF_TI_ACD_LAST_CONFLICT_T	See section 3.6	Last Detected Conflict This parameter configures attribute 11 of the TCP/IP Interface Object. See section 3.6 “TCP/IP Interface Object (Class Code: 0xF5) for more information.
bQuickConnectFlags	UINT8	0,1,3	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 3.6 “TCP/IP Interface Object (Class Code: 0xF5) for more information. Bit 0 (EIP_OBJECT_QC_FLAGS_ACTIVATE_ATTRIBUTE): If set (1), the Quick Connect Attribute 12 of the TCP Interface Object (0xF5) is activated (i.e. it is present and accessible via CIP services). The actual value of attribute 12 can be configured with 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	1, 2	Admin State This parameter configures attribute 9 of the Ethernet Link Object. See section 3.7 “Ethernet Link Object (Class Code: 0xF6)“ for more information.
abReserved[9]	UINT8	0	Set to zero
usEncapInactivityTimeout	UINT16	0-3600	This parameter corresponds to attribute 13 of the TCP/IP Interface Object (CIP Id 0xF5). The Encapsulation Inactivity Timeout is used to close sockets when the defined time (seconds) elapsed without Encapsulation activity. This attribute shall be stored in non-volatile memory.

Table 59: EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_REQ – Configuration Parameter Set V3

The following table gives an overview of the default device names depending on the used loadable firmware:

Loadable firmware	Default device name
COMX 100	"COMX 100XX-RE/EIS"
COMX 51	"COMX 51XX-RE/EIS"
netJACK 51	"NJ 51X-RE/EIS"
netJACK 50	"NJ 50X-RE/EIS"
netJACK 100	"NJ 100X-RE/EIS"
netX 100	"NETX 100 RE/EIS"

netX 51	"NETX 51 RE/EIS"
netX 50	"NETX 50 RE/EIS"
netX 500	"NETX 500 RE/EIS"
CIFX	"CIFX RE/EIS"

Table 60: Default device name for loadable firmwares

The following flags are available in the area ulTcpFlag:

In general, this 32 bit area can be devided into two 16-bit areas, the lower area (bits 15 – 0, Table 61) and the upper area (bits 31 – 16, Table 62).

The upper area gets active only if the Extended Flag (bit 15) is set to 1. Setting flags in the upper area without setting the Extended Flag will not have any effect.

Table 63 describes in more detail what the corresponding flags are used for.

ulTcpFlag - Lower 16 bit														
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Extended Flag	Reserved		Speed Selection	Duplex Operation	Auto-Negotiation	Reserved					DHCP	BOOTP	Gateway Address	Net Mask	IP Address

Table 61: Definition of area `ulTcpFlag` (Lower 16 bit)

ulTcpFlag - Upper 16 bit															
31	**30**	**29**	**28**	**27**	**26**	**25**	**24**	**23**	**22**	**21**	**20**	**19**	**18**	**17**	**16**
Reserved			Speed Selection, Port 1	Duplex Operation, Port 1	Auto-Negotiation, Port 1	Reserved						MDI Mode, Port 1		MDI Mode, Port 0	

Table 62: Definition of area `ulTcpFlag` (Upper 16 bit)

Bits	Description
31.. 29	Reserved
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..20	Reserved
19..18	MDI mode for Port 1 0: use default (Auto MDI-X except for Quick Connect) 1: Auto MDI-X 2: MDI 3: MDI-X

Bits	Description
17..16	MDI mode for Port 0 0: use default (Auto MDI-X except for Quick Connect) 1: Auto MDI-X 2: MDI 3: MDI-X
15	Extended Flag: This flag can be used if the device has two Ethernet ports. In that case the two ports can be configured individually regarding “Speed Selection”, “Duplex Operation”, “Auto-Negotiation” and “MDI Mode” If not set (0), both ports are configured with the same parameters using the bits 10 to 12. In that case the default MDI mode “Auto MDI-X” is used. If set (1), port 1 is configured using bits 10 to 12. Port 2 is configured using the bits 26 to 28. In addition, the MDI mode can be configured for both ports individually using the bits 16-17 (port 0) and 18-19 (port 1)
13..14	Reserved
12	Speed Selection: (Ethernet Port 1) If set (1), the device will operate at 100 MBit/s, otherwise at 10 MBit/s. This parameter will only be evaluated, 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 This parameter will only be evaluated, 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
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: If set (1), the content of the `ulGateway` parameter will be evaluated. If the flag is not set (0), `ulGateway` must be set to 0.0.0.0.
1	Netmask: If set (1), the content of the `ulNetMask` parameter will be evaluated. 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: If set (1), the content of the `ulIpAddr` parameter will be evaluated. In this case the parameter `ulNetMask` must be a valid net mask.

Table 63: Description of available flags for the area `ulTcpFlag`

The input assembly flags and the output assembly flags are defined as follows:

Flag	Meaning
Bit 0	This flag is used internally and must be set to 0.
Bit 1	This flag is used internally and must be set to 0.
Bit 2	This flag is used internally and must be set to 0.
Bit 3	If set (1), the assembly instance's real time format is modeless, i.e. it does **not** contain run/idle information. If not set (0), the assembly instance's real time format is the 32-Bit Run/Idle header. For more information about real time format see section 2.4.3.1 "Real Time Format".
Bit 4	This flag is used internally and must be set to 0
Bit 5	This flag is used internally and must be set to 0
Bit 6	This flag decides whether the assembly data which is mapped into the DPM memory area is cleared upon closing or timeout of the connection or whether the last sent/received data is left unchanged in the memory. If the bit is set (1) the data will be left unchanged.

Bit 7	This flag decides whether the assembly instance allows a connection to be established with a smaller connection size than defined in `ulInputLen/ulOutputLen` or whether only the exact match is accepted. If the bit is set (1), the connection size in a ForwardOpen must directly correspond to `ulInputLen/ulOutputLen`.
	Example:
	1) `ulInputLen = 16 (Bit 7 of ulInputAssFlags is not set (0))`
	`ulOutputLen = 32 (Bit 7 of ulOutputAssFlags is not set (0))`
	`A connection can be opened with smaller or matching I/O sizes, e.g. 8 for input and 20 for output.`
	2) `ulInputLen = 6 (Bit 7 of ulInputAssFlags is set (1))`
	`ulOutputLen = 10 (Bit 7 of ulOutputAssFlags is set (1))`
	`A connection can only be opened with matching I/O sizes, 6 for input and 10 for output.`

Table 64: Input Assembly Flags/ Output Assembly Flags

Packet Structure Reference

```
typedef   PACKED PRE struct EIP APS SET CONFIGURATION PARAMETERS CNF Ttag
{
  TLR_UINT32  ulPacketVersion;  /*!< Version related to the following union entry */

  __PACKED_PRE union
  {
    EIP APS CONFIGURATION PARAMETER SET V1 T tV1;
    EIP APS CONFIGURATION PARAMETER SET V2 T tV2;
    EIP_APS_CONFIGURATION_PARAMETER_SET_V2_T tV3;

  }__PACKED_POST unConfig;

}  PACKED POST  EIP APS SET CONFIGURATION PARAMETERS CNF T;


typedef struct EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_CNF_Ttag
{
  TLR PACKET HEADER T                       tHead;
  EIP APS SET CONFIGURATION PARAMETERS CNF T tData;
} EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_CNF_T;
```

Packet Description

Structure EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination Queue Reference
ulSrcId	UINT32	See rules in section 3.2.1	Source Queue Reference
ulLen	UINT32	size from request packet	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x3613	EIP_APS_SET_CONFIGURATION_PARAMETERS_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_APS_SET_CONFIGURATION_ PARAMETERS_CNF_T			
ulParameterVersion	UINT32		Version of the following parameter structure (from request packet)
unConfig	UNION		Configuration Set (from request packet)

Table 65: EIP_APS_PACKET_SET_CONFIGURATION_PARAMETERS_CNF – Set Configuration Parameters Confirmation

6.1.2 `EIP_APS_CLEAR_WATCHDOG_REQ/CNF` – Clear Watchdog error

This packet can be sent by the host application task to the AP-Task in order to clear a watchdog error. Figure 16 below displays a sequence diagram for the `EIP_APS_CLEAR_WATCHDOG_REQ/CNF` packet:

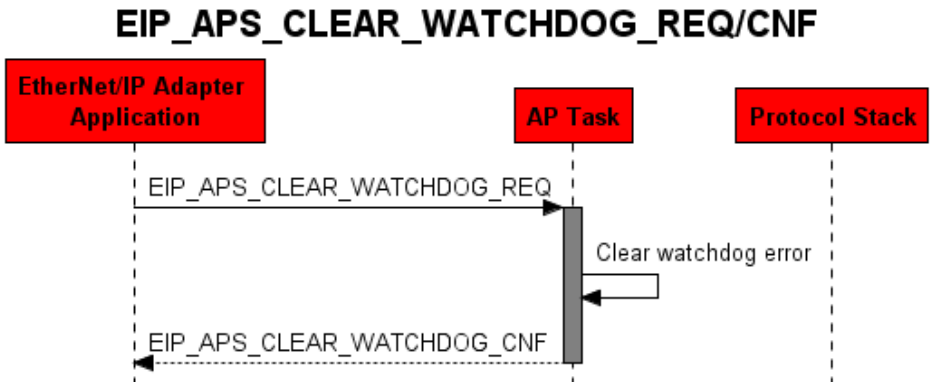

Figure 16: Sequence Diagram for the `EIP_APS_CLEAR_WATCHDOG_REQ/CNF` Packet

Packet Structure Reference

```
typedef struct  EIP_APS_PACKET_CLEAR_WATCHDOG_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
} EIP_APS_PACKET_CLEAR_WATCHDOG_REQ_T;

#define EIP_APS_CLEAR_WATCHDOG_REQ_SIZE 0
```

Packet Description

Structure Information EIP_APS_PACKET_CLEAR_WATCHDOG_REQ_T				
Type: Request				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	Structure Information			
	ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
	ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
	ulLen	UINT32	0	EIP_APS_CLEAR_WATCHDOG_REQ_SIZE Packet Data Length (In Bytes)
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
	ulSta	UINT32		See Packet Structure Reference
	ulCmd	UINT32	0x3602	EIP_APS_CLEAR_WATCHDOG_REQ - Command / Response
	ulExt	UINT32		Reserved
	ulRout	UINT32		Routing Information

Table 66: EIP_APS_CLEAR_WATCHDOG_REQ – Request to clear watchdog error

Packet Structure Reference

```
typedef struct  EIP APS PACKET CLEAR WATCHDOG CNF Ttag {
  TLR_PACKET_HEADER_T tHead;
} EIP APS PACKET CLEAR WATCHDOG CNF T;

#define EIP_APS_CLEAR_WATCHDOG_CNF_SIZE 0
```

Packet Description

Structure Information `EIP_APS_PACKET_CLEAR_WATCHDOG_CNF_T`				
Type: Confirmation				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	Structure Information			
	`ulDest`	UINT32	See rules in section 3.2.1	Destination Queue Handle
	`ulSrc`	UINT32	See rules in section 3.2.1	Source Queue Handle
	`ulDestId`	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
	`ulSrcId`	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	`ulLen`	UINT32	0	`EIP_APS_CLEAR_WATCHDOG_CNF_SIZE` Packet Data Length (In Bytes)
	`ulId`	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
	`ulSta`	UINT32		See chapter *Status/Error Codes Overview*
	`ulCmd`	UINT32	`0x00003603`	`EIP_APS_CLEAR_WATCHDOG_CNF` - Command / Response
	`ulExt`	UINT32		Reserved
	`ulRout`	UINT32		Routing Information

Table 67: `EIP_APS_CLEAR_WATCHDOG_CNF` – Confirmation to clear watchdog request

6.1.3 EIP_APS_SET_PARAMETER_REQ/CNF – Set Parameter Flags

This packet can be sent by the host application to activate special functionalities or behaviors of the AP-Task. The request packet therefore contains a flag field in which each bit stands for a specific functionality.

Table 68 shows all available flags:

Bit	Description
0	Flag IP_APS_PRM_SIGNAL_MS_NS_CHANGE (0x00000001) If set (1), the host application will be notified whenever the network or module status changes. The module and the network status are displayed by LEDs at EtherNet/IP devices (see section 9.1 "*Module and Network Status*" for more information). The notification will be sent with the indication packet 6.1.4EIP_APS_MS_NS_CHANGE_IND/RES – Module Status/ Network Status Change Indication. If not set (0) no notifications will be sent.
1..31	Reserved for future use.

Table 68: EIP_APS_SET_PARAMETER_REQ Flags

Figure 17 below displays a sequence diagram for the EIP_APS_SET_PARAMETER_REQ/CNF packet.

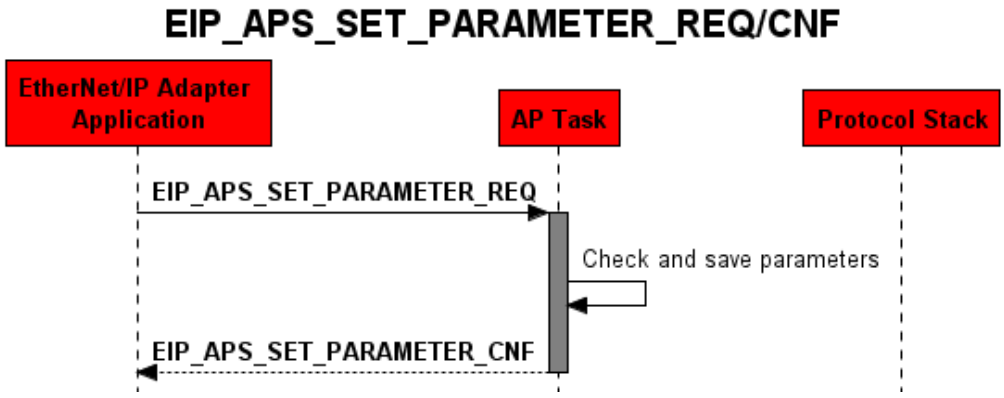

Figure 17: Sequence diagram for the EIP_APS_SET_PARAMETER_REQ/CNF packet

Packet Structure Reference

```
#define EIP APS PRM SIGNAL MS NS CHANGE           0x00000001

typedef struct EIP APS SET PARAMETER REQ Ttag
{
  TLR UINT32  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 struct EIP APS PACKET SET PARAMETER REQ Ttag
{
  TLR PACKET HEADER T             tHead;
  EIP APS SET PARAMETER REQ T     tData;
} EIP_APS_PACKET_SET_PARAMETER_REQ_T;
```

Packet Description

structure EIP_APS_PACKET_SET_PARAMETER_REQ_T				
Type: Request				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue-Handle
	ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
	ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See Packet Structure Reference
	ulCmd	UINT32	0x360A	EIP_APS_SET_PARAMETER_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure **EIP_APS_SET_PARAMETER_REQ_T**			
	ulParameterFlags	UINT32	See Table 68 for possible values	Bit field

Table 69: EIP_APS_SET_PARAMETER_REQ – Set Parameter Flags Request

Packet Structure Reference

```
#define EIP APS SET PARAMETER CNF SIZE 0

typedef struct EIP APS PACKET SET PARAMETER CNF Ttag
{
  TLR PACKET HEADER T tHead;
} EIP_APS_PACKET_SET_PARAMETER_CNF_T;
```

Packet Description

structure EIP_APS_PACKET_SET_PARAMETER_CNF_T				
Type: Confirmation				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure `TLR_PACKET_HEADER_T`			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See Packet Structure Reference
	ulCmd	UINT32	0x360B	`EIP_APS_SET_PARAMETER_CNF` - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 70: `EIP_APS_SET_PARAMETER_CNF` – Confirmation to Set Parameter Flags Request

6.1.4 EIP_APS_MS_NS_CHANGE_IND/RES – Module Status/ Network Status Change Indication

This packet indicates a change in either the module or network status. Both module status and network status are displayed at the device by LEDs.

Note: This functionality must be enabled in advance by setting the flag EIP_APS_PRM_SIGNAL_MS_NS_CHANGE using the packet EIP_APS_SET_PARAMETER_REQ/CNF – Set Parameter Flags (section 6.1.3).

Figure 18 below displays a sequence diagram for the EIP_APS_MS_NS_CHANGE_IND/RES packet:

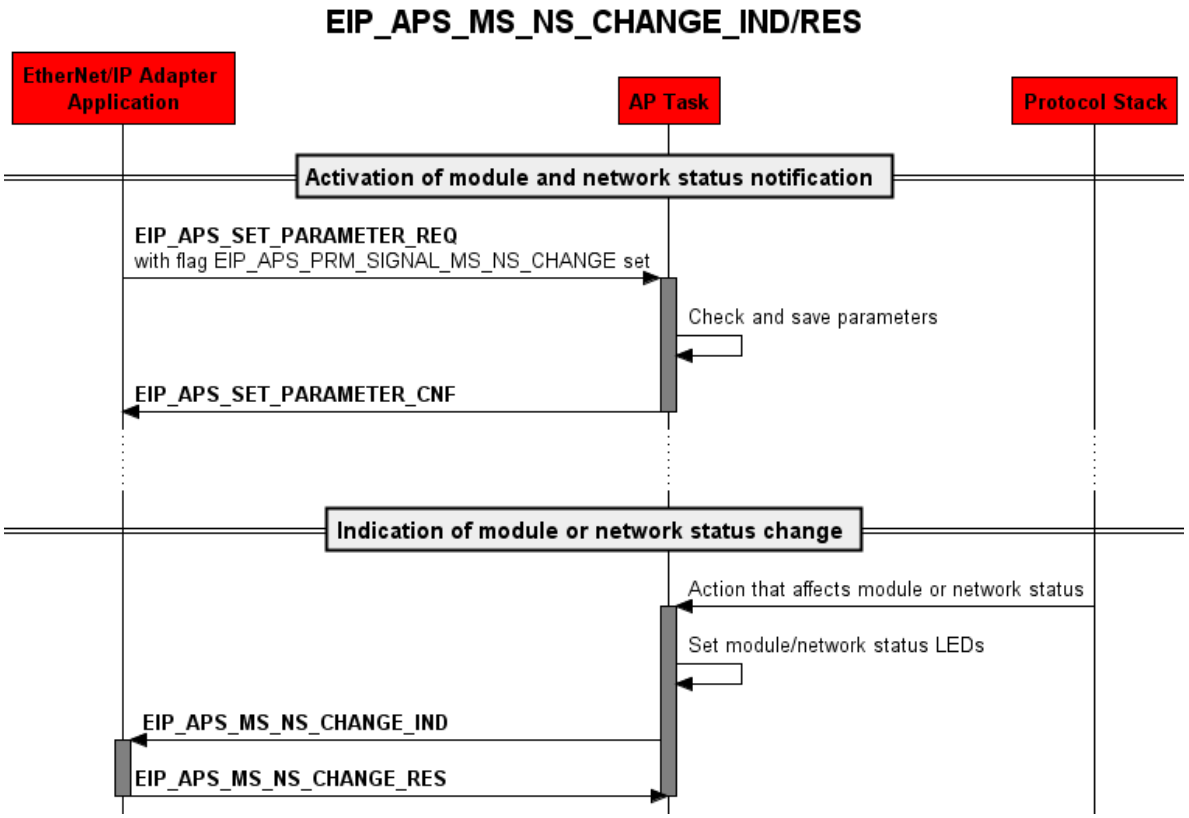

Figure 18: Sequence Diagram for the EIP_APS_MS_NS_CHANGE_IND/RES Packet

Packet Structure Reference

```
typedef struct EIP APS MS NS CHANGE IND Ttag
{
  TLR UINT32  ulModuleStatus;     /*!< Module Status \n
  TLR UINT32  ulNetworkStatus;     /*!< Network Status \n
} EIP APS MS NS CHANGE IND T;

#define EIP APS MS NS CHANGE IND SIZE (sizeof(EIP APS MS NS CHANGE IND T))

typedef struct EIP APS PACKET MS NS CHANGE IND Ttag
{
  TLR PACKET HEADER T             tHead;
  EIP APS MS NS CHANGE IND T      tData;
} EIP_APS_PACKET_MS_NS_CHANGE_IND_T;
```

Packet Description

structure EIP_APS_PACKET_MS_NS_CHANGE_IND_T				
Type: Indication				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure `TLR_PACKET_HEADER_T`			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
	ulLen	UINT32	8	Packet Data Length in bytes
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See Packet Structure Reference
	ulCmd	UINT32	0x360C	`EIP_APS_MS_NS_CHANGE_IND` - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure EIP_APS_MS_NS_CHANGE_IND_T			
	ulModuleStatus	UINT32	0…5	Module Status The module status describes the current state of the corresponding MS-LED (provided that it is connected). See Table 158 for more information.
	ulNetworkStatus	UINT32	0…5	Network Status The network status describes the current state of the corresponding NS-LED (provided that it is connected). See Table 159 for more information.

Table 71: `EIP_APS_MS_NS_CHANGE_IND` – Module Status/ Network Status Change Indication

Packet Structure Reference

```
#define EIP APS MS NS CHANGE RES SIZE 0

typedef struct EIP APS PACKET MS NS CHANGE RES Ttag
{
  TLR PACKET HEADER T             tHead;
} EIP_APS_PACKET_MS_NS_CHANGE_RES_T;
```

Packet Description

structure EIP_APS_PACKET_MS_NS_CHANGE_RES_T				
Type: Response				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
	ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See Packet Structure Reference
	ulCmd	UINT32	0x360D	EIP_APS_MS_NS_CHANGE_RES - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 72: EIP_APS_MS_NS_CHANGE_RES – Response to Module Status/ Network Status Change Indication

6.1.5 EIP_APS_GET_MS_NS_REQ/CNF – Get Module Status/Network Status

This packet can be used by the EtherNet/IP Adapter Application in order to obtain information about the current module and network status for further evaluation.

Table 158 on page 241 lists all possible values of the Module Status (Parameter `ulModuleStatus` of the confirmation packet) and their meanings.

Similarly, Table 159 on page 242 lists all possible values of the Network Status (Parameter `ulNetworkStatus` of the confirmation packet) and their meanings.

Figure 19 below displays a sequence diagram for the `EIP_APS_GET_MS_NS_REQ/CNF` packet:

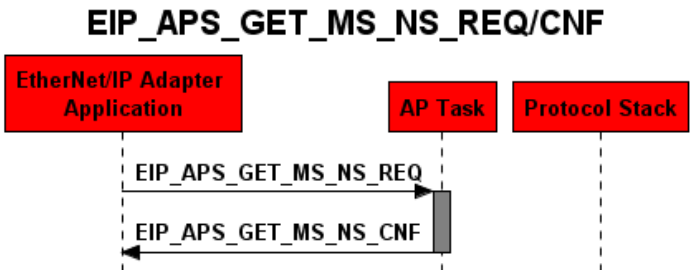

Figure 19: Sequence Diagram for the `EIP_APS_GET_MS_NS_REQ/CNF` Packet

Packet Structure Reference

```
#define EIP APS GET MS NS REQ SIZE       0

typedef struct EIP_APS_PACKET_GET_MS_NS_REQ_Ttag {
  TLR_PACKET_HEADER_T             tHead;
} EIP_APS_PACKET_GET_MS_NS_REQ_T;
```

Packet Description

structure EIP_APS_PACKET_GET_MS_NS_REQ_T				
Type: Request				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure `TLR_PACKET_HEADER_T`			
	ulDest	UINT32	`0x20/ DPMINTF_QUE`	Destination Queue-Handle
	ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
	ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See Packet Structure Reference
	ulCmd	UINT32	0x360E	`EIP_APS_GET_MS_NS_REQ` - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 73: `EIP_APS_GET_MS_NS_REQ` – Get Module Status/ Network Status Request

Packet Structure Reference

```
typedef struct EIP APS GET MS NS CNF Ttag
{
  TLR UINT32  ulModuleStatus;      /*!< Module Status \n
  TLR UINT32  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 struct EIP APS PACKET GET MS NS CNF Ttag
{
  TLR PACKET HEADER T      tHead;
  EIP APS GET MS NS CNF T  tData;

} EIP_APS_PACKET_GET_MS_NS_CNF_T;
```

Packet Description

structure EIP_APS_PACKET_GET_MS_NS_CNF_T				
Type: Confirmation				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure `TLR_PACKET_HEADER_T`			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32		Packet Data Length in bytes
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See Packet Structure Reference
	ulCmd	UINT32	0x360F	`EIP_APS_GET_MS_NS_CNF` - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure **EIP_APS_GET_MS_NS_CNF_T**			
	ulModuleStatus	UINT32	0..5	Module Status The module status describes the current state of the corresponding MS-LED (provided that it is connected). See Table 158 for more information.
	ulNetworkStatus	UINT32	0..5	Network Status The network status describes the current state of the corresponding NS-LED (provided that it is connected). See Table 159 for more information.

Table 74: `EIP_APS_GET_MS_NS_CNF` – Confirmation of Get Module Status/ Network Status Request

6.1.6 EIP_APS_SET_MODULE_STATUS_REQ/CNF – Set Module Status

The application can use this packet to set the current module status of the device. This also implicitly controls the module status LED of the device.

Table 158 on page 241 lists all possible values of the Module Status and their meaning.

By default the EtherNet/IP firmware does not support this service. The application has to activate this service in advance before using it. If not activated this service will be conformed with status 0xC0000002 (TLR_E_UNEXPECTED).

To activate this service, the application has to use the service EIP_OBJECT_SET_PARAMETER_REQ and set the flag EIP_OBJECT_PRM_APPLICATION_CONTROLS_IDENTITY_STATE_ATTRIBUTE.

Note: Using this service also means that the host application must take care of Identity object's attribute 8 (state). See section *EIP_OBJECT_SET_PARAMETER_REQ/CNF – Set Parameter* on page 181 for more information.

Packet Description

structure EIP_APS_PACKET_SET_MODULE_STATUS_REQ_T				
Type: Request				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue-Handle
	ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
	ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See Packet Structure Reference
	ulCmd	UINT32	0x3616	EIP_APS_SET_MODULE_STATUS_REQ - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure EIP_APS_SET_MODULE_STATUS_T			
	ulModuleStatus	UINT32	0..5	Module Status The module status describes the current state of the corresponding MS-LED (provided that it is physically connected). See Table 158 for more information.

Table 75: EIP_APS_SET_MODULE_STATUS_REQ – Set the Module Status

Packet Description

structure EIP_APS_PACKET_SET_MODULE_STATUS_CNF_T				
Type: Confirmation				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	4	Packet Data Length in bytes
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See Packet Structure Reference
	ulCmd	UINT32	0x3617	EIP_APS_SET_MODULE_STATUS_CNF - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	**structure** EIP_APS_SET_MODULE_STATUS_T			
	ulModuleStatus	UINT32	0..5	Module Status The module status describes the current state of the corresponding MS-LED (provided that it is connected). See Table 158 for more information.

Table 76: EIP_APS_GET_MS_NS_CNF – Confirmation of Get Module Status/ Network Status Request

6.1.7 Modify Configuration Parameters

The modifying configuration parameter function allows selectively changing configuration parameters of the EtherNet/IP Adapter protocol stack that has already been configured by SYCON.net or by netX Configuration Tool. Modifying a parameter after the device was configured by a tool requires that the start-up bebavoir is set to 'Controlled start of communication'.

The EtherNet/IP Adapter stack supports the following parameters to be modified:

ParameterID	Name	Type	Description
PID_EIP_IP_CONFIGURATION (0x3000A001)	ulIP	UINT32	IP address
	ulNetmask	UINT32	Network mask
	ulGateway	UINT32	Gateway address
PID_EIP_IP_CONFIGCONTROL (0x3000A002)	ulConfiguration Control	UINT32	PRM_CFGCTRL_STORED_CFG 0 PRM_CFGCTRL_DHCP 1 PRM_CFGCTRL_BOOTP 2 PRM_CFGCTRL_FIXIP 3

Table 77 RCX_SET_FW_PARAMETER_REQ ParameterID

Section *Modify Configuration Settings* in reference [2] describes the Set Parameter Request packet.

6.2 The EIS_OBJECT – Task

In detail, the following functionality is provided by the EIS_OBJECT -Task:

Overview over Packets of the EIS_OBJECT – Task			
No. of section	**Packet**	**Command code (REQ/CNF or IND/RES)**	**Page**
6.2.1	EIP_OBJECT_FAULT_IND/RES – Fault Indication	0x1A30/ 0x1A31	119
6.2.2	EIP_OBJECT_CONNECTION_IND/RES – Connection State Change Indication	0x1A2E/ 0x1A2F	122
6.2.3	EIP_OBJECT_MR_REGISTER_REQ/CNF – Register an additional Object Class at the Message Router	0x1A02/ 0x1A03	130
6.2.4	EIP_OBJECT_CL3_SERVICE_IND/RES - Indication of acyclic Data Transfer	0x1A3E/ 0x1A3F	134
6.2.5	EIP_OBJECT_AS_REGISTER_REQ/CNF – Register a new Assembly Instance	0x1A0C/ 0x1A0D	141
6.2.6	EIP_OBJECT_ID_SETDEVICEINFO_REQ/CNF – Set the Device's Identity Information	0x1A16/ 0x1A17	148
6.2.7	EIP_OBJECT_GET_INPUT_REQ/CNF – Getting the latest Input Data	0x1A20/ 0x1A21	154
6.2.8	EIP_OBJECT_RESET_IND/RES – Indication of a Reset Request from	0x1A24/ 0x1A25	157
6.2.9	EIP_OBJECT_RESET_REQ/CNF - Reset Request	0x1A26/ 0x1A27	162
6.2.10	EIP_OBJECT_READY_REQ/CNF – Set Ready and Run/Idle State	0x1A32/ 0x1A33	165
6.2.11	EIP_OBJECT_REGISTER_SERVICE_REQ/CNF – Register Service	0x1A44/ 0x1A45	168
6.2.12	EIP_OBJECT_CONNECTION_CONFIG_IND/RES – Indication of Configuration Data received during Connection Establishment	0x1A40/ 0x1A41	171
6.2.13	EIP_OBJECT_TI_SET_SNN_REQ/CNF – Set the Safety Network Number for the TCP/IP Interface Object	0x1AF0/ 0x1AF1	178
6.2.14	EIP_OBJECT_SET_PARAMETER_REQ/CNF – Set Parameter	0x1AF2/ 0x1AF3	181
6.2.16	EIP_OBJECT_CFG_QOS_REQ/CNF – Configure the QoS Object	0x1A42/ 0x1A43	192
6.2.17	EIP_OBJECT_CIP_SERVICE_REQ/CNF – CIP Service Request	0x1AF8/ 0x1AF9	196
6.2.18	EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES – CIP Object Change Indication	0x1AFA/ 0x1AFB	201

Table 78: Overview over Packets of the EIS_OBJECT -Task of the EtherNet/IP-Adapter Protocol Stack

6.2.1 EIP_OBJECT_FAULT_IND/RES – Fault Indication

This indication packet is sent from the EtherNet/IP Adapter protocol stack to the user application in order to indicate a fault within the EtherNet/IP protocol stack. The error is reported in the ulSta field of the packet header. This indication is for informational purpose only. There is no action required on the host application side, except sending the response packet.

Figure 20 and Figure 21 below display a sequence diagram for the EIP_OBJECT_FAULT_IND/RES packet in case the host application uses the Basic, Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”):

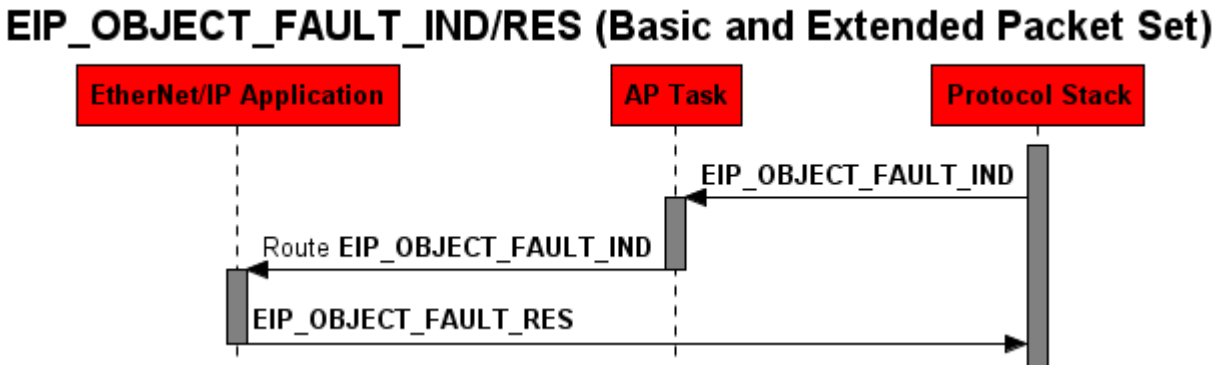

Figure 20: Sequence Diagram for the EIP_OBJECT_FAULT_IND/RES Packet for the Basic and Extended Packet Set

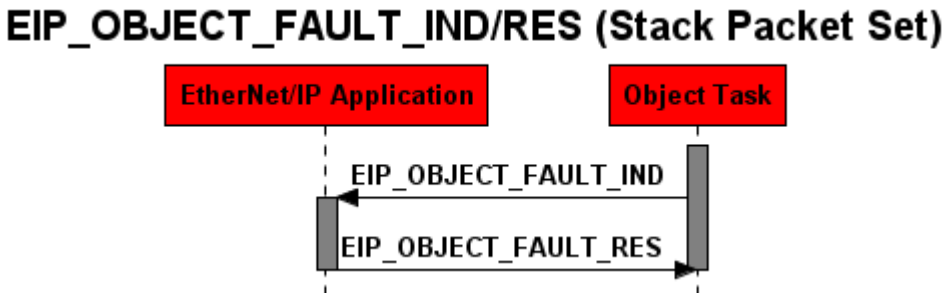

Figure 21: Sequence Diagram for the EIP_OBJECT_FAULT_IND/RES Packet for the Stack Packet Set

Packet Structure Reference

```
typedef struct EIP OBJECT PACKET FAULT IND Ttag
{
  TLR_PACKET_HEADER_T     tHead;
} EIP_OBJECT_PACKET_FAULT_IND_T;

#define EIP_OBJECT_FAULT_IND_SIZE 0
```

Packet Description

Structure EIP_OBJECT_PACKET_FAULT_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with linkable object modules. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	0	EIP_OBJECT_FAULT_IND – Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1A30	EIP_OBJECT_FAULT_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	×	Routing, do not change

Table 79: EIP_OBJECT_FAULT_IND – Indication Packet of a Fault

Packet Structure Reference

```
typedef struct EIP OBJECT PACKET FAULT RES Ttag
{
  TLR PACKET HEADER T     tHead;
}EIP OBJECT PACKET FAULT RES T;

#define EIP_OBJECT_FAULT_RES_SIZE 0
```

Packet Description

Structure EIP_OBJECT_PACKET_FAULT_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	EIP_OBJECT_FAULT_RES – Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32	0	See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A31	EIP_OBJECT_FAULT_RES - Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information

Table 80: EIP_OBJECT_FAULT_RES – Response to Indication Packet of a fatal Fault

6.2.2 EIP_OBJECT_CONNECTION_IND/RES – Connection State Change Indication

This indication will be sent to the application task every time a CIP connection is established, closed or has timed out. This applies to Exclusive Owner, Input Only, Listen Only and Explicit connections.

Connection State - ulConnectionState

The variable ulConnectionState indicates whether a connection has been established or closed.

ulConnectionState =	Numeric Value	Meaning
EIP_CONNECTED	1	Connection has been established
EIP_UNCONNECT	2	Connection was closed. If connection timed out, the value of ulExtendedState will be 1, otherwise 0.

Table 81: Meaning of variable ulConnectionState

Extended Connection State - ulExtendedState

The variable ulExtendedState (only valid if ulConnectionState is EIP_UNCONNECT (0)) contains information about the extended connection state according to the following table:

ulExtendedState =	Numeric Value	Meaning
EIP_CONN_STATE_UNDEFINED	0	Undefined, not used
EIP_CONN_STATE_TIMEOUT	1	Connection timed out

Table 82: Meaning of variable ulExtendedState

Connection Info - tConnection

For the EtherNet/IP adapter only the union entry tTOConnection is important:

ulClass: Class to which the connection was directed

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

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

ulInstance: Corresponding instance of the class provided in ulClass

If ulClass is 0x04, ulInstance is the configuration assembly instance.
If ulClass is 0x02, ulInstance is always 1.

ulOTConnPoint: Input connection point (Only valid if ulClass == 0x04)

Provides the connection point (assembly instance) in O→T direction.

ulTOConnPoint: Output connection point (Only valid if ulClass == 0x04)

Provides the connection point (assembly instance) in T→O direction.

`ulConnectionType`: Provides the connection application type

`ulConnectionType` defines	**Value**	**Meaning**
EIP_CONN_TYPE_UNDEFINED	0	No connection type available
EIP_CONN_TYPE_CLASS_0_1_EXCLUSIVE_OWNER	1	(Implicit) Exclusive owner connection
EIP_CONN_TYPE_CLASS_0_1_REDUNDANT_OWNER	2	(Implicit) Redundant owner connection (not supported)
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) Class 3 connection

Table 83: ulConnectionType - Enum

Extended Connection Info - `tExtInfo`

`tExtInfo` contains a structure of type `EIP_OBJECT_EXT_CONNECTION_INFO_T` providing additional information concerning incoming connections having been established. This structure has the following elements:

`tExtInfo` Element	**Type**	**Meaning**
`ulProConnId`	TLR_UINT32	Producer Connection ID (T→O)
`ulConConnId`	TLR_UINT32	Consumer Connection ID (O→T)
`ulConnSerialNum`	TLR_UINT32	Connection serial number
`usOrigVendorId`	TLR_UINT16	Originator device vendor ID
`ulOrigDeviceSn`	TLR_UINT32	Originator device serial number
`ulProApi`	TLR_UINT32	Actual packet interval (specified in microseconds) (T→O)
`usProConnParams`	TLR_UINT16	Connection parameters (T→O) from ForwardOpen
`ulConApi`	TLR_UINT32	Actual packet interval (specified in microseconds) (O→T)
`usConConnParams`	TLR_UINT16	Connection parameters (O→T) from ForwardOpen
`bTimeoutMultiplier`	TLR_UINT8	Connection timeout multiplier

Table 84: Structure `tExtInfo`

`ulProConnID` contains the Connection ID for the Producer Connection (i.e. from target to originator).

`ulConConnID` contains the Connection ID for the Consumer Connection (i.e. from originator to target).

`ulConnSerialNum` contains the serial number of the connection. This must be a unique 16-bit value. For more details, see *"The CIP Networks Library, Volume 1"*, section 3-5.5.1.5.

`usOrigVendorId` contains the Vendor ID of the connection originator (i.e. the contents of attribute #1 of instance #1 of the connection originator's Identity Object).

`ulOrigDeviceSn` contains the Serial Number of the connection originator (i.e. the contents of attribute #6 of instance #1 of the connection originator's Identity Object).

`ulProApi` contains the actual packet interval for the producer of the connection (T→O direction). The actual packet interval is the time between two subsequent packets (specified in units of microseconds).

`usProConnParams` contains the producer connection parameter for the connection (T→O direction). It follows the rules for network connection parameters as specified in section 3-5.5.1.1 “Network Connection Parameters“ in reference [4].

The 16-bit word of the producer connection parameter (connected to a `Forward_Open` command) is structured as follows:

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	Connection Size (in bytes)

Table 85: Meaning of Variable `ulProParams`

The values have the following meaning

- Connection Size

  This is the maximum size of data for each direction of the connection to be opened.

- Fixed/Variable

  This bit indicates whether the connection size discussed above is variable or fixed to the size specified as connection size.

  If *fixed* is chosen (bit is equal to 0), then the actual amount of data transferred in one transmission is exactly the specified connection size.

  If *variable* is chosen (bit is equal to 1), the amount of data transferred in one single transmission may be the value specified as connection size or a lower value. This option is currently not supported.

Note: The option „*variable*” is NOT supported.

- Priority

  These two bits code the priority according to the following table:

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

Table 86: Priority

- Connection Type

  The connection type can be specified according to the following table:

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 87: Connection Type

Note: The option „*Multicast*" is only supported for connections with CIP transport class 0 and class 1.

- 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. Reserved fields should always be set to the value.

Note: Redundant Owner connections are not supported by the EtherNet/IP Stack.

`ulConApi` contains the actual packet interval for the consumer of the connection (O→T direction). The actual packet interval is the time between two directly subsequent packets (specified in units of microseconds).

`usConConnParams` Similarly to `usProConnParams`, this variable contains the consumer connection parameter for the connection (O→T direction).. It also follows the rules for network connection parameters as specified in section 3-5.5.1.1 "Network Connection Parameters" in reference [4].

`bTimeoutMultiplier` contains the value of the connection timeout multiplier, which is needed for the determination of the connection timeout value. The connection timeout value is calculated by multiplying the RPI value (requested packet interval) with the connection timeout multiplier. Transmission on a connection is stopped when a timeout occurs after the connection timeout value calculated by this rule. The multiplier is specified as a code according to the subsequent table:

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

Table 88: Coding of Timeout Multiplier Values

For more details, see reference [4] section 3-5.5.1.4.

Figure 22 and Figure 23 below display a sequence diagram for the `EIP_OBJECT_CONNECTION_IND/RES` packet in case the host application uses the Basic, Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”).

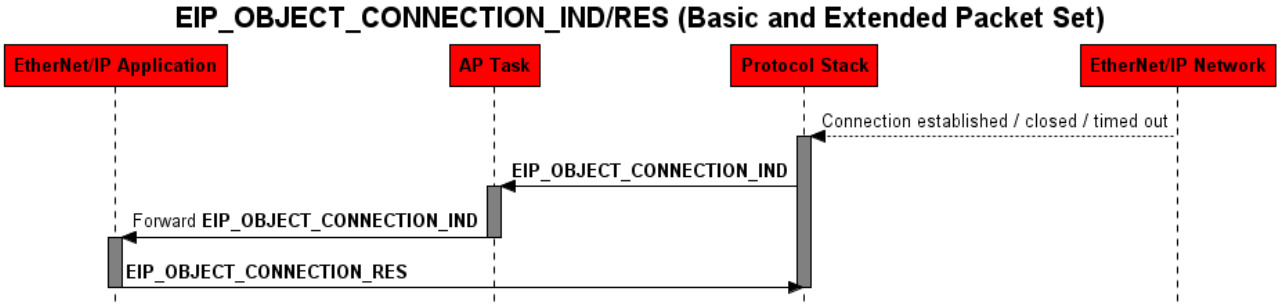

Figure 22: Sequence Diagram for the `EIP_OBJECT_CONNECTION_IND/RES` Packet for the Basic and Extended Packet Set

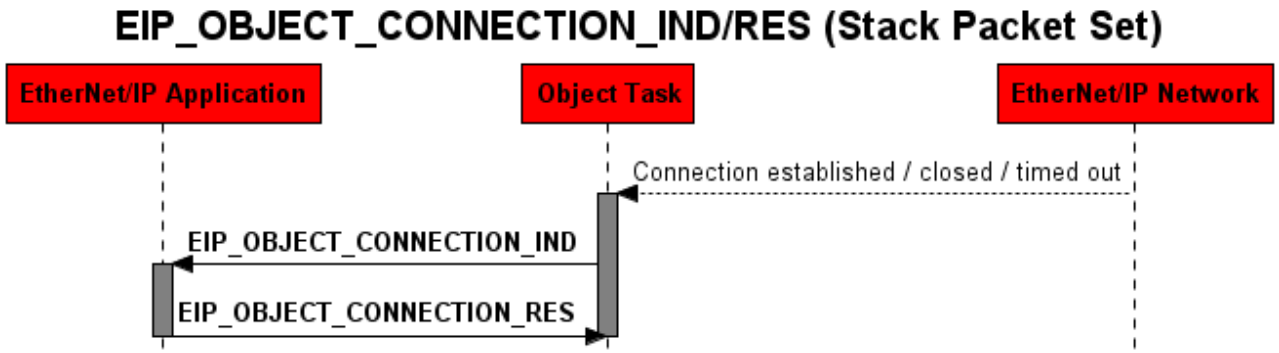

Figure 23: Sequence Diagram for the `EIP_OBJECT_CONNECTION_IND/RES` Packet for the Stack Packet Set

Packet Structure Reference

```
typedef struct EIP OBJECT OT CONNECTION Ttag
{
    TLR UINT32 ulConnHandle;
    TLR UINT32 ulReserved[3];
} EIP OBJECT OT CONNECTION T;

typedef struct EIP OBJECT TO CONNECTION Ttag
{
    TLR UINT32 ulClass;
    TLR UINT32 ulInstance;
    TLR UINT32 ulOTConnPoint;
    TLR UINT32 ulTOConnPoint;
    TLR UINT32 ulConnectionType;
} EIP_OBJECT_TO_CONNECTION_T;

typedef union EIP OBJECT CONNECTION Ttag
{
    EIP_OBJECT_OT_CONNECTION_T tOTConnection;
    EIP_OBJECT_TO_CONNECTION_T tTOConnection;
} EIP OBJECT CONNECTION T;

typedef struct EIP OBJECT EXT CONNECTION INFO Ttag
{
  TLR_UINT32  ulProConnId;
  TLR_UINT32  ulConConnId;

  TLR UINT32  ulConnSerialNum;
  TLR UINT16  usOrigVendorId;
  TLR UINT32  ulOrigDeviceSn;

  /* Producer parameters */
  TLR UINT32  ulProApi;
  TLR UINT16  usProConnParams;

  /* Consumer parameters */
  TLR_UINT32  ulConApi;
  TLR_UINT16  usConConnParams;

  TLR UINT8   bTimeoutMultiplier;
} EIP_OBJECT_EXT_CONNECTION_INFO_T;

typedef struct EIP_OBJECT_CONNECTION_IND_Ttag
{
  TLR UINT32 ulConnectionState;    /*!< Reason of changing the connection state  */
  TLR UINT32 ulConnectionCount;    /*!< Number of active connections             */
  TLR_UINT32 ulOutConnectionCount; /*!< Number of active originate connections   */
  TLR_UINT32 ulConfiguredCount;
  TLR_UINT32 ulActiveCount;
  TLR UINT32 ulDiagnosticCount;

  TLR UINT32 ulOrginator;
  EIP_OBJECT_CONNECTION_T tConnection; /*!< Gives extended information concerning
                                        the connection state (ulConnectionState)*/
  TLR UINT32 ulExtendedState;
  EIP OBJECT EXT CONNECTION INFO T tExtInfo;
}EIP OBJECT CONNECTION IND T;

#define EIP_OBJECT_CONNECTION_IND_SIZE \
                           sizeof(EIP OBJECT CONNECTION IND T)

typedef struct EIP OBJECT PACKET CONNECTION IND Ttag {
  TLR PACKET HEADER T tHead;
  EIP_OBJECT_CONNECTION_IND_T tData;
} EIP_OBJECT_PACKET_CONNECTION_IND_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_CONNECTION_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	83	*EIP_OBJECT_CONNECTION_IND* – Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See chapter* Status/Error Codes Overview
ulCmd	UINT32	0x1A2E	EIP_OBJECT_CONNECTION_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_OBJECT_CONNECTION_IND_T			
ulConnectionState	UINT32	0, 1	Reason of changing the connection state Connection established (1) Connection disconnected (0)
ulConnectionCount	UINT32		Number of established connections (does not include Listen Only connections)
ulOutConnectionCount	UINT32	0	Not supported for EtherNet/IP adapter
ulConfiguredCount	UINT32	0	Not supported for EtherNet/IP adapter
ulActiveCount	UINT32	0	Not supported for EtherNet/IP adapter
ulDiagnosticCount	UINT32	0	Not supported for EtherNet/IP adapter
ulOrginator	UINT32	0	Will always be 0 for EtherNet/IP adapter
tConnection	union EIP_OBJECT_CONNECTION_T		For the EtherNet/IP adapter only the union entry tTOConnection is important: ulClass: Class to which the connection was directed ulInstance: Corresponding class instance ulOTConnPoint: Input connection point ulTOConnPoint: Output connection point ulConnectionType: Type of the connection
ulExtendedState	UINT32	0, 1	0: No extended status 1: Connection timeout

Structure EIP_OBJECT_PACKET_CONNECTION_IND_T			Type: Indication
tExtInfo	EIP_OBJECT_EXT_CONNECTION_INFO_T		Additional connection information for incoming connections (i.e. ulOrginator == 0)

Table 89: EIP_OBJECT_CONNECTION_IND – Indication of Connection

Packet Structure Reference

```
struct EIP OBJECT PACKET CONNECTION RES Ttag
{
  TLR_PACKET_HEADER_T     tHead;
};
```

Packet Description

Structure EIP_OBJECT_PACKET_CONNECTION_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination Queue Reference
ulSrcId	UINT32	See rules in section 3.2.1	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A2F	Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information

Table 5: EIP_OBJECT_CONNECTION_RES – Response to indication of Connection

6.2.3 EIP_OBJECT_MR_REGISTER_REQ/CNF – Register an additional Object Class at the Message Router

This service can be used by the host application in order to register an additionally object class at the message router. This automatically extends the object model of the device by the given object class (see Figure 8 for the basic object model).

All explicit messages addressing this additional object class will then be forwarded to the host application via the indication EIP_OBJECT_CL3_SERVICE_IND (section 6.2.4).

Note: If using the Stack Packet Set: The source queue of this packet is directly bound to the new object. All indications for the new object will be sent to ulSrc and ulSrcId of the request packet (packet header).

The ulClass parameter represents the class code of the registered class. The predefined class codes are described in at the CIP specification Vol. 1 chapter 5.

CIP Class IDs are divided into the following address ranges to provide for extensions to device profiles.

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 90: Address Ranges for the ulClass parameter

Figure 24 and Figure 25 below display a sequence diagram for the EIP_OBJECT_MR_REGISTER_REQ/CNF packet in case the host application uses the Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”).

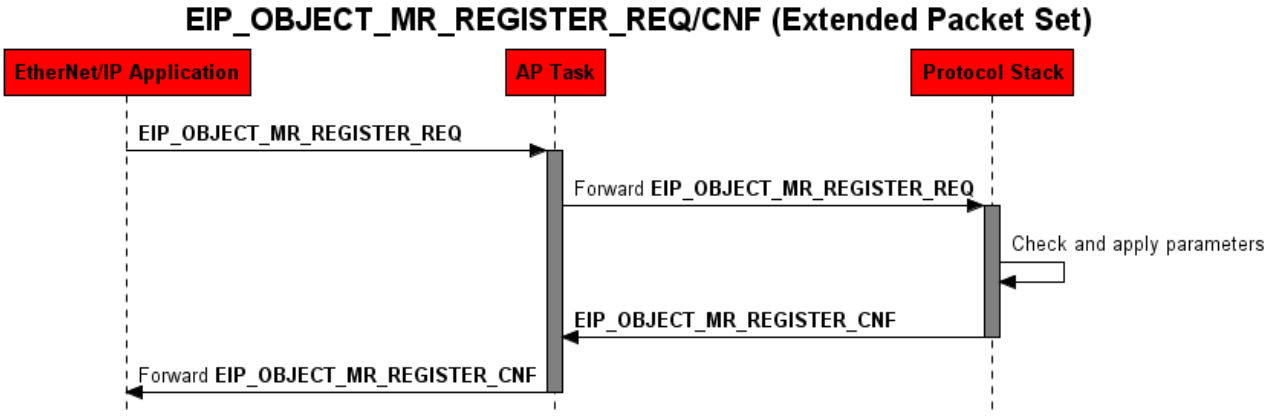

Figure 24: Sequence Diagram for the EIP_OBJECT_MR_REGISTER_REQ/CNF Packet for the Extended Packet Set

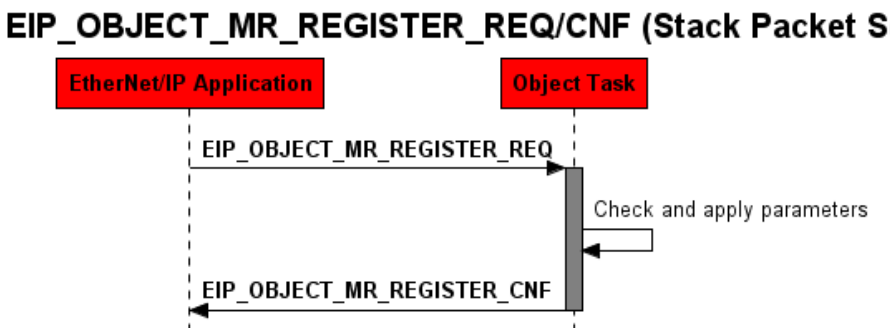

Figure 25: Sequence Diagram for the `EIP_OBJECT_MR_REGISTER_REQ/CNF` Packet for the Stack Packet Set

Packet Structure Reference

```
typedef struct EIP_OBJECT_MR_REGISTER_REQ_Ttag {
  TLR_HANDLE  hObjectQue;
  TLR_UINT32  ulClass;
  TLR_UINT32  ulAccessTyp;
} EIP_OBJECT_MR_REGISTER_REQ_T;

#define EIP_OBJECT_MR_REGISTER_REQ_SIZE \
                          sizeof(EIP_OBJECT_MR_REGISTER_REQ_T)

typedef struct EIP_OBJECT_MR_PACKET_REGISTER_REQ_Ttag {
  TLR_PACKET_HEADER_T tHead;
  EIP_OBJECT_MR_REGISTER_REQ_T tData;
} EIP_OBJECT_MR_PACKET_REGISTER_REQ_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_MR_REGISTER_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ OBJECT_QUE	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	12	EIP_OBJECT_MR_REGISTER_REQ_SIZE – Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	See *Table 44: EIP_OBJECT_MR_REGISTER_REQ – Packet Status/Error*
ulCmd	UINT32	0x1A02	EIP_OBJECT_MR_REGISTER_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	×	Routing, do not change
tData - Structure EIP_OBJECT_MR_REGISTER_REQ_T			
hObjectQue	HANDLE	0	Deprecated, set to 0
ulClass	UINT32	1..0xFFFF	Class identifier (predefined class code as described in the CIP specification Vol. 1 chapter 5 (reference [4]) Take care of the address ranges specified above within *Table 90: Address Ranges for the ulClass parameter.*
ulAccessTyp	UINT32	0	Reserved, set to 0.

Table 91: EIP_OBJECT_MR_REGISTER_REQ – Request Command for register a new class object

Packet Structure Reference

```
typedef struct EIP OBJECT PACKET MR REGISTER CNF Ttag {
  TLR_PACKET_HEADER_T tHead;
} EIP_OBJECT_PACKET_MR_REGISTER_CNF_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_MR_REGISTER_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1A03	EIP_OBJECT_MR_REGISTER_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	See rules in section 3.2.1	Destination Queue Handle

Table 92: EIP_OBJECT_MR_REGISTER_CNF – Confirmation Command of register a new class object

6.2.4 EIP_OBJECT_CL3_SERVICE_IND/RES - Indication of acyclic Data Transfer

This packet indicates an acyclic service coming from the network. It will only be received if:

- an additional object class has been registered using the command EIP_OBJECT_MR_REGISTER_REQ/CNF (see section 6.2.3 on page 130 of this document)
- or a service has been registered for an existing object using EIP_OBJECT_REGISTER_SERVICE_REQ/CNF (see section 6.2.11 on page 168 of this document)

It delivers the following parameters:

- the O→T connection ID of the class 3 connection, in case the service request is bound to a class 3 connection (connected)
- a CIP Service Code
- the CIP Object Class ID
- the CIP Instance number
- the CIP Attribute number
- an array containing unstructured data (depending on the service code)

The parameters service code, class ID, instance and attribute 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[] may not be present for services that do not need data sent along with the request (e.g. Get services). The ulLen field of the packet header can be evaluated to determine whether there is data available.

```
Service_data_size = tHead.ulLen - EIP_OBJECT_CL3_SERVICE_IND_SIZE
```

The parameter ulService holds the requested CIP service that shall be applied to the object instance selected by the variables ulObject and ulInstance of the indication packet.

CIP services are divided into different address ranges. The subsequent *Table 93: 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 [4]*).

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 [5])

Table 93: Specified Ranges of numeric Values of Service Codes (Variable ulService)

Note: Not every service is available on every object.
If you use a Class IDs that are in the Vendor Specific range (see *Table 7: Ranges for Object Class Identifiers*), use need to define by yourself what services and attributes are supported by this object class.
If you use a Class IDs that are not in the Vendor Specific range, the CIP specification describes all required and optional services and attributes the class supports.
Depending on this the host application must implement the handling of incoming services.

Table 94: Service Codes for the Common Services according to the CIP specification lists the service codes for the Common Services. This table is taken from the CIP specification (“Volume 1 Common Industrial Protocol Specification Chapter 5, Table 5-1.1”, see reference [4]).

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

Service code (numeric value of `ulService`)	Service to be executed
10	Set_Attribute_Single
11	Find_Next_Object_Instance
12-13	Reserved for future use
14	Error Response (used by DevNet only)
15	Restore
16	Save
17	No Operation (NOP)
18	Get_Member
19	Set_Member
1A	Insert_Member
1B	Remove_Member
1C	GroupSync
1D-31	Reserved for additional Common Services

Table 94: Service Codes for the Common Services according to the CIP specification

Depending on what services, instances and attributes are supported by the addressed object, the host application must answer the service with either success or with an appropriate error code.

Therefore, the response packet holds two error fields: ulGRC and ulERC

The Generic Error Code (`ulGRC`) can be used to indicate whether the service request could be processed successfully or not. A list of all possible codes is provided in section 8.5 "*General EtherNet/IP Error Codes*" of this document. The most common General Error Codes are:

General Status Code (specified hexadecimally)	Status Name	Description
00	Success	The service has successfully been performed by the specified object.
05	Path destination unknown	The path references an unknown object class, instance or structure element causing the abort of path processing.
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.
0C	Object state conflict	The object is not able to perform the requested service in the current mode or state
0E	Attribute not settable	It has been tried to change a non-modifiable attribute.
10	Device state conflict	The current mode or state of the device prevents the execution of the requested service.
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.

General Status Code (specified hexadecimally)	Status Name	Description
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.

Table 95: Most common General Status Codes

The Extended Error Code (ERC) can be used to describe the occurred error having already been classified by the generic error code in more detail.

If the service will be answered with success, additional data can be sent with the reply in the abData field. The byte size of the data must be added to the basic packet length (`EIP_OBJECT_CL3_SERVICE_RES_SIZE`) in the `ulLen` field of the packet header.

Figure 26 and Figure 29 below display a sequence diagram for the `EIP_OBJECT_CL3_SERVICE_IND/RES` packet in case the host application uses the Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”).

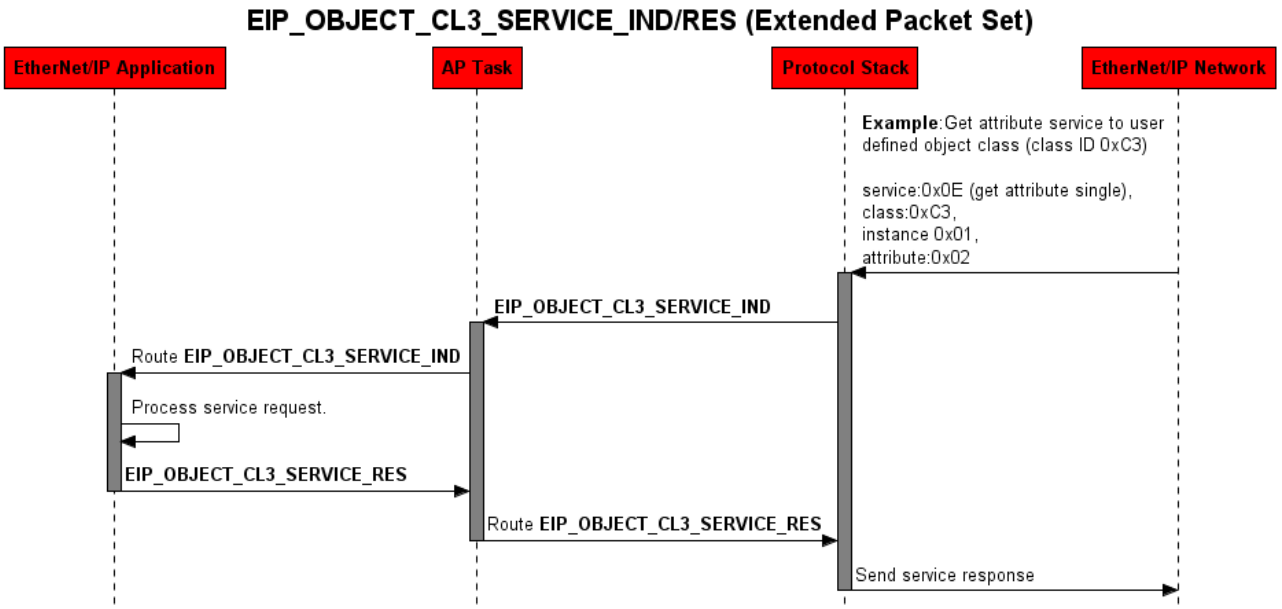

Figure 26: Sequence Diagram for the `EIP_OBJECT_CL3_SERVICE_IND/RES` Packet for the Extended Packet Set

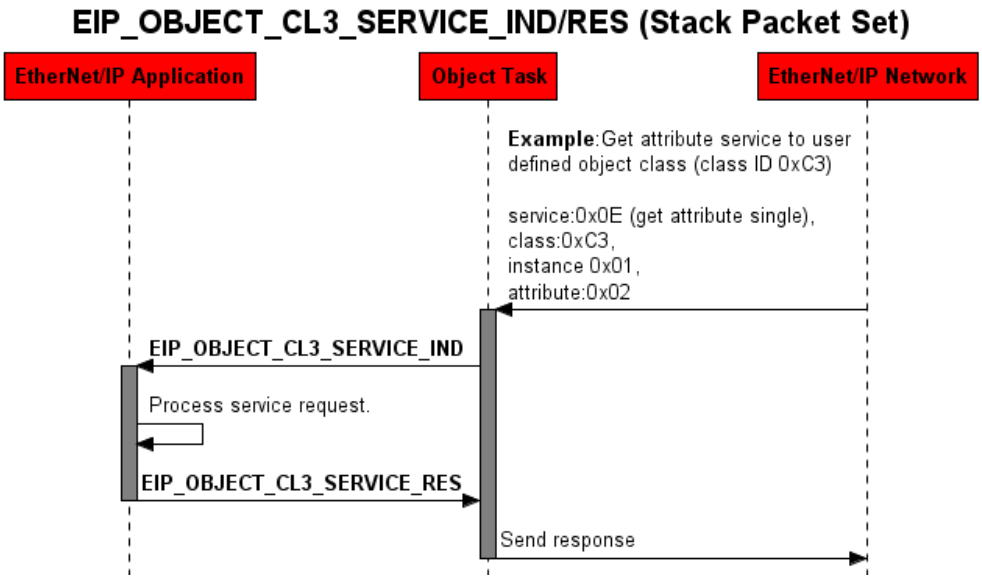

Figure 27: Sequence Diagram for the `EIP_OBJECT_CL3_SERVICE_IND/RES` Packet for the Stack Packet Set

Packet Structure Reference

```
 typedef struct EIP OBJECT CL3 SERVICE IND Ttag
{
  TLR_UINT32    ulConnectionId;          /*!< Connection Handle    */
  TLR_UINT32    ulService;
  TLR_UINT32    ulObject;
  TLR UINT32    ulInstance;
  TLR UINT32    ulAttribute;
  TLR UINT8     abData[1];
} EIP_OBJECT_CL3_SERVICE_IND_T;


typedef struct EIP OBJECT PACKET CL3 SERVICE IND Ttag
{
  TLR 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

Structure EIP_OBJECT_PACKET_CL3_SERVICE_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	20 + n	Packet Data Length (In Bytes) n = Length of Service Data Area
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See Packet Structure Reference
ulCmd	UINT32	0x1A3E	EIP_OBJECT_CL3_SERVICE_IND - Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information
tData - Structure EIP_OBJECT_CL3_SERVICE_IND_T			
ulConnectionId	UINT32	0 ... 2^{32}-1	For connected (class 3) service request, this field holds the O→T connection ID of that connection. If the service request is unconnected this filed is always 0.
ulService	UINT32	1-0xFF	CIP Service Code
ulObject	UINT32	1-0xFFFF	CIP Class ID
ulInstance	UINT32	1-0xFFFF	CIP Instance Number
ulAttribute	UINT32	0-0xFFFF	CIP Attribute Number The attribute number is 0, if the service does not address a specific attribute but the whole instance.
abData[]	Array of UINT8		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 96: EIP_OBJECT_CL3_SERVICE_IND - Indication of acyclic Data Transfer

Packet Structure Reference

```
 typedef struct EIP OBJECT CL3 SERVICE RES Ttag
{
  TLR UINT32    ulConnectionId;          /*!< Connection Handle    */
  TLR UINT32    ulService;
  TLR UINT32    ulObject;
  TLR UINT32    ulInstance;
  TLR UINT32    ulAttribute;
  TLR_UINT32    ulGRC;                 /*!< Generic Error Code   */
  TLR UINT32    ulERC;                 /*!< Extended Error Code  */
  TLR UINT8     abData[1];
}EIP OBJECT CL3 SERVICE RES T;

typedef struct EIP OBJECT PACKET CL3 SERVICE RES Ttag
{
  TLR 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

Structure EIP_OBJECT_PACKET_CL3_SERVICE_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	28 + n	Packet Data Length (In Bytes) where n = Length of Service Data Area
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A3F	EIP_OBJECT_CL3_SERVICE_RES - Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information
tData - Structure EIP_OBJECT_CL3_SERVICE_RES_T			
ulConnectionId	UINT32	0 ... 2^{32}-1	Connection Id from the indication packet
ulService	UINT32	1-0xFF	CIP Service Code from the indication packet
ulObject	UINT32	1-0xFFFF	CIP Object from the indication packet
ulInstance	UINT32	1-0xFFFF	CIP Instance from the indication packet
ulAttribute	UINT32	0-0xFFFF	CIP Attribute from the indication packet
ulGRC	UINT32		Generic Error Code
ulERC	UINT32		Extended Error Code
abData[]	Array of UINT8		n bytes of service data (depending on service)

Table 97: EIP_OBJECT_CL3_SERVICE_RES – Response to Indication of acyclic Data Transfer

6.2.5 EIP_OBJECT_AS_REGISTER_REQ/CNF – Register a new Assembly Instance

This service can be used by the host application in order to create a new Assembly Instance (for more information about assembly instances, see section *The CIP Messaging Model* on page 25).

The parameter `ulInstance` is the assembly instance number that hast to be registered at the assembly class object.

Table 98 lists the Assembly Instance Number Ranges specified by the CIP Networks Library (reference [4]).

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 98: 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 connections. These instances numbers must not be used for additional assembly instances.

Data belonging to this specific assembly instance will be mapped into the dual port memory at the offset address `ulDPMOffset`.

Note: This offset (`ulDPMOffset`) is not the total DPM offset. It is the relative offset within the beginning of the corresponding input/output data images a`bPd0Input[5760]` and `abPd0Output[5760]`.
So, usually the first instance (for each data direction) that is created will have `ulDPMOffset = 0`.
If multiple assembly instances are registered, make sure that the data range of this instance do not overlap in the DPM.

Note: When using the Stack Packet Set actually no DPM Offset is necessary. However, the stack still checks this parameter. So make sure that there are no overlapping data areas.

The data length (in bytes) the assembly instance shall hold can be provided in `ulSize`. The maximum size of an instance may not exceed 504 bytes.

With the parameter `ulFlags` the properties of the assembly instance can be configured. Properties can be set according to Table 100: Assembly Instance below.

As long as no data has ever been set and no connection has been established, the Assembly Object Instance holds zeroed data.

For host applications using the Stack Packet Set: The confirmation of the command returns a tri-state buffer (`hDataBuf`). This triple buffer is used to update the assembly instance's process data.

Figure 28 and Figure 29 below display a sequence diagram for the `EIP_OBJECT_AS_REGISTER_REQ/CNF` packet in case the host application uses Extended or Stack Packet Set (see 4.3 "Configuration Using the Packet API").

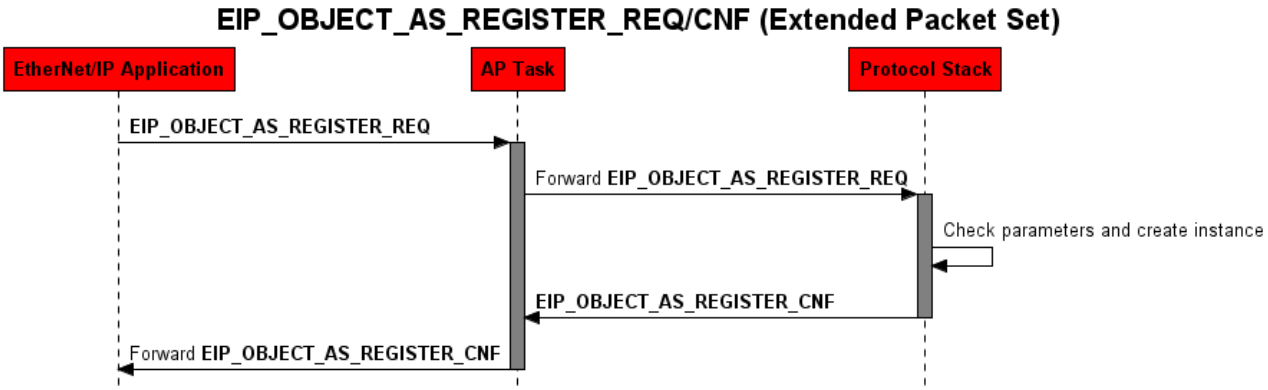

Figure 28: Sequence Diagram for the `EIP_OBJECT_AS_REGISTER_REQ/CNF` Packet for the Extended Packet Set

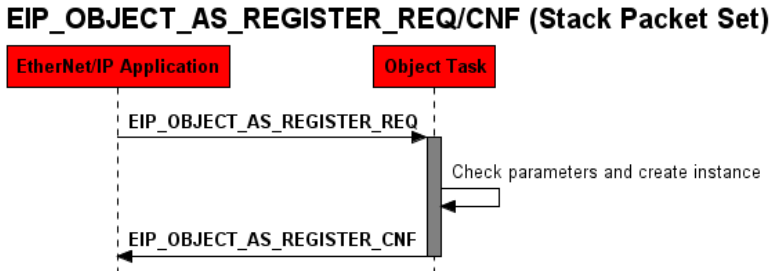

Figure 29: Sequence Diagram for the `EIP_OBJECT_AS_REGISTER_REQ/CNF` Packet for the Stack Packet Set

Packet Structure Reference

```
typedef struct EIP OBJECT AS REGISTER REQ Ttag {
  TLR UINT32      ulInstance;
  TLR UINT32      ulDPMOffset;
  TLR_UINT32      ulSize;
  TLR_UINT32      ulFlags;
} EIP OBJECT AS REGISTER REQ T;

#define EIP OBJECT AS REGISTER REQ SIZE \
                          sizeof(EIP OBJECT AS REGISTER REQ T)

typedef struct EIP_OBJECT_AS_PACKET_ REGISTER_REQ_Ttag {
  TLR PACKET HEADER T tHead;
  EIP OBJECT AS REGISTER REQ T tData;
} EIP_OBJECT_AS_PACKET_REGISTER_REQ_T;
```

Packet Description

Structure EIP_OBJECT_A_PACKET _REGISTER_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0, 0x20	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	16	EIP_OBJECT_AS_REGISTER_REQ_SIZE - Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1A0C	EIP_OBJECT_AS_REGISTER_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	×	Routing, do not change
tData - Structure EIP_OBJECT_AS_REGISTER_REQ_T			
ulInstance	UINT32	0x0000001...0xFFFFFFFF (except 0xC0 and 0xC1, see description above)	Assembly instance number See *Table 98: Assembly Instance Number Ranges*
ulDPMOffset	UINT32	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]. So, usually the first instance (for each data direction) that is created 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	1..504	Size of the data area for the assembly instance data in bytes. **Note:** the size of the assembly instance also depends on the flags that are set in the field ulFlags.
ulFlags	UINT32	Bitmap	Property Flags for the assembly instance See *Table 100: Assembly Instance*

Table 99: EIP_OBJECT_AS_REGISTER_REQ – Request Command for create an Assembly Instance

The following table shows the meaning of the single bits which can be used to configured specific assembly instance properties:

Bits	Name (Bitmask)	Description
31 - 11	Reserved	Reserved for future use

10	EIP_AS_FLAG_FORWARD_SEQUENCE_COUNT (0x00000400)	For input assemblies (consuming data from the network), this flag decides whether the 4 byte sequence count is mapped into the IO data when being written into the triple buffer or DPM. 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.
		This flag can only be used in conjunction with flag EIP_AS_FLAG_FORWARD_RUNIDLE. Mapping only the sequence count without mapping the run/idle header is not possible.
		The sequence counter wraps-around to zero at value 65536.
		If the bit is set, the sequence count will be part of the IO data image. In that case the triple buffer / DPM layout of assembly data looks like the following:
		`| ulSize = 4 + 4 + size of data |`
		`|----------|`
		`| 32 Bit | 32 Bit | |`
		`|----------|----------|----------|`
		`| Seq. Count | Run/Idle | data |`
		Note: In case the assembly instance does not receive a run/idle header from the network, the run/idle header in the IO image will always show "run" (0x00000001).
		Note: For class 0 connections, there is no sequence count that can be received from the network. In that case, the sequence count value in the IO image will be incremented each time the EtherNet/IP stack receives a process data frame. For class 1 connections, the most recent sequence count field encountered on the network is copied into the triple buffer (LOM) and DPM (LFW).
		Note: The sequence count is incremented only when the connected PLC application updates its production data.
		Note: The sequence count is not designed to detect lost packets
		Note: The sequence count information remains unchanged when the assembly data is modified over an EtherNet/IP explicit service, whereas the data may has changed.

9	EIP_AS_FLAG_INVISIBLE (0x00000200)	This bit decides whether or not the assembly instance can be accessed via explicit services from the network. If the bit is set (1), the assembly instance is **not** accessible (invisible). If the bit is cleared (0), the assembly instance is accessible (visible)

Bits	Name (Bitmask)	Description

8	EIP_AS_FLAG_FORWARD_RUNIDLE
	(0x00000100)
		For input assemblies instances, this flag decides whether the run/idle header shall be present in the triple buffer (LOM) or DPM (LFW).
		This way the host application has the possibility to evaluate the run/idle information on its own. 4 additional bytes have to be reserved in the assembly's size and offsets.
		If the bit is set (1), the run/idle header will be part of the input data image.
		Note: This property only applies to assembly instances that have bit 0 set (1).
		In that case, the triple buffer / DPM layout of assembly data looks like the following:
		`| ulSize = 4 + size of data |`
		`|-----------------------------------|`
		`| 32 Bit | |`
		`|---------------|--------------------|`
		`| Run/Idle | data |`

7	EIP_AS_FLAG_FIX_SIZE
	(0x00000080)
		This flag decides whether the assembly instance allows a connection to be established with a smaller connection size than defined in `ulSize` or whether only the exact match is accepted.
		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:
		1) `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.`
		2) `ulSize = 6 (Bit 7 of ulFlags is 1)`
		`A connection can only be opened with a matching I/O size, i.e. 6.`

6	EIP_AS_FLAG_HOLDSTATE (0x00000040)	This flag decides whether the assembly instance data that is mapped into the DPM memory area is cleared upon closing or timeout of the connection or whether the last received data is left unchanged in the memory. If the bit is set (1), the data will be left unchanged. This property only applies to assembly instances that have bit 0 set (1), since only those instances receive data from the network.
5	EIP_AS_FLAG_CONFIG (0x00000020)	If set (1), this assembly instance is a configuration assembly instance, which can be used to receive configuration data upon connection establishment. For further information have a look at the Packet `EIP_OBJECT_CONNECTION_CONFIG_IND/RES` – Indication of Configuration Data received during 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 exchange cyclically.
4	EIP_AS_FLAG_NEWDATA (0x00000010)	This flag is used internally and must be set to 0

Bits	Name (Bitmask)	Description
3	EIP_AS_FLAG_MODELESS (0x00000008)	If set (1), the assembly instance's real time format is modeless, i.e. it does **not** contain run/idle information. If not set (0), the assembly instance's real time format is the 32-Bit Run/Idle header. For more information about real time format see section 2.4.3.1 "Real Time Format".
2	EIP_AS_FLAG_TRIPLEBUF (0x00000004)	This flag is used internally and must be set to 0
1	EIP_AS_FLAG_ACTIVE (0x00000002)	This flag is used internally and must be set to 0
0	EIP_AS_FLAG_READONLY (0x00000001)	This flag decides whether the newly registered assembly is an input or an output assembly. If set (1), the assembly instance is an output assembly instance (can be used for the O→T direction). It is able to consume data from the network. Data for this instance will be mapped into the DPM Input area (data flow: network → DPM). If cleared (0), the assembly instance is an input assembly instance (can be used for the T→O direction). It is able to produce data on the network. Data for this instance will be mapped from the DPM Output area (data flow: DPM → network).

Table 100: Assembly Instance Property Flags

Source Code Example

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

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

EIP OBJECT AS PACKET REGISTER REQ T tReq;

tReq.tHead.ulCmd = EIP OBJECT AS REGISTER REQ;
tReq.tHead.ulLen = EIP_OBJECT_AS_REGISTER_REQ_SIZE;

tReq.tData.ulInstance  = 100;
tReq.tData.ulSize      = 16;
tReq.tData.ulFlags     = EIP AS FLAG MODELESS | EIP AS FLAG FIX 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 realtime 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 FLAG READONLY;
tReq.tData.ulDPMOffset = 0;
```

Packet Structure Reference

```
typedef struct EIP OBJECT AS REGISTER CNF Ttag {
  TLR UINT32  ulInstance;
  TLR UINT32  ulDPMOffset;
  TLR_UINT32  ulSize;
```

```
  TLR UINT32  ulFlags;
  TLR HANDLE  hDataBuf;
} EIP OBJECT AS REGISTER CNF T;

#define EIP_OBJECT_AS_REGISTER_CNF_SIZE \
                          sizeof(EIP OBJECT AS REGISTER CNF T)

typedef struct EIP OBJECT AS PACKET REGISTER CNF Ttag {
  TLR_PACKET_HEADER_T tHead;
} EIP_OBJECT_AS_PACKET_ REGISTER_CNF_T;
```

Packet Description

Structure EIP_OBJECT_AS_PACKET_ REGISTER_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue-Handle, unchanged
ulSrc	UINT32	See rules in section 3.2.1	Source Queue-Handle, unchanged
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	20	EIP_OBJECT_AS_REGISTER_CNF_SIZE - Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1A0D	EIP_OBJECT_AS_REGISTER_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	×	Routing, do not change
tData - Structure EIP_OBJECT_AS_REGISTER_CNF_T			
ulInstance	UINT32		Instance of the Assembly Object (from the request packet)
ulDPMOffset	UINT32		Offset of the data in the dual port memory (from the request packet)
ulSize	UINT32	<=504	Size of the assembly instance data (from the request packet)
ulFlags	UINT32		Property Flags of the assembly instance (from the request packet)
hDataBuf	UINT32		Handle to the tri-state buffer of the assembly instance

Table 101: EIP_OBJECT_AS_REGISTER_CNF – Confirmation Command of register a new class object

6.2.6 EIP_OBJECT_ID_SETDEVICEINFO_REQ/CNF – Set the Device's Identity Information

This request packet can be used by the host application in order to configure the device's Identity Object Instance (CIP Class ID 0x01).

Figure 30 and Figure 31 below display a sequence diagram for the EIP_OBJECT_ID_SETDEVICEINFO_REQ/CNF packet in case the host application uses Extended or Stack Packet Set (see section *Configuration Using the Packet API* on page *77*).

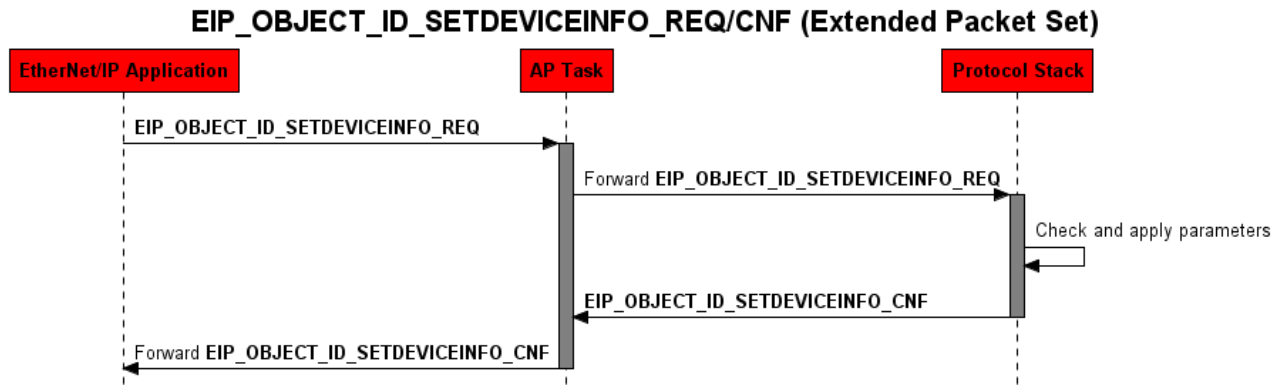

Figure 30: Sequence Diagram for the EIP_OBJECT_ID_SETDEVICEINFO_REQ/CNF Packet for the Extended Packet Set

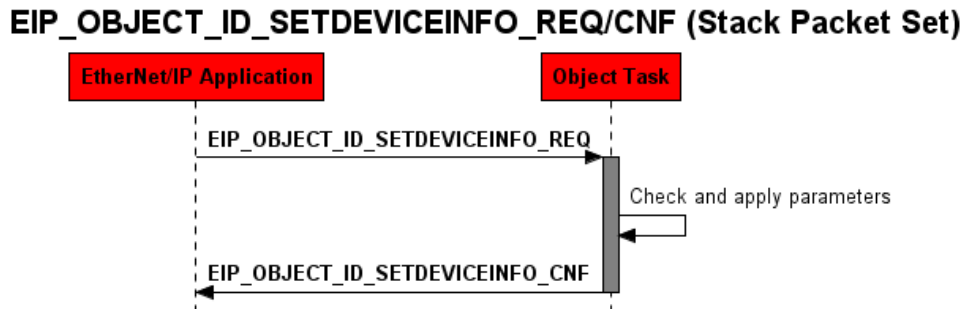

Figure 31: Sequence Diagram for the EIP_OBJECT_ID_SETDEVICEINFO_REQ/CNF Packet for the Stack Packet Set

Packet Structure Reference

```
#define EIP_ID_MAX_PRODUKTNAME_LEN 32
typedef struct EIP OBJECT ID SETDEVICEINFO REQ Ttag {
  TLR UINT32 ulVendId;
  TLR UINT32 ulProductType;
  TLR_UINT32 ulProductCode;
  TLR_UINT32 ulMajRev;
  TLR UINT32 ulMinRev;
  TLR UINT32 ulSerialNumber;
  TLR UINT8  abProductName[EIP ID MAX PRODUKTNAME LEN]
} EIP OBJECT ID SETDEVICEINFO REQ T;

#define EIP_OBJECT_ID_SETDEVICEINFO_REQ_SIZE \
                          (sizeof(EIP OBJECT ID SETDEVICEINFO REQ T))

typedef struct EIP OBJECT PACKET ID SETDEVICEINFO REQ Ttag {
  TLR PACKET HEADER T tHead;
  EIP_OBJECT_ID_SETDEVICEINFO_REQ_T tData;
} EIP_OBJECT_PACKET_ID_SETDEVICEINFO_REQ_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_ID_SETDEVICEINFO_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ OBJECT_QUE	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	24 + *n*	24 + *n* - Packet data length in bytes *n* is the Application data count of abProductName[] in bytes *n* = 0 … EIP_ID_MAX_PRODUKTNAME_LEN (32)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1A16	EIP_OBJECT_ID_SETDEVICEINFO_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	×	Routing, do not change
tData - Structure EIP_OBJECT_ID_SETDEVICEINFO_REQ_T			
ulVendID	UINT32	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). Default value: 283 (Hilscher) The value 0 is not a valid Vendor ID. However, when using value 0 here, the stack automatically chooses the default Vendor ID (283 - Hilscher GmbH).

Structure `EIP_OBJECT_PACKET_ID_SETDEVICEINFO_REQ_T`			**Type: Request**
`ulProductType`	UINT32	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. Default: 0x0C (Communication Device) 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 The value 0 is not a valid Product Type. However, when using value 0 here, the stack automatically chooses the default Product Type (0x0C).
`ulProductCode`	UINT32	0..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. The value 0 is not a valid Product Code. However, when using value 0 here, the stack automatically chooses the default Product Code dependent on the chip type (netX50/100 etc.) that is used.
`ulMajRev`	UINT32	1..127	Major revision Value 0 is not a valid major revision number. If major revision and minor revision both are set to 0, the stack uses the default value predefined in the firmware.
`ulMinRev`	UINT32	1..255	Minor revision Value 0 is not a valid minor revision number. If major revision and minor revision both are set to 0, the stack uses the default value predefined in the firmware.
`ulSerialNumber`	UINT32	0..65535	Serial Number of the device This parameter is a number used in conjunction with the Vendor ID to form 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. Usually, this number will be set automatically by the firmware, if a security memory is available. In this case leave this parameter at value 0.

Structure `EIP_OBJECT_PACKET_ID_SETDEVICEINFO_REQ_T`			**Type: Request**
`abProductName[32]`	UINT8[]		Product/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” `abProductName` [0] = 9 `abProductName [1..9] = "Test Name"` **Note:** If an empty device name (“”) is configured, the firmware will use the default device name. For an overview of default names see Table 60.

Table 102: `EIP_OBJECT_ID_SETDEVICEINFO_REQ` – Request Command for open a new connection

Source Code Example

```
#define MY VENDOR ID 283
#define PRODUCT_COMMUNICATION_ADAPTER 12

void APS SetDeviceInfo req(EIP APS RSC T FAR* ptRsc )
{
  EIP APS PACKET T* ptPck;

  if(TLR_POOL_PACKET_GET(ptRsc->tLoc.hPool,&ptPck) == TLR_S_OK) {

    ptPckt->tDeviceInfoReq.tHead.ulCmd = EIP OBJECT ID SETDEVICEINFO REQ;
    ptPckt->tDeviceInfoReq.tHead.ulSrc = (UINT32)ptRsc->tLoc.hQue;
    ptPckt->tDeviceInfoReq.tHead.ulSta = 0;
    ptPckt->tDeviceInfoReq.tHead.ulId  = ulIdx;
    ptPckt->tDeviceInfoReq.tHead.ulLen = EIP_OBJECT_ID_SETDEVICEINFO_REQ_SIZE;

    ptPckt->tDeviceInfoReq.tData.ulVendId = MY VENDOR ID;
    ptPckt->tDeviceInfoReq.tData.ulProductType = PRODUCT COMMUNICATION ADAPTER;
    ptPckt->tDeviceInfoReq.tData.ulProductCode = 1;
    ptPckt->tDeviceInfoReq.tData.ulMajRev = 1;
    ptPckt->tDeviceInfoReq.tData.ulSerialNumber = 1;
    ptPckt->tDeviceInfoReq.tData.abProductName[0] =15;
    TLR MEMCPY(&ptPckt->tDeviceInfoReq.tData.abProductName[1], “Scanner Example”,
               ptPckt->tDeviceInfoReq.tData.abProductName[0]);

    TLR_QUE_SENDPACKET_FIFO((TLR_HANDLE)ptRsc->tRem.hQueEipObject, ptPck,
                            TLR INFINITE);
  }
}
```

Packet Structure Reference

```
typedef struct EIP OBJECT ID SETDEVICEINFO CNF Ttag {
  TLR_PACKET_HEADER_T tHead;
} EIP_OBJECT_PACKET_ID_SETDEVICEINFO_CNF_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_ID_SETDEVICEINFO_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1A17	EIP_OBJECT_ID_SETDEVICEINFO_CNF – Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	×	Routing, do not change

Table 103: EIP_OBJECT_ID_SETDEVICEINFO_CNF – Confirmation Command of setting device information

Source Code Example

```
void APS SetDeviceInfo cnf(EIP APS RSC T FAR* ptRsc, EIP APS PACKET T* ptPck )
{
  if( ptPck->tDeviceInfoCnf.tHead.ulSta != TLR S OK){
    APS ErrorHandling(ptRsc);
  }

  TLR_POOL_PACKET_RELEASE(ptRsc->tLoc.hPool, ptPck);
```

6.2.7 EIP_OBJECT_GET_INPUT_REQ/CNF – Getting the latest Input Data

Note: Host applications should not use this packet anymore.
To read the input data always use the Triple-Buffers (LOM) or read the corresponding DPM area the input data is mapped to.

This service can be used by the host application to get the latest input data.

As long as no input data has ever been received, 0 data as Input Data Block will be returned.

The flag `fClearFlag` indicates that the Input Data Block is valid or cleared. In the event the flag is set to `TLR_FALSE`(0), data exchange is successful. If the flag is `TLR_TRUE`(1), the device is not in data exchange.

The flag `fNewFlag` indicates whether the input data has been updated by the stack. If not, the flag is set to `TLR_FALSE`(0) and the returned Input Data Block will be the same as the previous one.

The maximum number of input data that may be passed cannot exceed 504 bytes.

Packet Structure Reference

```
typedef struct EIP_OBJECT_GET_INPUT_REQ_Ttag {
  TLR UINT32 ulInstance;
} EIP OBJECT GET INPUT REQ T;

#define EIP OBJECT GET INPUT REQ SIZE \
                   sizeof(EIP OBJECT GET INPUT REQ T)

typedef struct EIP OBJECT PACKET GET INPUT REQ Ttag {
  TLR PACKET HEADER T tHead;
  EIP OBJECT GET INPUT REQ T tData;
} EIP_OBJECT_PACKET_GET_INPUT_REQ_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_GET_INPUT_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ OBJECT_QUE	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	4	EIP_OBJECT_GET_INPUT_REQ_SIZE - Packet data length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview.*
ulCmd	UINT32	0x1A20	EIP_OBJECT_GET_INPUT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	×	Routing, do not change
tData - structure EIP_OBJECT_GET_INPUT_REQ_T			
ulInstance	UINT32		Reference to the Instance of the Assembly Object

Table 104: EIP_OBJECT_GET_INPUT_REQ – Request Command for getting Input Data

Packet Structure Reference

```
#define EIP OBJECT MAX INPUT DATA SIZE 2048

typedef struct EIP OBJECT GET INPUT CNF Ttag {
  TLR UINT32 ulInstance;
  TLR BOOLEAN32 fClearFlag;
  TLR BOOLEAN32 fNewFlag;
  TLR UINT8 abInputData[EIP OBJECT MAX INPUT DATA SIZE];
} EIP_OBJECT_GET_INPUT_CNF_T;

#define EIP OBJECT GET INPUT CNF SIZE \
                            (sizeof(EIP OBJECT GET INPUT CNF T)- \
                             EIP OBJECT MAX INPUT DATA SIZE)

typedef struct EIP_OBJECT_PACKET_GET_INPUT_CNF_Ttag {
  TLR PACKET HEADER T tHead;
  EIP OBJECT GET INPUT CNF T tData;
} EIP_OBJECT_PACKET_GET_INPUT_CNF_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_GET_INPUT_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination queue handle, untouched
ulSrc	UINT32	See rules in section 3.2.1	Source queue handle, untouched
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	12 + *n*	EIP_OBJECT_GET_INPUT_REQ_SIZE + *n* - Packet data length in bytes *n* is the Application data count of abInputData[] in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, untouched
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1A21	EIP_OBJECT_GET_INPUT_CNF - Command
ulExt	UINT32	0	Extension, untouched
ulRout	UINT32	×	Routing, do not change
tData - structure EIP_OBJECT_GET_INPUT_CNF_T			
ulInstance	UINT32		Reference to the Assembly Instance
fClearFlag	BOOL32	0,1	Flag that indicates if set to TLR_FALSE(0) that the Output data block is valid. If set to TLR_TRUE(1), the Output data block is cleared and zeroed.
fNewFlag	BOOL32	0,1	Flag that indicates if set to TLR_TRUE(1) that new Output data has been received since the last received EIP_OBJECT_GET_OUTPUT command.
abInputData[…]	UINT8[]		Field for input data

Table 105: EIP_OBJECT_GET_INPUT_CNF – Confirmation Command of getting the Input Data

6.2.8 EIP_OBJECT_RESET_IND/RES – Indication of a Reset Request from the network

This indication 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.

It is important to send the reset response packet right away, since this triggers the response to the reset service on the network. So, in case the response to the indication is not sent at all, the requesting node on the network will not get any answer to its reset request.

There are two reset types defined (0 and 1) that tell the host application how the reset shall be performed. Basically, the difference between these is the way the configuration data is handled. Reset type 0 (the default reset type that every EtherNet/IP device needs to support) only emulates a power cycle, where all configuration data (such as the IP settings) will be kept. Reset type 1 on the other side shall bring the device back to the factory defaults.

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. **Note:** This reset type is not supported by default. It needs to be enabled separately using the command EIP_OBJECT_SET_PARAMETER_REQ (see section 6.2.14).
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 106: Allowed Values of ulResetTyp

Figure 32, Figure 33 and Figure 34 below display a sequence diagram for the EIP_OBJECT_RESET_IND/RES packet with reset type 0 and 1. For all available Packet Sets (Basic, Extended or Stack Packet Set - see 4.3 “Configuration Using the Packet API”) it is illustrated what the host application needs to do when receiving the reset indication.

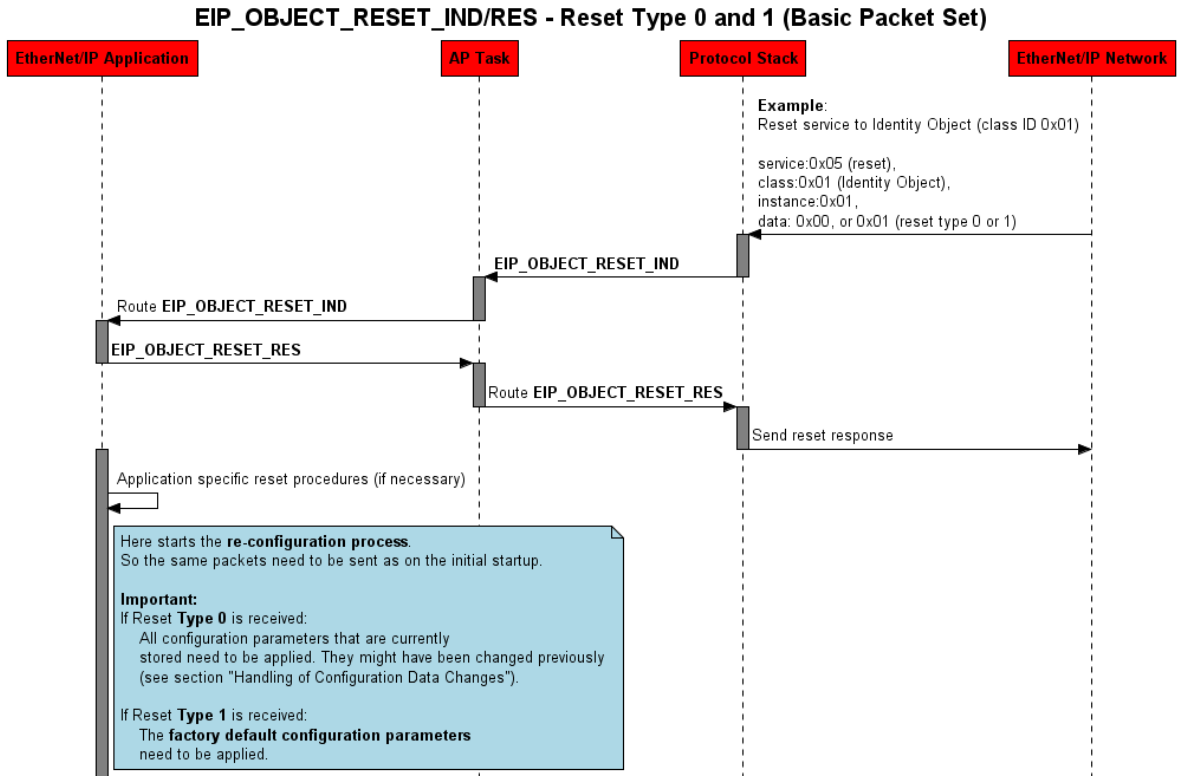

Figure 32: Sequence Diagram for the `EIP_OBJECT_RESET_IND/RES` Packet for the Basic Packet Set

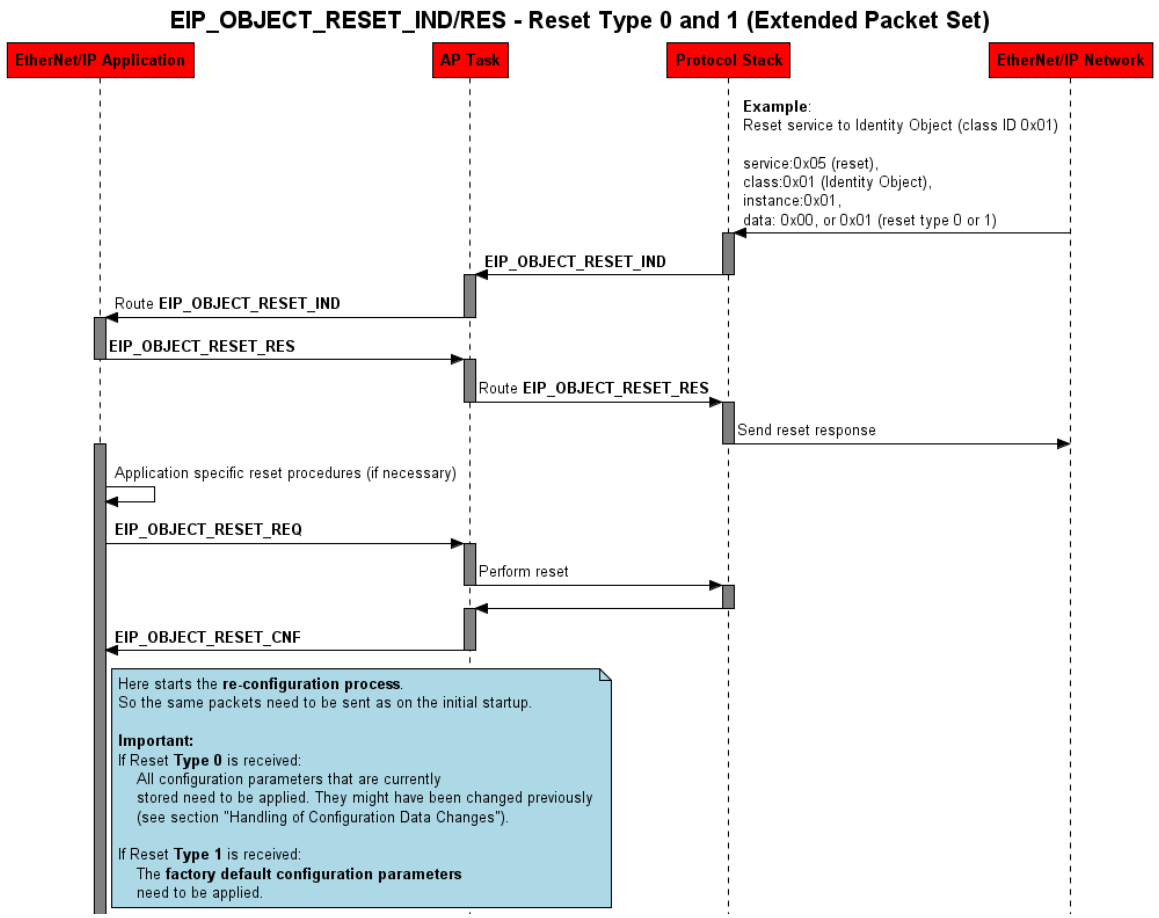

Figure 33: Sequence Diagram for the `EIP_OBJECT_RESET_IND/RES` Packet for the Extended Packet Set

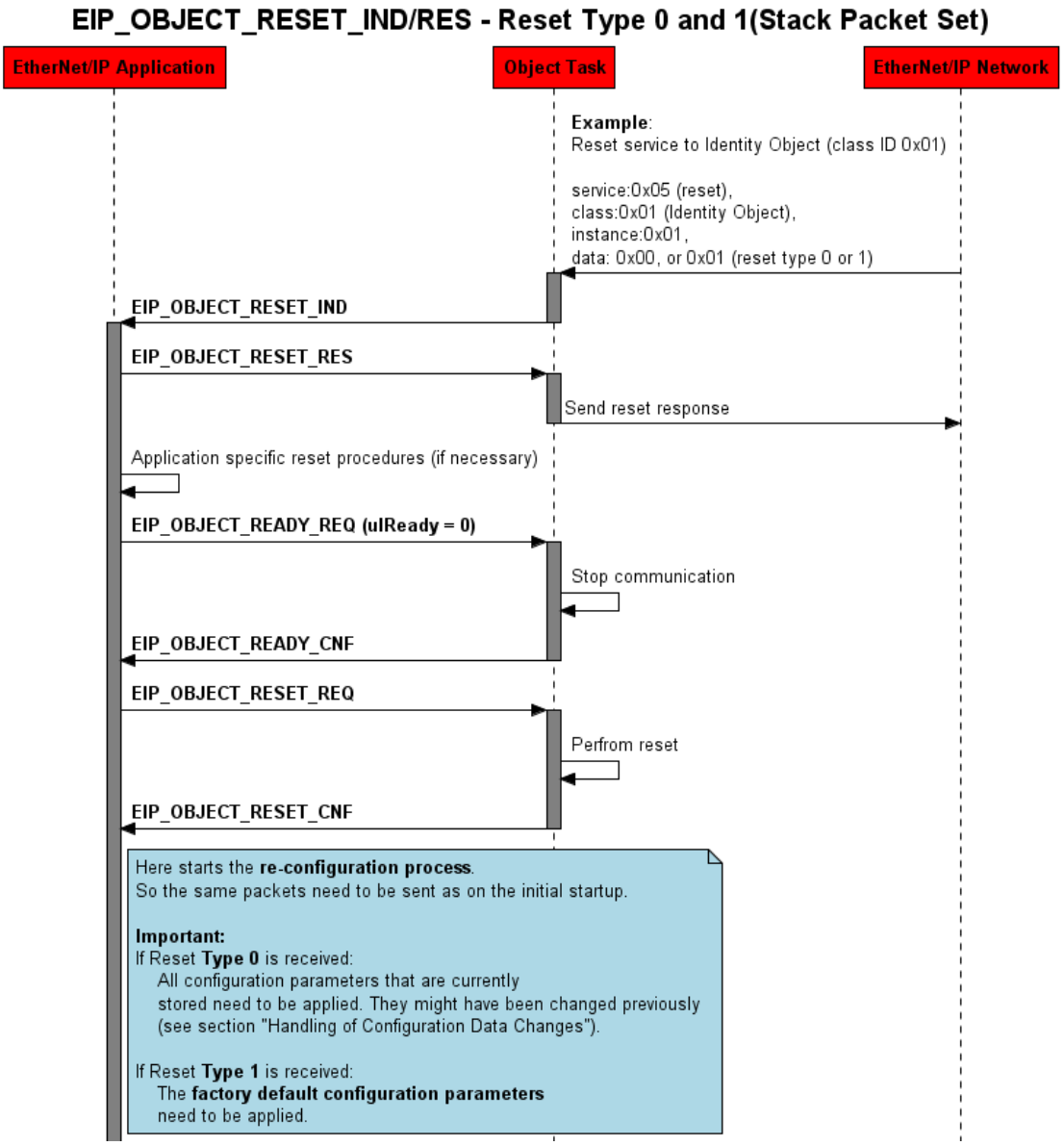

Figure 34: Sequence Diagram for the `EIP_OBJECT_RESET_IND/RES` Packet for the Stack Packet Set

Packet Structure Reference

```
 struct EIP OBJECT RESET IND Ttag
{
  TLR UINT32 ulDataIdx;                /*!< Index of the service */
  TLR_UINT32 ulResetTyp;               /*!< Type of the reset    */
};

struct EIP OBJECT PACKET RESET IND Ttag
{
  TLR_PACKET_HEADER_T     tHead;
  EIP_OBJECT_RESET_IND_T  Data;
};
```

Packet Description

Structure EIP_OBJECT_PACKET_RESET_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	8	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A24	EIP_OBJECT_RESET_IND - Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information
tData - structure EIP_OBJECT_RESET_IND_T			
ulDataIdx	UINT32		Index of the service (host application does not need to evaluate this parameter)
ulResetTyp	UINT32	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. **Note:** Reset type 1 is not supported by default. It needs to be enabled separately using the command EIP_OBJECT_SET_PARAMETER_REQ (see section 6.2.14).

Table 107: EIP_OBJECT_RESET_IND – Reset Request from Bus Indication

Packet Structure Reference

```
typedef struct EIP OBJECT PACKET RESET RES Ttag
{
  TLR PACKET HEADER T     tHead;
} EIP_OBJECT_PACKET_ RESET _RES_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_RESET_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A25	EIP_OBJECT_RESET_RES – Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information

Table 108: EIP_OBJECT_RESET_RES – Response to Indication to Reset Request

6.2.9 EIP_OBJECT_RESET_REQ/CNF - Reset Request

This packet can be sent by the host application in order to initiate a reset of the EtherNet/IP protocol stack. All running connections will be closed and the IP address will be released, so that the device will no longer be accessible via the network until it is re-configured again. Additionally, it can be used to clear a watchdog error.

There are three reset modes that can be used:

- Mode 0 resets the stack. The configuration remains unchanged.
- Mode 1 resets the stack and additionally sets the configuration to the factory default settings, which means the device is not accessible from the network anymore.
- Mode 2 can be set in order to clear a watchdog error (applies only when the Extended Packet Set is used). This mode does not reset the stack. Using this mode is the same as sending the packet EIP_APS_CLEAR_WATCHDOG_REQ/CNF – Clear Watchdog error (see section 6.1.2).

Figure 35 and Figure 36 below display a sequence diagram for the EIP_OBJECT_RESET_REQ/CNF packet in case the host application uses the Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”).

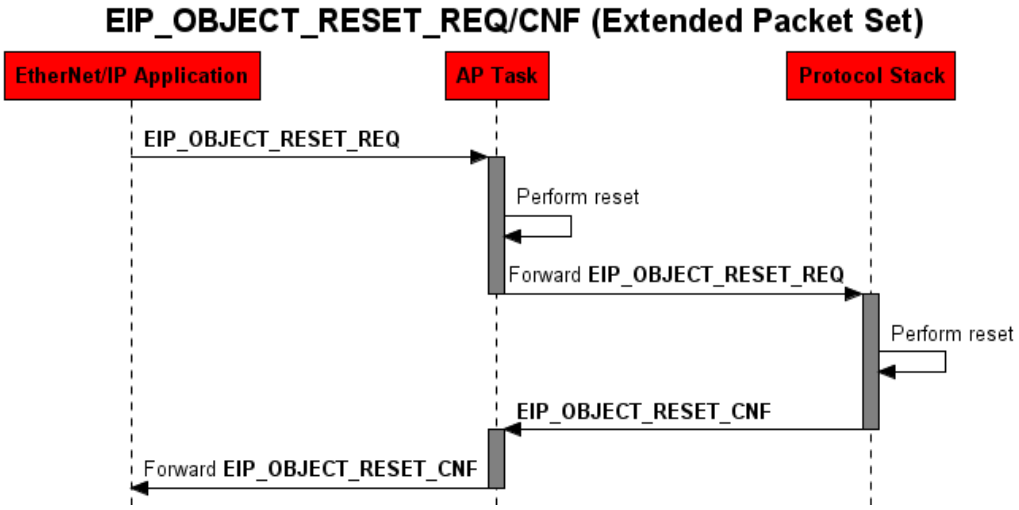

Figure 35: Sequence Diagram for the EIP_OBJECT_RESET_REQ/CNF Packet for the Extended Packet Set

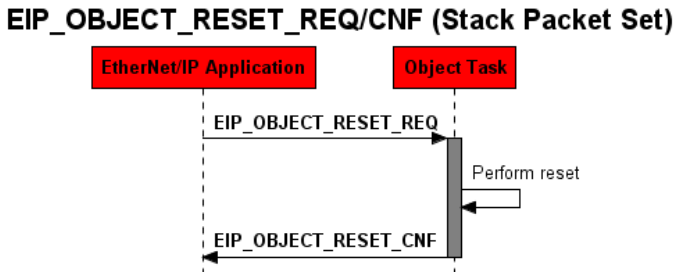

Figure 36: Sequence Diagram for the EIP_OBJECT_RESET_REQ/CNF Packet for the Stack Packet Set

Packet Structure Reference

```
struct EIP OBJECT RESET REQ Ttag
{
  TLR UINT32 ulDataIdx;              /*!< Index of the service */
  TLR UINT32 ulResetMode;            /*!< Mode of the reset    */
};

struct EIP OBJECT PACKET RESET REQ Ttag
{
  TLR PACKET HEADER T     tHead;
  EIP OBJECT RESET REQ T  tData;
};
```

Packet Description

Structure EIP_OBJECT_PACKET_RESET_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ OBJECT_QUE	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	8	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		*See chapter* Status/Error Codes Overview
ulCmd	UINT32	0x00001A26	EIP_OBJECT_RESET_REQ – Response
ulExt	UINT32	0	Reserved
ulRout	UINT32	x	Routing Information
tData - structure EIP_OBJECT_RESET_REQ_T			
ulDataIdx	UINT32		Reserved (set to 0)
ulResetMode	UINT32	0, 2	Mode of the reset 0: Reset is done emulating power cycling of the device (default). Configuration is not touched. 1: Reset is done emulating power cycling of the device and additionally sets configuration back to factory defaults 2: Clears a watch dog error. In case a watchdog error occurred the stack stops at a specific point and does not go into normal operation anymore. Using this type of reset clears this state and the stack starts over again.

Table 109: EIP_OBJECT_RESET_REQ – Bus Reset Request and Confirmation

Packet Structure Reference

```
struct EIP OBJECT PACKET RESET CNF Ttag
{
  TLR PACKET HEADER T     tHead;
/*  EIP OBJECT RESET CNF T  tData;*/
};
```

Packet Description

Structure EIP_OBJECT_PACKET_RESET_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter Status/Error Codes Overview
ulCmd	UINT32	0x00001A27	EIP_OBJECT_RESET_CNF - Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information

Table 110: EIP_OBJECT_RESET_CNF – Response to Indication to Reset Request

6.2.10 EIP_OBJECT_READY_REQ/CNF – Set Ready and Run/Idle State

This packet can be used for changing the state of the host application between *"Ready"* and *"Not ready"* and between *"Run"* and *"Idle"* and vice versa.

Note: Send this packet only when using the Stack Packet Set (see 4.3 "Configuration Using the Packet API").

Parameter	Value	Description
ulReady	0	Sets the host application state to NOT_READY, which means the device will not go into cyclic communication. All incoming Forward_Open frames will be rejected with General Status Code 0x0C (Object State Conflict). Already running connections will be closed.
	1	Sets the host application state to READY, which means the device will now go into cyclic communication if it receives an appropriate Forward_Open frame from a Scanner (Master).
ulRunIdle	0	Sets the run/idle state of the application to "idle". This parameter is only relevant if the device uses T→O assembly instances that are configured to have the 32-Bit run/idle header format as real time format. In that case the run/idle bit in the header will be cleared → set to "Idle"
	1	Sets the run/idle state of the application to "run". This parameter is only relevant if the device uses T→O assembly instances that are configured to have the 32-Bit run/idle header format as real time format. In that case the run/idle bit in the header will be set → set to "run"

Table 111: Ready Request Parameter Values

Figure 37 below displays a sequence diagram for the EIP_OBJECT_READY_REQ/CNF packet.

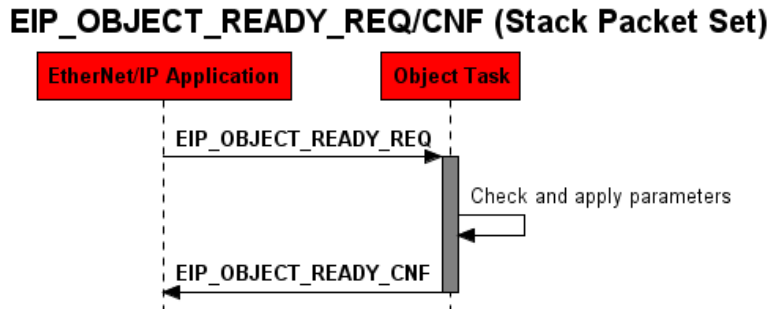

Figure 37: Sequence Diagram for the EIP_OBJECT_READY_REQ/CNF Packet

Packet Structure Reference

```
 struct EIP OBJECT READY REQ Ttag
{
  TLR UINT32 ulReady;           /* Ready state of the application */
  TLR UINT32 ulRunIdle;
};

struct EIP OBJECT PACKET READY REQ Ttag
{
  TLR PACKET HEADER T     tHead;
  EIP OBJECT READY REQ T  tData;
};
```

Packet Description

Structure EIP_OBJECT_PACKET_READY_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	OBJECT_QUE	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	8	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A32	EIP_OBJECT_READY_REQ - Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information
tData - structure EIP_OBJECT_READY_REQ_T			
ulReady	UINT32	0,1	Ready state of the application (starts/stops cyclic communication) 0: Sets application state to "not ready". Cyclic communication is disabled. 1: Sets application state to "ready". Cyclic communication is enabled (see also Table 111)
ulRunIdle	UINT32	0,1	Run/Idle state of the application (sets the run/idle bit in the run/idle header for cyclic I/O connections, if used) 0: Sets run/idle state to "idle". 1: Sets run/idle state to "run" (see also Table 111)

Table 112: EIP_OBJECT_READY_REQ - Request Ready State of the Application

Packet Structure Reference

```
 struct EIP OBJECT PACKET READY CNF Ttag
{
  TLR PACKET HEADER T     tHead;
};
```

Packet Description

Structure EIP_OBJECT_PACKET_READY_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A33	EIP_OBJECT_READY_CNF - Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information

Table 113: EIP_OBJECT_READY_CNF – Confirmation Command for Request Ready State of the Application

6.2.11 EIP_OBJECT_REGISTER_SERVICE_REQ/CNF – Register Service

This packet can be used if the device shall support services that are not directly bound to a CIP object. Usually, services use the CIP addressing format Class→Instance→Atttribute. But if for example TAGs (access data within the device by using strings instead of the normal CIP addressing) shall be supported, no specific object can be addressed.

Therefore, the host application can register a vendor specific service code (see Table 93). If the device then receives this service (sent from a Scanner of Tool) it will be forwarded to the host application via the indication EIP_OBJECT_CL3_SERVICE_IND (section 6.2.4). Again, the indication is only sent if the service does not address an object directly.

Figure 38 and Figure 39 below display a sequence diagram for the EIP_OBJECT_REGISTER_SERVICE_REQ/CNF packet in case the host application uses the Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”).

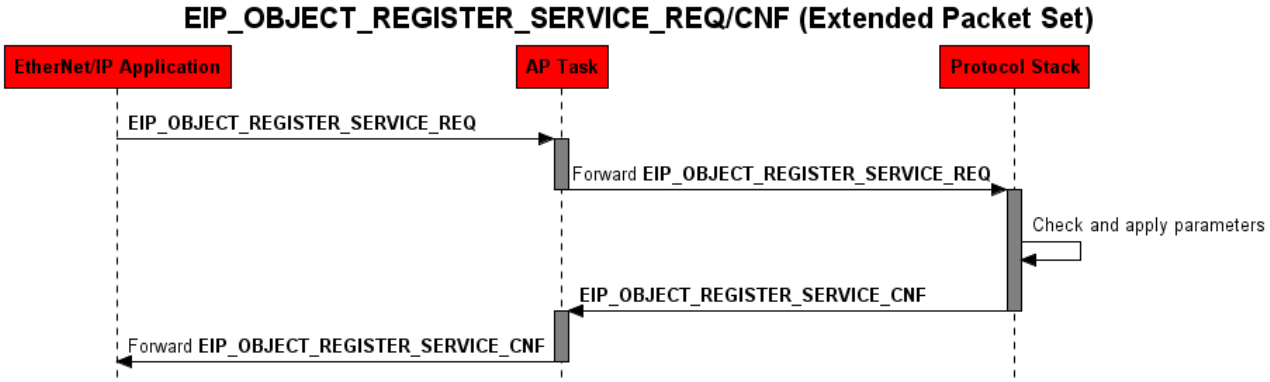

Figure 38: Sequence Diagram for the EIP_OBJECT_REGISTER_SERVICE_REQ/CNF Packet for the Extended Packet Set

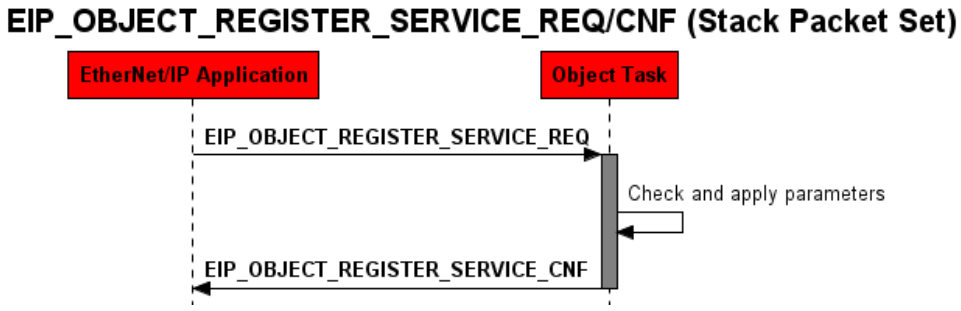

Figure 39: Sequence Diagram for the EIP_OBJECT_REGISTER_SERVICE_REQ/CNF Packet for the Stack Packet Set

Packet Structure Reference

```
/* EIP_OBJECT_REGISTER_SERVICE_REQ */
  struct EIP_OBJECT_REGISTER_SERVICE_REQ_Ttag
  {
      TLR UINT32 ulService;                        /* Service Code */
  };

  /* command for register a new object to the message router */
   struct EIP_OBJECT_PACKET_REGISTER_SERVICE_REQ_Ttag
   {
    TLR PACKET HEADER T           tHead;
    EIP OBJECT REGISTER SERVICE REQ T  tData;
   };
```

Packet Description

Structure EIP_OBJECT_PACKET_REGISTER_SERVICE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	OBJECT_QUE	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	4	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A44	EIP_OBJECT_REGISTER_SERVICE_REQ - Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information
tData - structure EIP_OBJECT_REGISTER_SERVICE_REQ_T			
ulService	UINT32		Vendor specific service code (see Table 93)

Table 114: EIP_OBJECT_READY_REQ - Register Service

Packet Structure Reference

```
struct EIP OBJECT PACKET REGISTER SERVICE CNF Ttag
{
  TLR PACKET HEADER T           tHead;
};
```

Packet Description

Structure EIP_OBJECT_PACKET_REGISTER_SERVICE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A45	EIP_OBJECT_REGISTER_SERVICE_CNF - Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information

Table 115: EIP_OBJECT_READY_CNF – Confirmation Command for Register Service Request

6.2.12 EIP_OBJECT_CONNECTION_CONFIG_IND/RES – Indication of Configuration Data received during Connection Establishment

This indication will be received by the host application when the device receives a Forward_Open frame that addresses a previously registered configuration assembly instance (for more information see section Implicit Messaging on page 32).

Note: This indication will not be received by the host application in case the parameter **EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING** is set (see 6.2.14 EIP_OBJECT_SET_PARAMETER_REQ/CNF – Set Parameter for more information).

The configuration assembly instance can be registered using the packet EIP_OBJECT_AS_REGISTER_REQ/CNF – Register a new Assembly Instance (see section 6.2.5 on page 141).

A common use case could is that the host application needs additional configuration that must be set by the Scanner (PLC). So the PLC can send configuration data within the so called Forward_Open Message. The host application then has the possibility to make arrangements according to that configuration data. If the data holds invalid values or there is not enough or less data received, it is also possible to reject the connection by sending an appropriate error within the response packet.

The content and size of the configuration data is not specified within the CIP specification and can completely be defined by the user (maximum size is 400 bytes).

The parameters of the indication packet have the following meaning:

- ulConnectionId

  This variable contains the connection handle that is used by the protocol stack and must not be changed, when sending the response packet to the stack.

- tConnectionTriad

  This variable contains the Connection Triad that was received with the ForwardOpen request. The “Connection Triad” used in the Connection Manager specification relates to the combination of Connection Serial Number, Originator Vendor ID and Originator Serial Number parameters. In addition, this field holds the variable fConnectionTriadMatch, which indicates whether the Connection Triad matches an existing connection. This is only relevant to devices that support the NULL-ForwardOpen service (see section *Using the Null Forward Open Feature* on page 258).

- ulOTParameter

  This variable contains the connection parameter for the originator-to-target direction of the connection. It follows the rules for network connection parameters as specified in section 3-5.5.1.1 „Network Connection Parameters“ of the document “The CIP Networks Library, Volume 1” (reference #3).

Bits 31-16	Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bits 8-0
Reserved	Redundant Owner	Connection Type		Reserved	Priority		Fixed /Variable	Connection Size (in bytes)

- ulOTRpi

  This variable contains the requested packet interval (RPI) for the originator-to-target direction of the connection. The time is specified in microseconds.

- ulOTConnPoint

  This variable contains the connection point for originator-to-target direction. It should match one of the input assembly instances (flag `EIP_AS_FLAG_READONLY` set) that were registered during the configuration process.

- ulTOParameter

  Similarly to `ulOTParameter`, this variable contains the connection parameter for the target-to-originator direction of the connection. It also follows the rules for network connection parameters as specified in section 3-5.5.1.1 „*Network Connection Parameters*“ of the *“The CIP Networks Library, Volume 1”* document (reference #3) which are explained above at variable `ulOTParameter`.

- ulTORpi

  This variable contains the requested packet interval for the target-to-originator direction.

  The time is specified in microseconds.

- ulTOConnPoint

  This variable contains the connection point for the target-to-originator direction. It should match one of the input assembly instances (flag `EIP_AS_FLAG_READONLY` not set) that were registered during the configuration process.

- ulCfgConnPoint

  This variable contains the connection point for the configuration data. It should match one of the configuration assembly instances (flag `EIP_AS_FLAG_CONFIG` set) that were registered during the configuration process.

- abData

  This byte array includes the configuration data. The size of the data is included in the field `ulLen` in the packet header.

And below display a sequence diagram for the `EIP_OBJECT_CONNECTION_CONFIG_IND/RES` packet in case the host application uses the Extended or Stack Packet Set (see 4.3 “*Configuration Using the Packet API*”).

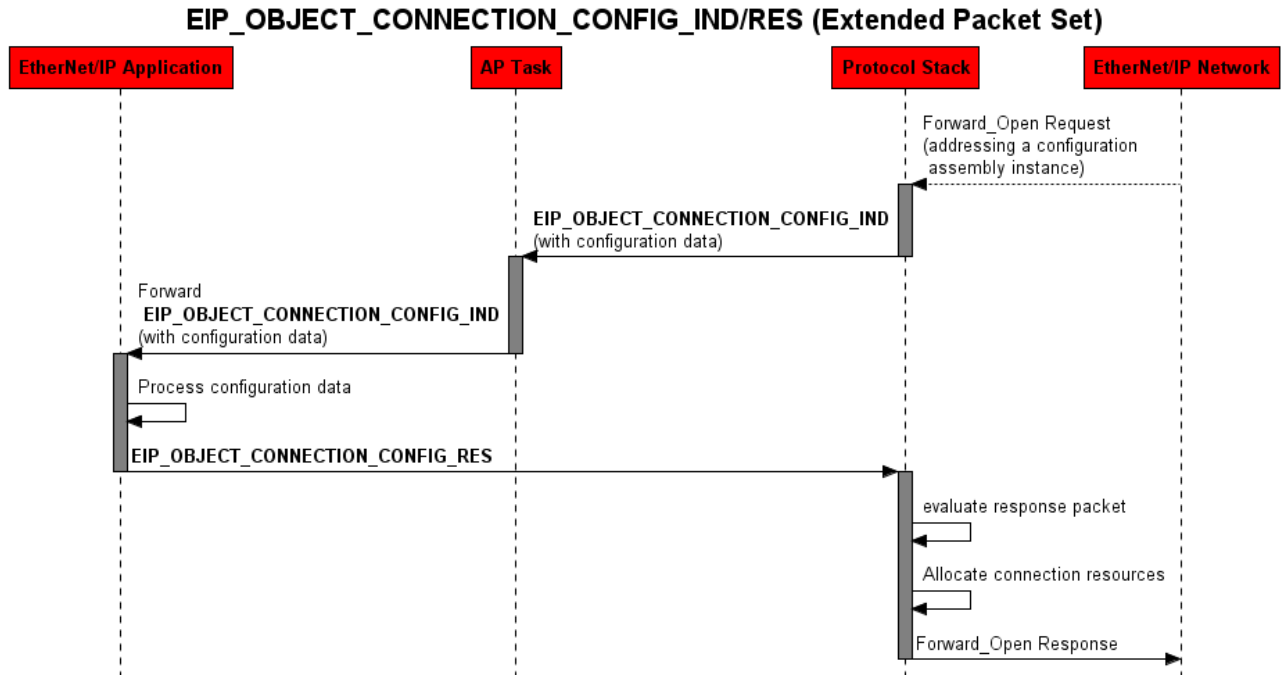

Figure 40: Sequence Diagram for the `EIP_OBJECT_CONNECTION_CONFIG_IND/RES` Packet for the Extended Packet Set

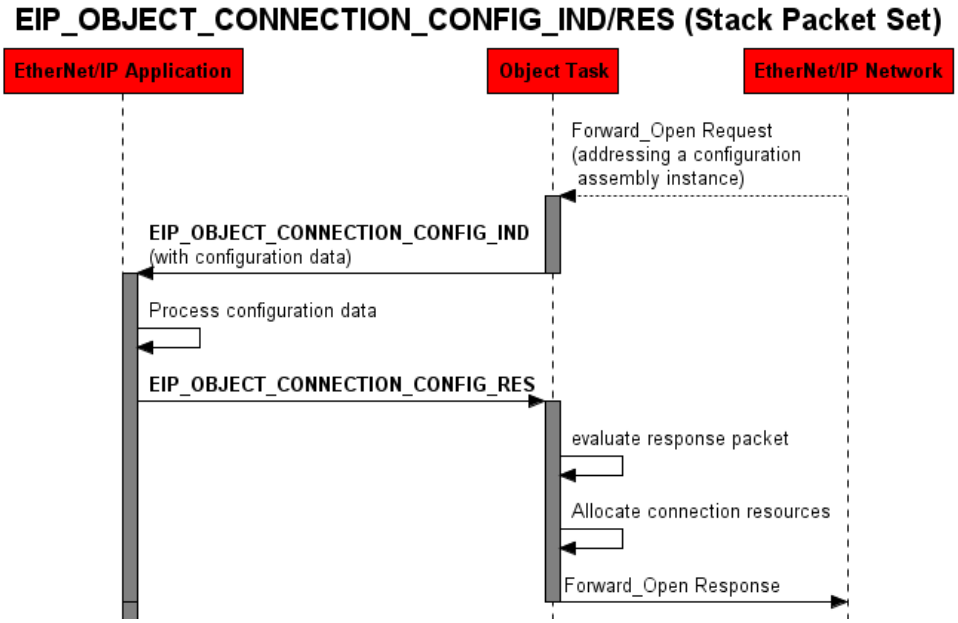

Figure 41: Sequence Diagram for the `EIP_OBJECT_CONNECTION_CONFIG_IND/RES` Packet for the Stack Packet Set

Packet Structure Reference

```
typedef struct EIP OBJECT CONNECTION TRIAD Ttag
{
  /* Connection Triad */
  TLR UINT16    usConnectionSerialNumber;  /*!< Connection serial number from
                                                ForwardOpen frame */
  TLR UINT16    usOriginatorVendorId;      /*!< Originator device vendor ID
                                                from ForwardOpen frame */
  TLR_UINT32    ulOriginatorSerialNumber;  /*!< Originator device serial number
                                                from ForwardOpen frame */
  TLR BOOLEAN32 fConnectionTriadMatch;     /*!< Indicates, whether the above connection triad
                                                matches an existing connection.
                                                For non NULL-ForwardOpen requests:
                                                  - always false
                                                For NULL-ForwardOpen requests:
                                                  - false, if (Connection Triad DOES NOT match)
                                                           i.e. NULL-ForwardOpen is used to
                                                                configure the application
                                                  - true,  if (Connection Triad DOES match)
                                                           i.e. NULL-ForwardOpen is used to
                                                                re-configure the application */
} EIP OBJECT CONNECTION TRIAD T;

typedef struct EIP OBJECT CONNECTION CONFIG IND Ttag
{
  TLR_UINT32                    ulConnectionId;   /* Connection Handle                */
  EIP OBJECT CONNECTION TRIAD T tConnectionTriad; /* "Connection triad" received with
                                                     ForwardOpen request.             */
  TLR UINT32                    ulOTParameter;    /* OT Connection Parameter          */
  TLR UINT32                    ulOTRpi;          /* OT RPI                           */
  TLR_UINT32                    ulOTConnPoint;    /* Produced Connection Point        */

  TLR UINT32                    ulTOParameter;    /* TO Connection Parameter          */
  TLR UINT32                    ulTORpi;          /* TO RPI                           */
  TLR UINT32                    ulTOConnPoint;    /* Consumed Connection Point        */

  TLR_UINT32                    ulCfgConnPoint;   /* Configuration Connection Point   */
  TLR_UINT8                     abData[1];        /* First byte of configuration data */
} EIP OBJECT CONNECTION CONFIG IND T;

#define EIP_OBJECT_CONNECTION_CONFIG_IND_SIZE
                          (sizeof(EIP_OBJECT_CONNECTION_CONFIG_IND_T)-1)

typedef struct EIP OBJECT PACKET CONNECTION CONFIG IND Ttag
{
  TLR PACKET HEADER T                  tHead;
  EIP_OBJECT_CONNECTION_CONFIG_IND_T    tData;
} EIP_OBJECT_PACKET_CONNECTION_CONFIG_IND_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_CONNECTION_CONFIG_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0, 0x20	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	44 + n	Packet Data Length (In Bytes); EIP_OBJECT_CONNECTION_CONFIG_IND_SIZE + n n = Length of configuration Data
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1A40	EIP_OBJECT_CONNECTION_CONFIG_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	×	Routing, do not change
tData - Structure EIP_OBJECT_CONNECTION_CONFIG_IND_T			
ulConnectionId	UINT32		Connection Handle
tConnectionTriad	EIP_OBJECT_CONNECTION_TRIAD_T		Connection triad and matching info
ulOTParameter	UINT32	Bit mask	Originator to Target Parameter
ulOTRpi	UINT32		Originator to Target RPI
ulOTConnPoint	UINT32		Originator to Target Connection Point
ulTOParameter	UINT32		Target to Originator Parameter
ulTORpi	UINT32		Target to Originator RPI
ulTOConnPoint	UINT32		Target to Originator Connection Point
ulCfgConnPoint	UINT32		Configuration Connection Point
abData[]	UINT8		Configuration Data

Table 116: EIP_OBJECT_CONNECTION_CONFIG_IND – Indicate Configuration Data during Connection Establishment

Packet Structure Reference

```
typedef struct EIP OBJECT CONNECTION CONFIG RES Ttag
{
  TLR UINT32    ulConnectionId;    /* Connection Handle                          */
  TLR UINT32    ulGRC;             /* Generic Error Code                         */
  TLR UINT32    ulERC;             /* Extended Error Code                        */
  TLR UINT8     abData[1];         /* Can be used to send Application Reply data */
} EIP OBJECT CONNECTION CONFIG RES T;

#define EIP_OBJECT_CONNECTION CONFIG RES SIZE
                          (sizeof(EIP OBJECT CONNECTION CONFIG RES T)-1)

typedef struct EIP OBJECT PACKET CONNECTION CONFIG RES Ttag
{
  TLR_PACKET_HEADER_T                 tHead;
  EIP OBJECT CONNECTION CONFIG RES T   tData;
} EIP OBJECT PACKET CONNECTION CONFIG RES T;
```

Packet Description

Structure EIP_OBJECT_PACKET_CONNECTION_CONFIG_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue-Handle, unchanged
ulSrc	UINT32	See rules in section 3.2.1	Source Queue-Handle, unchanged
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	12 + n	Packet Data Length (In Bytes); EIP_OBJECT_CONNECTION_CONFIG_RES_SIZE + n n = Length of application reply Data
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1A41	EIP_OBJECT_CONNECTION_CONFIG_RES - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	×	Routing, do not change

Structure EIP_OBJECT_PACKET_CONNECTION_CONFIG_RES_T			Type: Response
tData - Structure EIP_OBJECT_CONNECTION_CONFIG_RES_T			
ulConnectionId	UINT32	x	Unchanged connection handle from indication packet
ulGRC	UINT32		General Error Code (specified in the CIP specification Vol. 1 chapter 3-5.6) 0: success != 0: Forward open will be rejected with this status code **Note:** if the forward open shall be reject with this error code, also ulSta in the packet header must be unequal to 0.
ulERC	UINT32		Extended Error Code (specified in the CIP specification Vol. 1 chapter 3-5.6) 0: Success != 0: Forward open will be rejected with this status code **Note:** if ulERC is unequal to 0, also ulGRC must be unequal to 0. If ulERC is set to something unequal to 0, the below abData field can additionally be used as extended status.
abData[]	UINT8		If ulSta == 0: Can be used as “Application Reply Data” that will be sent with the Forward_Open_Response. Maximum number of bytes for application reply data is 254. If more bytes are sent with this packet, the data will be truncated. Else: Can be used as “Extended Status” data that will be sent with the Forward_Open_Response. The number of Extended status data bytes must not exceed 32 bytes. If more bytes are sent with this packet, the data will be truncated.

Table 117: EIP_OBJECT_CONNECTION_CONFIG_RES – Response command of connection configuration indication

6.2.13 EIP_OBJECT_TI_SET_SNN_REQ/CNF – Set the Safety Network Number for the TCP/IP Interface Object

This service can be used by the host application in order to set the “Safety Network Number” (Attribute 7) within the TCP/IP Interface Object (0xF5). The Safety Network Number is needed when using the EtherNet/IP Adapter protocol stack in CIP Safety applications.

Note: The SNN can also be set by addressing attribute 7 of the TCP/IP Interface Object with the packet EIP_OBJECT_CIP_SERVICE_REQ/CNF – CIP Service Request described in section 6.2.17 on page 196.

Figure 42 and *Figure 43* below display a sequence diagram for the EIP_OBJECT_TI_SET_SNN_REQ/CNF packet in case the host application uses the Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”).

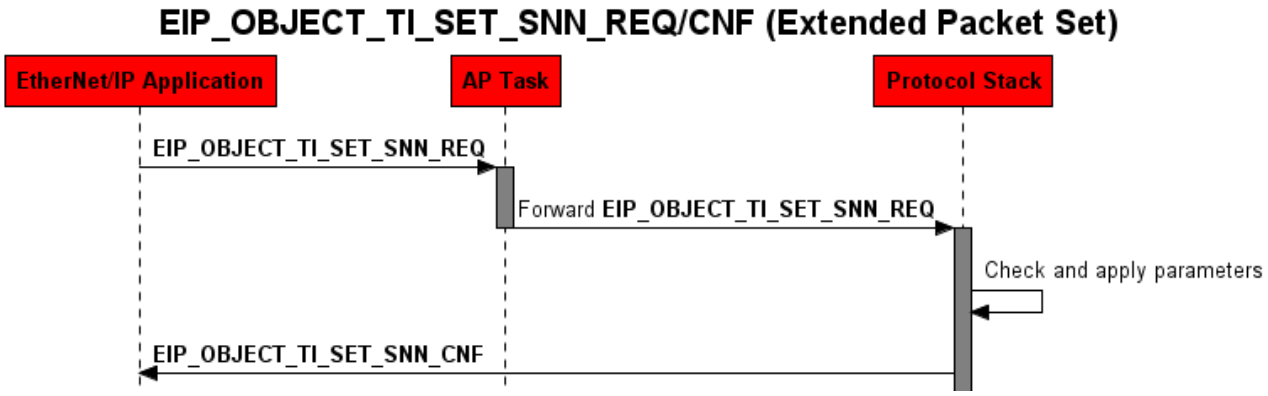

Figure 42: Sequence Diagram for the EIP_OBJECT_TI_SET_SNN_REQ/CNF Packet for the Extended Packet

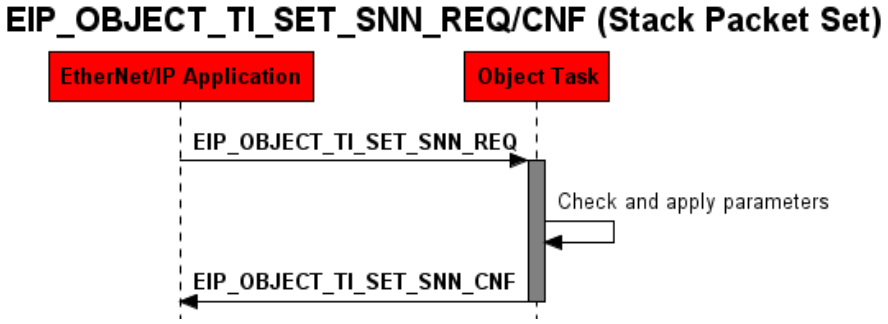

Figure 43: Sequence Diagram for the EIP_OBJECT_TI_SET_SNN_REQ/CNF Packet for the Stack Packet

Packet Structure Reference

```
typedef struct EIP_OBJECT_TI_SET_SNN_REQ_Ttag
{
  TLR UINT8 abSNN[6];

} EIP OBJECT TI SET SNN REQ T;

typedef struct EIP_OBJECT_TI_PACKET_SET_SNN_REQ_Ttag
{
  TLR PACKET HEADER T          tHead;
  EIP OBJECT TI SET SNN REQ T  tData;
} EIP_OBJECT_TI_PACKET_SET_SNN_REQ_T;

#define EIP_OBJECT_TI_SET_SNN_REQ_SIZE    sizeof(EIP_OBJECT_TI_SET_SNN_REQ_T)
```

Packet Description

Structure EIP_OBJECT_TI_PACKET_SET_SNN_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0, 0x20	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	6	Packet Data Length (In Bytes); EIP_OBJECT_TI_SET_SNN_REQ_SIZE
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1AF0	EIP_OBJECT_TI_SET_SNN_REQ - Command
ulExt	UINT32	0, 0x20	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulRout	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
tData - structure EIP_OBJECT_TI_SET_SNN_REQ_T			
abSNN[6]	6*UINT8		Safety Network Number

Table 118: EIP_OBJECT_TI_SET_SNN_REQ – Set the Safety Network Number of the TCP/IP Interface Object

Packet Structure Reference

```
typedef struct EIP_OBJECT_TI_PACKET_SET_SNN_CNF_Ttag
{
  TLR PACKET HEADER T          tHead;
} EIP OBJECT TI PACKET SET SNN CNF T;

#define EIP OBJECT TI SET SNN CNF SIZE    0
```

Packet Description

Structure EIP_OBJECT_TI_PACKET_SET_SNN_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination Queue Reference
ulSrcId	UINT32	See rules in section 3.2.1	Source Queue Reference
ulLen	UINT32	0	Packet Data Length (in Bytes) EIP_OBJECT_TI_SET_SNN_CNF_SIZE
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1AF1	EIP_OBJECT_TI_SET_SNN_CNF - Command
ulExt	UINT32	0	Extension, reserved
ulRout	UINT32	×	Routing, do not change

Table 119: EIP_OBJECT_TI_SET_SNN_CNF – Confirmation command of set safety network number request

6.2.14 EIP_OBJECT_SET_PARAMETER_REQ/CNF – Set Parameter

This packet can be used to activate special options and behavior of the protocol stack.

Table 120 gives an overview of all possible parameters:

Parameter Flags – ulParameterFlags

Bit	Description
0	**EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING** Enables forwarding of Forward_Open and Forward_Close frames to the user application task. **Forward_Open frames:** If set (1), all Forward_Open frames that address the assembly object will be forwarded to the host application via the packet EIP_OBJECT_FWD_OPEN_FWD_IND (6.2.21). **Note:** If set (1), the host application will no longer received the indication packet EIP_OBJECT_CONNECTION_CONFIG_IND (6.2.12). If a configuration assembly instance is used, the configuration data for that assembly instance is part of the Forward_Open frame (included in the connection path). In that case, the host application is responsible to extract the configuration data out of the Forward_Open frame. If not set (0), the Forward_Open will not be forwarded. **Forward_Close frames:** If set (1), all Forward_Close frames that address the assembly object will be forwarded via the packet EIP_OBJECT_FWD_CLOSE_FWD_IND (6.2.23). If not set (0), the Forward_Close will not be forwarded.
1	**EIP_OBJECT_PRM_APPL_TRIG_NO_RPI** Disables the RPI timer for “Application Object Triggered” and “Change of State” data production. Using the trigger mechanism "Application Object Triggered" the user application is able to define at what time the I/O data is being produced on the network. If the host application does not trigger data production within the RPI time, the data will be produced automatically by the RPI timeout in order to avoid connection timeouts. This is the behavior the CIP specification describes. However, some applications need to turn off the mentioned RPI timer to avoid double data production. If set, the RPI timer will be turned off for all connections using the “Application Object Triggered” or “Change of State” mechanism. If not set, the RPI timer is used as described in the CIP specification. **Note:** Also have a look at bit 5 (**EIP_OBJECT_PRM_SUPPORT_AOT_COS_DATA_PRODUCTION**). Using this enables the host application (LFW only) to trigger IO frames for different connections independent of each other.
2	**EIP_OBJECT_PRM_SUPPORT_SNN** This flag enables attribute 7 (Safety Network Number) of the TCP/IP-Interface object as defined in the EtherNet/IP CIP Specification (Volume 2 Edition 1.9 chapter 5-3.2.2). Additionally, the value of this attribute can be set using the command EIP_OBJECT_TI_SET_SNN_REQ or EIP_OBJECT_CIP_SERVICE_REQ. Attribute 7 can also be activated using the packet EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ. **Note:** Activation of the SSN implicitly deactivates the support of the identity object’s reset service. All reset services that address the Identity Object will then be rejected with general status code 0x08 (Service not supported).
3	**EIP_OBJECT_PRM_ACTIVATE_IDENTITY_RESET_TYPE_1** This flag enables the additional reset type 1 of the identity object reset service (for more information see section 6.2.8 “EIP_OBJECT_RESET_IND/RES – Indication of a Reset Request from the network” on page 157). The default reset type is 0. **Default type 0:** This type is supported as default. It emulate as closely as possible cycling power. **Additional type 1:** Return as closely as possible to the factory default configuration. Then, emulate cycling power as closely as possible. **Note:** Reset type 1 is only possible when configuration is not done via data base and there is a registered application available. The host application needs to handle this type of reset by itself (setting configuration back to factory default). The application can determine the requested reset type within the EIP_OBJECT_RESET_IND packet in the field ulResetTyp.

Bit	Description
4	**EIP_OBJECT_PRM_HARDWARE_CONFIGURABLE** This flag affects attribute #2 of the TCP/IP Interface object (class ID 0xF5) If set (1), the hardware configurable flag within this attribute is set. If not set, the hardware configurable flag within this attribute is not set.
5	**EIP_OBJECT_PRM_SUPPORT_AOT_COS_DATA_PRODUCTION** This flag enables the "Change of State" (COS) and "Application Object Trigger" (AOT) feature. It allows the host application to trigger the sending of output assemnbly data for different COS/AOT connections independently of each other (see 6.2.14.1 “Handling of connections of type “Application Object Trigger” or “Change of State””). **Note:** if this feature is used, make sure to first send this SET_PARAMETER_REQ and afterwards register all assembly instances at the stack.
6	Reserved
7	**EIP_OBJECT_PRM_FORWARD_CIP_SERVICE_FOR_UNKNOWN_ASSEMBLY_TO_HOST** Setting this flag the host application will receive all CIP service request to assembly instances that are not registered (indication is done using command EIP_OBJECT_CL3_SERVICE_IND).
8	**EIP_OBJECT_PRM_NULL_FORWARD_OPEN_SUPPORT** Activates/Deactivates the support of the NULL-ForwardOpen feature (see also 9.5.3 “Using the Null Forward Open Feature”). If set (1), the NULL-ForwardOpen feature is activated. If not set (0), the NULL-ForwardOpen feature is deactivated.
9	**EIP_OBJECT_PRM_APPLICATION_CONTROLS_IDENTITY_STATE_ATTRIBUTE** If set (1), the state attribute 8 of the Identity object must be controlled by the host application. If not set (0), the state attribute 8 of the Identity object is controlled by the EtherNet/IP stack itself (default). **Note:** Care must be taken when using this functionality. Usually, this is not necessary to activate this, but there are types of applications that might require write access to this attribute (e.g. CIP Safety applications). When enabling write access, the EtherNet/IP stack does not handle this attribute anymore. The application is responsible of providing the correct attribute values depending on the current device state. **Note:** The designer of the application must decide whether or not it needs this feature. Enabling and after some time disabling the write access must be avoided as this might lead to invalid state attribute values. **Note:** When using this functionality, the host application has additionally to care about the current module status of the device (module status LED). The host application must send the packet `EIP_APS_SET_MODULE_STATUS_REQ` in order to control the module status LED. The firmware will not control the moduel status LED as soon as this functionality is activated.
10 - 31	**Reserved** Must be set to 0

Table 120: `EIP_OBJECT_SET_PARAMETER_REQ` – Flags

Figure 44 and Figure 45 below display a sequence diagram for the `EIP_OBJECT_SET_PARAMETER_REQ/CNF` packet in case the host application uses the Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”).

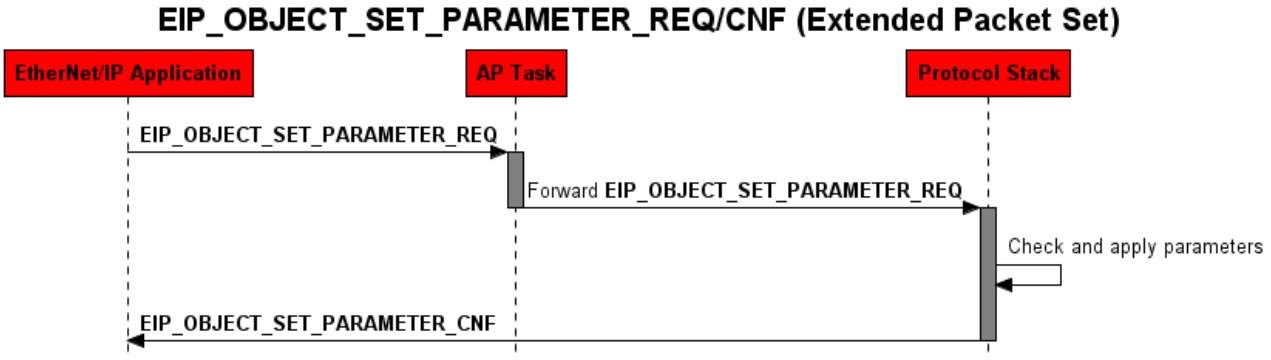

Figure 44: Sequence Diagram for the `EIP_OBJECT_SET_PARAMETER_REQ/CNF` Packet for the Extended Packet

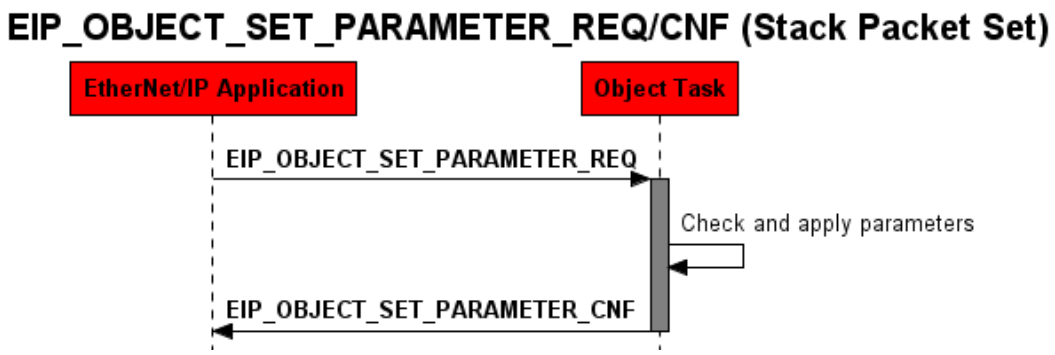

Figure 45: Sequence Diagram for the `EIP_OBJECT_SET_PARAMETER_REQ/CNF` Packet for the Stack Packet

Packet Structure Reference

```
#define EIP_OBJECT_PRM_FWRD_OPEN_CLOSE_FORWARDING                          0x00000001
#define EIP_OBJECT_PRM_APPL_TRIG_NO_RPI                                    0x00000002
#define EIP_OBJECT_PRM_SUPPORT_SNN                                         0x00000004
#define EIP_OBJECT_PRM_ACTIVATE_IDENTITY_RESET_TYPE_1                      0x00000008
#define EIP_OBJECT_PRM_HARDWARE_CONFIGURABLE                               0x00000010
#define EIP_OBJECT_PRM_SUPPORT_AOT_COS_DATA_PRODUCTION                     0x00000020
#define EIP_OBJECT_PRM_FORWARD_CIP_SERVICE_FOR_UNKNOWN_ASSEMBLY_TO_HOST    0x00000080

  typedef struct EIP_OBJECT_SET_PARAMETER_REQ_Ttag
  {
    TLR UINT32 ulParameterFlags;
  } EIP OBJECT SET PARAMETER REQ T;

  #define EIP_OBJECT_SET_PARAMETER_REQ_SIZE
       sizeof(EIP_OBJECT_SET_PARAMETER_REQ_T)

  typedef struct EIP OBJECT PACKET SET PARAMETER REQ Ttag
  {
    TLR_PACKET_HEADER_T             tHead;
    EIP OBJECT SET PARAMETER REQ T  tData;
  }EIP_OBJECT_PACKET_SET_PARAMETER_REQ_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_SET_PARAMETER_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	EIP_OBJECT_SET_PARAMETER_REQ_SIZE Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001AF2	EIP_OBJECT_SET_PARAMETER_REQ – Command
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information
tData - structure EIP_OBJECT_SET_PARAMETER_REQ_T			
ulParameterFlags	UINT32		See *Table 120: EIP_OBJECT_SET_PARAMETER_REQ –*

Table 121: EIP_OBJECT_SET_PARAMETER_REQ – Set Parameter Request Packet

Packet Structure Reference

```
  typedef struct EIP OBJECT PACKET SET PARAMETER CNF Ttag
  {
    TLR PACKET HEADER T          tHead;
  } EIP OBJECT PACKET SET PARAMETER CNF T;

#define EIP_OBJECT_SET_PARAMETER_CNF_SIZE  0
```

Packet Description

Structure EIP OBJECT_PACKET_SET_PARAMETER_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination Queue Reference
ulSrcId	UINT32	See rules in section 3.2.1	Source Queue Reference
ulLen	UINT32	0	EIP_OBJECT_SET_PARAMETER_CNF_SIZE Packet Data Length (In Bytes)
ulId	UINT32		Packet Identification As Unique Number
ulSta	UINT32		See *Table 5: EIP_OBJECT_SET_PARAMETER_CNF – Packet Status/Error*
ulCmd	UINT32	0x00001AF3	EIP_OBJECT_SET_PARAMETER_CNF– Command
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information

Table 122: EIP_OBJECT_SET_PARAMETER_CNF – Set Parameter Confirmation Packet

Packet Status/Error

Definition / (Value)	Description
TLR_S_OK (0x00000000)	Status ok
TLR_E_INVALID_PARAMETER (0xC0000009)	Invalid Parameter Flag

Table 123: EIP_OBJECT_SET_PARAMETER_CNF – Packet Status/Error

6.2.14.1 Handling of connections of type “Application Object Trigger” or “Change of State”

Usually, the sending of process data messages is completely managed by the protocol stack. The protocol stack sends process data messages according to the configured RPI (Request Packet Rate) that is negotiated during connection establishment. This interval is handled by the internal “Transmission Trigger Timer” of the protocol stack. This applies to all connection trigger types (cyclic, change of state (COS) and application object triggered (AOT)).

In case, the connection trigger type is AOT or COS, the host application can additionally influence the sending of process data messages. Every time the host application updates the output process data, the protocol stack sends a process data message right away independent of the transmission trigger timer’s state.

This requires that the host application can update the process data of different connections independently of each other. In the LOM use case (Linkable Object Module, see 4.3 “Configuration Using the Packet API”), this is easily possible as the connection’s process data is updated for each assembly instance separately anyway. In the LFW use case this is not easily possible as the host application provides all output process data (assembly instance data) at once to the protocol stack via the DPM’s output data area (see [1]). Therefore, the following only applies to the LFW use case.

In order to be able to update the output process data for different assembly instances separately via the DPM, the parameter `EIP_OBJECT_PRM_SUPPORT_AOT_COS_DATA_PRODUCTION` must be configured (see 6.2.14).

Figure 46 illustrates the DPM area for registered assemblies in case the flag `EIP_OBJECT_PRM_SUPPORT_AOT_COS_DATA_PRODUCTION` is **not** set. All assembly instances are registered by the host application. So, the information about size and offset of each assembly instance within the DPM is known (also see 6.2.5 “`EIP_OBJECT_AS_REGISTER_REQ/CNF` – Register a new Assembly Instance”).

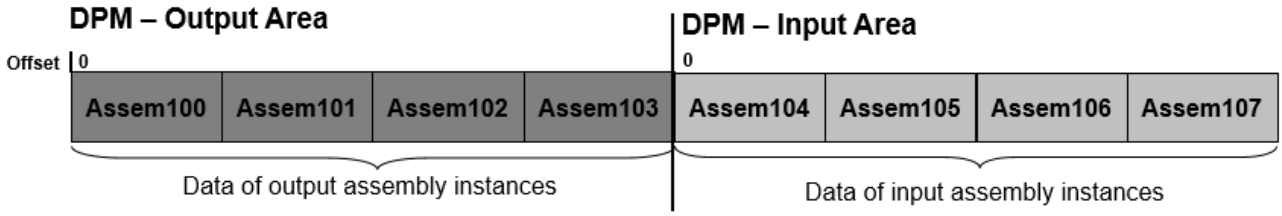

Figure 46: DPM output area for EtherNet/IP, AOT and COS data production not enabled.

When setting flag `EIP_OBJECT_PRM_SUPPORT_AOT_COS_DATA_PRODUCTION` the DPM output area is extended by an additional “Assembly Update Bit List”. This bit list is always 16 bytes long and starts right behind the last assembly instance data at a 32-bit aligned offset (Figure 47).

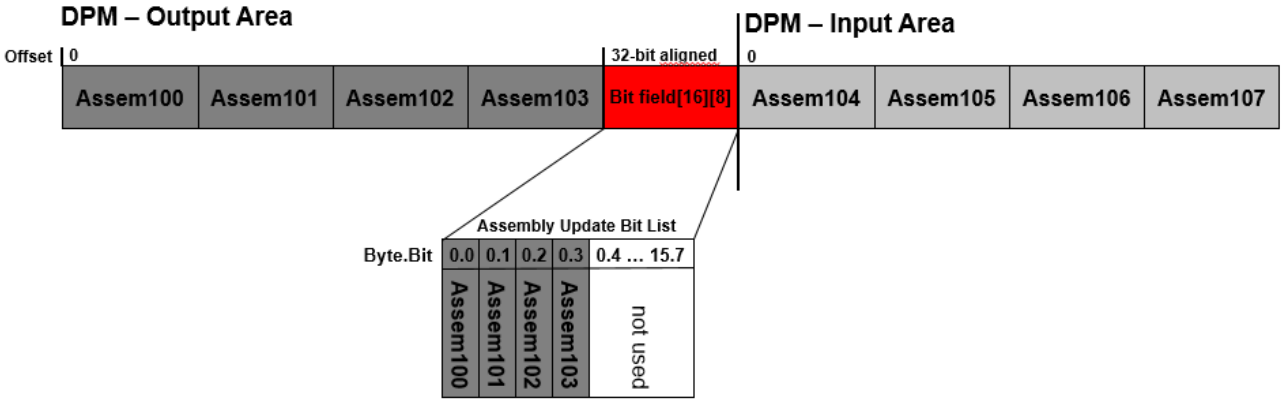

Figure 47: DPM output area for EtherNet/IP, AOT and COS data production enabled.

Each bit of this bit field corresponds to exactly one assembly instance data. The mapping of assembly instance data to the bits wthin the bist list is done by taking the order of the assembly instances in the area into acaount.

```
Assem100 → Bitfield[0][0]
Assem101 → Bitfield[0][1]
Assem102 → Bitfield[0][2]
Assem103 → Bitfield[0][3]
```

In order to trigger an output process data message for a specific assembly instance that is participating in a connection of type AOT or COS, the host application now has to set (‘1’) the corresponding bit in the bit field when writing the output image. In order not to trigger the IO message, the host application has to clear (‘0’) the corresponding bit. This applies only to assembly instances that are used in a AOT or COS connection. The message triggering of all other assemblies (that are not of type “AOT” or “COS”) is done by the protocol stack corresponding to the connection’s RPI interval as described above. So the bit field does not influence assembly instances that are used in a connection of type “cyclic”.

The originator (e.g. PLC) of a specific connection defines the the trigger type (Cyclic, AOT, COS), that shall be used. Therefore, the trigger type can change during runtime. The information about the currently used trigger type is indicated to the host application during connection establishment via the packet `EIP_OBJECT_CONNECTION_IND (0x00001A2E)`. This way the host application knows for which assembly instances it must handle the corresponding bits within the update bit list.

6.2.15 EIP_OBJECT_AS_TRIGGER_TYPE_IND/RES – Indication of the currently used trigger type

This indication notifies the host application of the trigger type of the connection that a particular assembly instance is currently participating in. This indication is sent by the protocol stack only if the parameter EIP_OBJECT_PRM_SUPPORT_AOT_COS_DATA_PRODUCTION was set in advance (see section *EIP_OBJECT_SET_PARAMETER_REQ/CNF – Set Parameter* on page 181). Trigger type indications are only sent for assembly instances used for data production (output / T2O).

Figure 48 below displays a sequence diagram for the EIP_OBJECT_AS_TRIGGER_TYPE_IND packet.

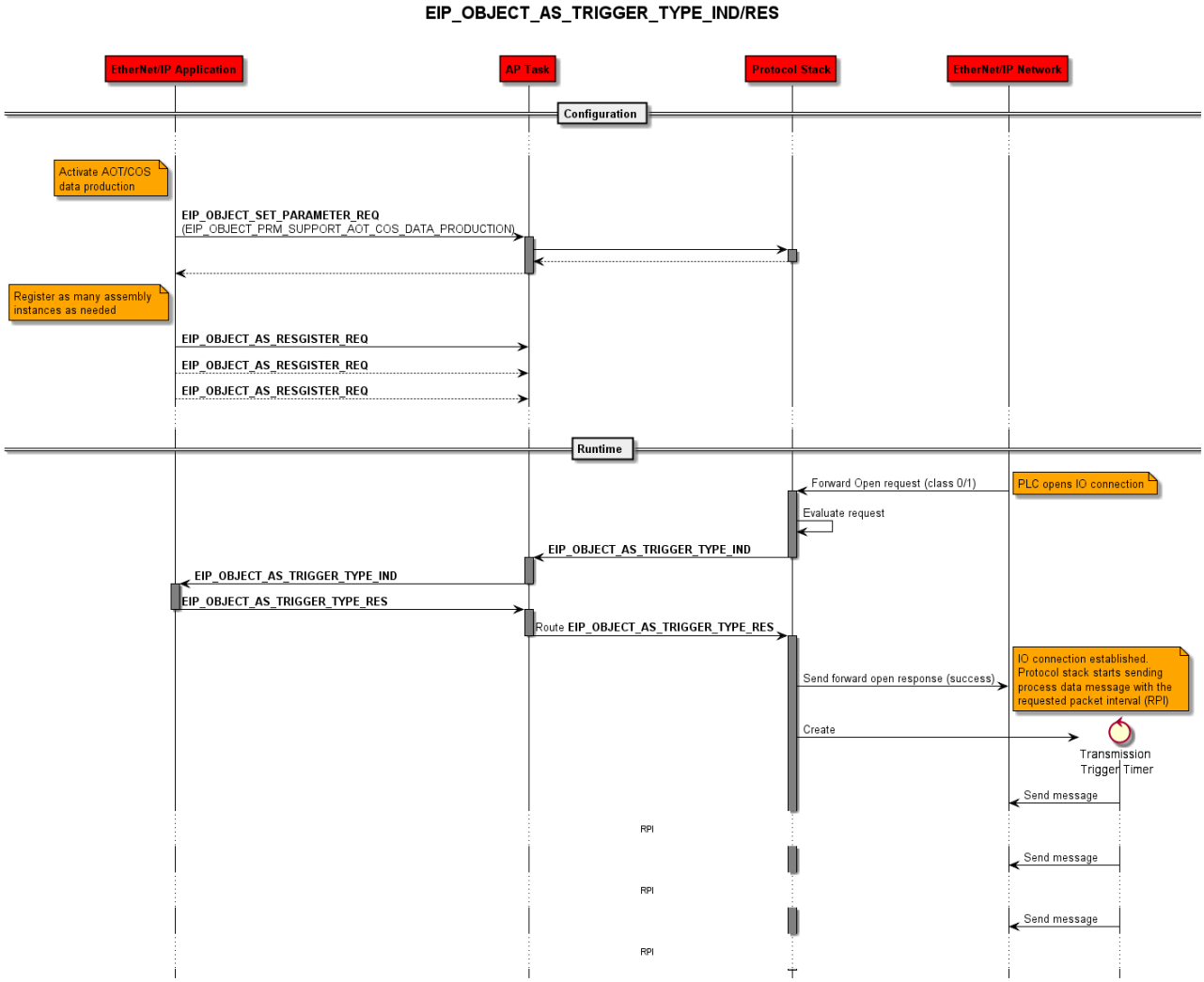

Figure 48: Sequence Diagram for the EIP_OBJECT_AS_TRIGGER_TYPE_IND/RES Packet

Packet Structure Reference

```
#define   EIP_AS_TRIGGER_TYPE_CYCLIC               0x00000001
#define   EIP_AS_TRIGGER_TYPE_CHANGE_OF_STATE      0x00000002
#define   EIP_AS_TRIGGER_TYPE_APPL_OBJ_TRIGGERED   0x00000003

typedef struct EIP_OBJECT_AS_TRIGGER_TYPE_IND_Ttag
{
  TLR_UINT32      ulInstance;
  TLR UINT32      ulDPMOffset;
  TLR UINT32      ulSize;
  TLR UINT32      ulFlags;
  TLR UINT32      ulTriggerType;

} EIP_OBJECT_AS_TRIGGER_TYPE_IND_T;

typedef struct EIP_OBJECT_PACKET_AS_TRIGGER_TYPE_IND_Ttag
{
  TLR_PACKET_HEADER_T               tHead;
  EIP_OBJECT_AS_TRIGGER_TYPE_IND_T  tData;
}EIP OBJECT PACKET AS TRIGGER TYPE IND T;

#define EIP_OBJECT_AS_TRIGGER_TYPE_IND_SIZE     sizeof(EIP_OBJECT_AS_TRIGGER_TYPE_IND_T)
```

Packet Description

Structure EIP_OBJECT_PACKET_AS_TRIGGER_TYPE_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32		Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	20	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001AFE	EIP_OBJECT_AS_TRIGGER_TYPE_IND - Command / Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information
tData - structure EIP_OBJECT_AS_TRIGGER_TYPE_IND_T			
ulInstance	UINT32		CIP assembly instance number
ulDPMOffset	UINT32		DPM Offset of the assembly instance
ulSize	UINT32		Size of assembly instance in number of bytes
ulFlags	UINT32		Assembly flags as provided when instance was registered
ulTriggerType	UINT32		Tigger type currently used for this assembly instance

Table 124: EIP_OBJECT_AS_TRIGGER_TYPE_IND – Assembly Trigger Type Indication

Packet Structure Reference

```
typedef struct EIP OBJECT PACKET AS TRIGGER TYPE RES Ttag
{
  TLR PACKET HEADER T     tHead;
} EIP_OBJECT_PACKET_AS_TRIGGER_TYPE_RES_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_AS_TRIGGER_TYPE_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process.
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001AFF	EIP_OBJECT_AS_TRIGGER_TYPE_RES – Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information

Table 125: EIP_OBJECT_AS_TRIGGER_TYPE_RES – Assembly Trigger Type Response

6.2.16 `EIP_OBJECT_CFG_QOS_REQ/CNF` – Configure the QoS Object

This packet can be sent by the host application in order to activate and configure the Quality of Service (QoS) object (Class ID 0x48) within the EtherNet/IP Adapter protocol stack.

Important: Sending this packet is mandatory if you want to use DLR in your EtherNet/IP application.

Important: This packet must always be send before sending the packet `TCPIP_IP_CMD_SET_CONFIG_REQ`.

Figure 49 and Figure 50 below display a sequence diagram for the `EIP_OBJECT_CFG_QOS_REQ/CNF` packet in case the host application uses the Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”).

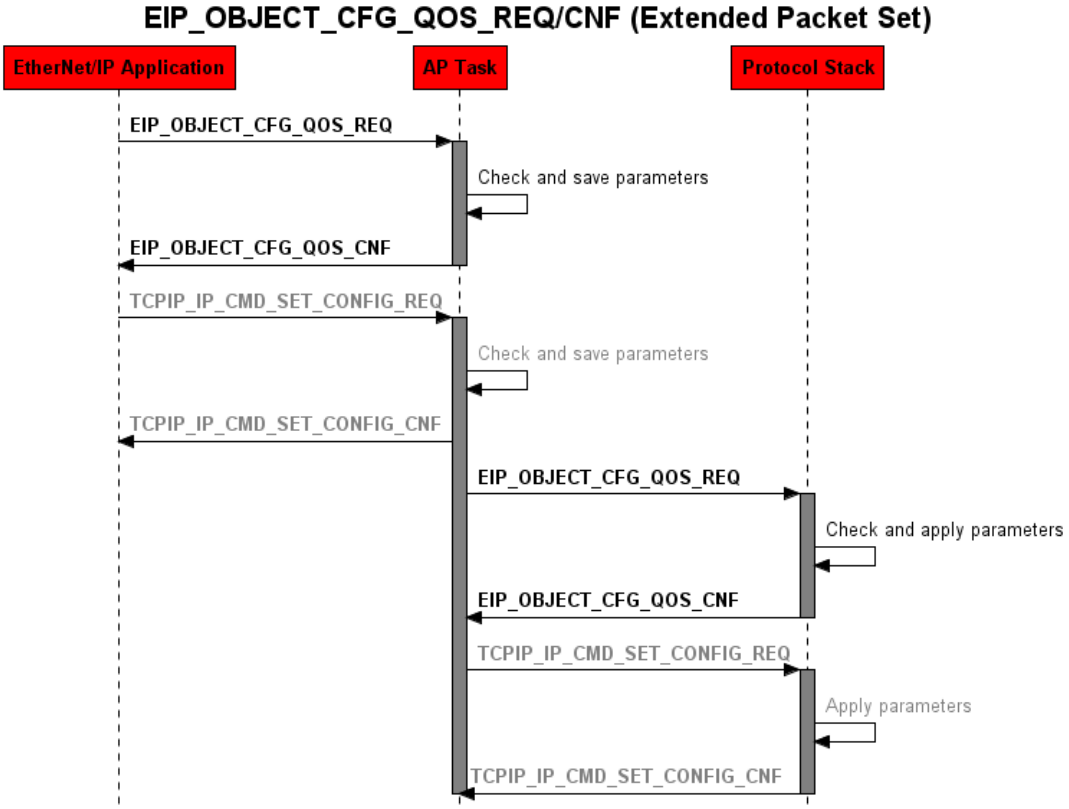

Figure 49: Sequence Diagram for the `EIP_OBJECT_CFG_QOS_REQ/CNF` Packet for the Extended Packet Set

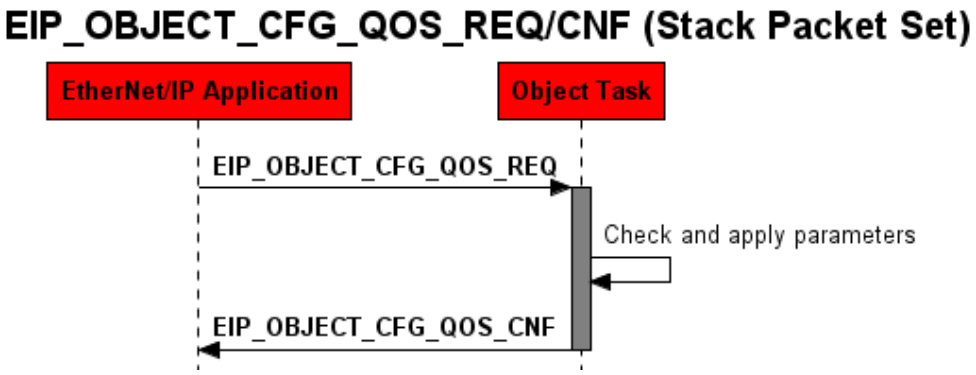

Figure 50: Sequence Diagram for the `EIP_OBJECT_CFG_QOS_REQ/CNF` Packet for the Stack Packet Set

Packet Structure Reference

```
#define EIP OBJECT QOS FLAGS ENABLE          0x00000001
#define EIP_OBJECT_QOS_FLAGS_DEFAULT         0x00000002
#define EIP OBJECT QOS FLAGS DISABLE 802 1Q  0x00000004

typedef struct EIP OBJECT CFG QOS REQ Ttag
{
  TLR UINT32    ulQoSFlags;
  TLR_UINT8     bTag802Enable;
  TLR UINT8     bDSCP PTP Event;
  TLR UINT8     bDSCP PTP General;
  TLR UINT8     bDSCP Urgent;
  TLR UINT8     bDSCP Scheduled;
  TLR UINT8     bDSCP High;
  TLR_UINT8     bDSCP_Low;
  TLR UINT8     bDSCP Explicit;
} EIP OBJECT CFG QOS REQ T;

/* command for register a new object to the message router */
typedef struct EIP_OBJECT_PACKET_CFG_QOS_REQ_Ttag
{
  TLR PACKET HEADER T           tHead;
  EIP OBJECT CFG QOS REQ T      tData;
} EIP OBJECT PACKET CFG QOS REQ T;

#define EIP_OBJECT_CFG_QOS_REQ_SIZE        sizeof(EIP_OBJECT_CFG_QOS_REQ_T)
```

Packet Description

Structure EIP_OBJECT_PACKET_CFG_QOS_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20	Destination Queue Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	See rules in section 3.2.1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	12	EIP_OBJECT_CFG_QOS_REQ_SIZE Packet Data Length (In Bytes)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A42	EIP_OBJECT_CFG_QOS_REQ – Command
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information
tData - Structure EIP_OBJECT_CFG_QOS_REQ_T			
ulQoSFlags	UINT32	0…7	Enables or disables sending 802.1Q frames on CIP messages **Bit 0:** (EIP_OBJECT_QOS_FLAGS_ENABLE) Activates the QoS object **Bit 1:** (EIP_OBJECT_QOS_FLAGS_DEFAULT) **DO NOT USE, DEPRECATED!!!** **Bit 2:** (EIP_OBJECT_QOS_FLAGS_DISABLE_802_1Q) If set (1), the stack deactivates attribute 1 of the QoS object. So, the 802.1q functionality (VLAN tagging) will not be supported.
bTag802Enable	UINT8	0,1	Enables or disables sending 802.1Q frames on CIP messages 0: 802.1Q is disabled (default) 1: 802.1Q is enabled
bDSCP_PTP_Event	UINT8	0	Not used
bDSCP_PTP_General	UINT8	0	Not used
bDSCP_Urgent	UINT8	0…63	DSCP value for CIP transport class 0/1 Urgent priority messages Default: 55
bDSCP_Scheduled	UINT8	0…63	DSCP value for CIP transport class 0/1 Scheduled priority messages Default: 47
bDSCP_High	UINT8	0…63	DSCP value for CIP transport class 0/1 High priority messages Default: 43

Structure EIP_OBJECT_PACKET_CFG_QOS_REQ_T			Type: Request
bDSCP_Low	UINT8	0…63	DSCP value for CIP transport class 0/1 low priority messages Default: 31
bDSCP_Explicit	UINT8	0…63	DSCP value for CIP explicit messages (messages with transport class 2/3 and UCMM messages) Default: 27

Table 126: EIP_OBJECT_CFG_QOS_REQ – Enable Quality of Service Object

Packet Structure Reference

```
typedef struct EIP_OBJECT_PACKET_CFG_QOS_CNF_Ttag
{
  TLR_PACKET_HEADER_T           tHead;
} EIP_OBJECT_PACKET_CFG_QOS_CNF_T;

#define EIP_OBJECT_CFG_QOS_CNF_SIZE        0
```

Packet Description

Structure EIP OBJECT_PACKET_CFG_QOS_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	See rules in section 3.2.1	Destination Queue Reference
ulSrcId	UINT32	See rules in section 3.2.1	Source Queue Reference
ulLen	UINT32	0	EIP_OBJECT_CFG_QOS_CNF_SIZE Packet Data Length (In Bytes)
ulId	UINT32		Packet Identification As Unique Number
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x00001A43	EIP_OBJECT_CFG_QOS_CNF – Response
ulExt	UINT32		Reserved
ulRout	UINT32		Routing Information

Table 127: EIP_OBJECT_CFG_QOS_CNF – Confirmation Command for Unregister Application

6.2.17 EIP_OBJECT_CIP_SERVICE_REQ/CNF – CIP Service Request

This packet can be used to access a CIP object within the EtherNet/IP Stack.The service to be performed is selected by setting the parameter ulService of the request packet. What attributes of an object can be accessed and what services are available for the objects please see section 3 "Available CIP Classes in the Hilscher EtherNet/IP Stack".

For a list of applicable service codes, see *Table 10: Service Codes according to the CIP specification* on page 24.

The class and the instance of the object to be accessed are selected by the variables ulClass and ulInstance of the request packet. In case the requested service will affect an attribute (e.g. services Get_Attribute_Single and Set_Attribute_Single), this attribute is selected by variable ulAttribute of the request packet. Set ulAttribute to 0 when selection of an attribute is not necessary.

If data need to be sent along with the service, this can be achieved by using the array abData[]. The length of data in abData[] must then be added to the ulLen field of the packet header.

The result of the service is delivered in the fields ulGRC (Generic Error Code) and ulERC (Additional Error Code) of the confirmation packet (see Table 128).

If there is data received along with the confirmation this can be found in the array abData[].The ulLen field of the packet header then shows how many bytes are valid within the array.

In case of successful execution, the variables ulGRC and ulERC of the confirmation packet will have the value 0. Usually, in case of an error only the Generic Error Code of the confirmation packet is unequal to 0. Table 128 shows possible GRC values and their meaning.

ulGRC

ulGRC	Signification
0	No error
2	Resources unavailable
8	Service not available
9	Invalid attribute value
11	Already in request mode
12	Object state conflict
14	Attribute not settable
15	A permission check failed
16	State conflict, device state prohibits the command execution
19	Not enough data received
20	Attribute not supported
21	Too much data received
22	Object does not exist
23	Reply data too large, internal buffer to small

Table 128: Generic Error (Variable ulGRC)

Figure 51 and Figure 52 below display a sequence diagram for the EIP_OBJECT_CIP_SERVICE_REQ/CNF packet: in case the host application uses the Basic, Extended or Stack Packet Set (see 4.3 “Configuration Using the Packet API”).

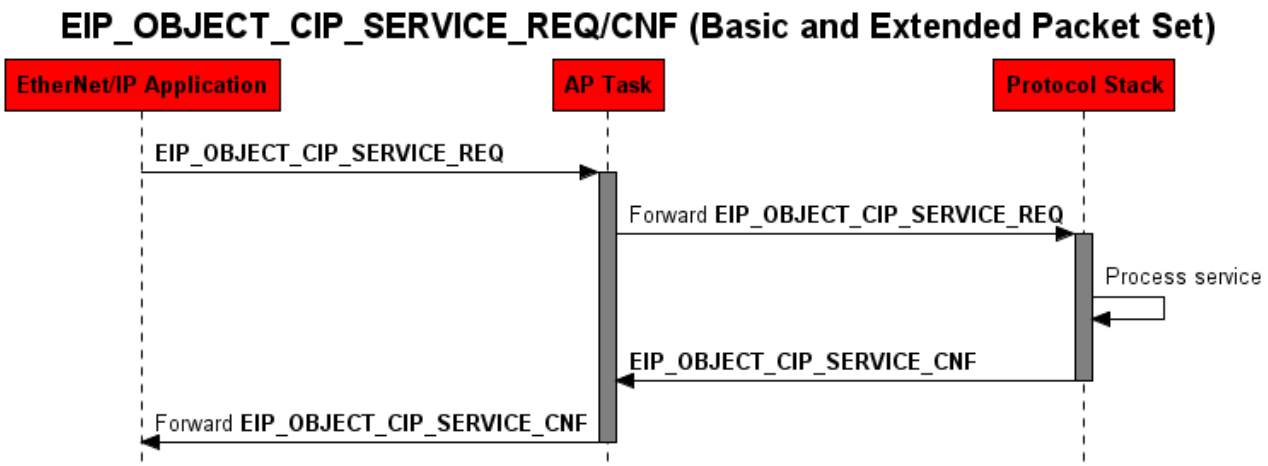

Figure 51: Sequence Diagram for the `EIP_OBJECT_CIP_SERVICE_REQ/CNF` Packet for the Basic and Extended Packet Set

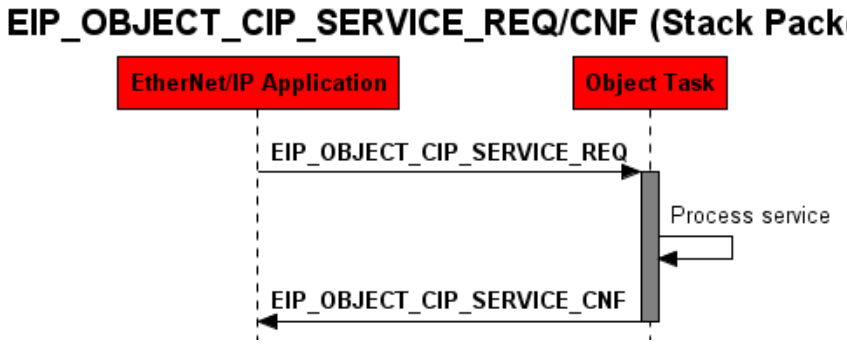

Figure 52: Sequence Diagram for the `EIP_OBJECT_CIP_SERVICE_REQ/CNF` Packet for the Stack Packet Set

Packet Structure Reference

```
#define EIP OBJECT CIP SERVICE MAX PACKET LEN   1520            /*!< Maximum packet length */

typedef struct EIP_OBJECT_CIP_SERVICE_REQ_Ttag
{
  TLR UINT32    ulService;                          /*!< CIP service code          */
  TLR UINT32    ulClass;                            /*!< CIP class ID              */
  TLR UINT32    ulInstance;                         /*!< CIP instance number       */
  TLR_UINT32    ulAttribute;                        /*!< CIP attribute number      */
  TLR_UINT8     abData[EIP_OBJECT_CIP_SERVICE_MAX_PACKET_LEN];  /*!< CIP Service Data. <br><br>
} EIP_OBJECT_CIP_SERVICE_REQ_T;

typedef struct EIP OBJECT PACKET CIP SERVICE REQ Ttag
{
  TLR_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_CIP_SERVICE_MAX_PACKET_LEN)
```

Packet Description

Structure EIP_OBJECT_PACKET_CIP_SERVICE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ OBJECT_QUE	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY()}: when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	16+n	Packet Data Length in bytes n = Length of service data in bytes (see field abData[])
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1AF8	EIP_OBJECT_CIP_SERVICE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_OBJECT_CIP_SERVICE_REQ_T			
ulService	UINT32	1-31	CIP Service Code (see *Table 10: Service Codes according to the CIP specification*)
ulClass	UINT32	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 3 "*Available CIP Classes in the Hilscher EtherNet/IP Stack*" on page 42.
ulInstance	UINT32	Valid Instance number	CIP Object Instance number. For available object classes and instances see section 3 "*Available CIP Classes in the Hilscher EtherNet/IP Stack*" on page 42.
ulAttribute	UINT32	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 3 "*Available CIP Classes in the Hilscher EtherNet/IP Stack*" on page 42.
abData[1520]	UINT8[ ]		CIP Service data Number of bytes n provided in this field must be added to the packet header length field ulLen. Do the following to set the proper packet length: ptReq->tHead.ulLen = EIP_OBJECT_CIP_SERVICE_REQ_SIZE + n

Table 129: EIP_OBJECT_CIP_SERVICE_REQ – CIP Service Request

Packet Structure Reference

```
#define EIP OBJECT CIP SERVICE MAX PACKET LEN   1520            /*!< Maximum packet length */

typedef struct EIP OBJECT CIP SERVICE CNF Ttag
{
  TLR UINT32    ulService;                          /*!< CIP service code          */
  TLR UINT32    ulClass;                            /*!< CIP class ID              */
  TLR UINT32    ulInstance;                         /*!< CIP instance number       */
  TLR_UINT32    ulAttribute;                        /*!< CIP attribute number      */

  TLR UINT32    ulGRC;                              /*!< Generic Error Code        */
  TLR UINT32    ulERC;                              /*!< Extended Error Code       */

  TLR UINT8     abData[EIP OBJECT CIP SERVICE MAX PACKET LEN];  /*!< CIP service data. <br><br>
} EIP_OBJECT_CIP_SERVICE_CNF_T;


typedef struct EIP OBJECT PACKET CIP SERVICE CNF Ttag
{
  TLR_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_CIP_SERVICE_MAX_PACKET_LEN
```

Packet Description

Structure EIP_OBJECT_PACKET_CIP_SERVICE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	See rules in section 3.2.1	Destination Queue Handle
ulSrc	UINT32	See rules in section 3.2.1	Source Queue Handle
ulDestId	UINT32	0	Destination End Point Identifier
ulSrcId	UINT32	x	Source End Point Identifier
ulLen	UINT32	24+n	Packet Data Length in bytes n = Length of service data in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		See chapter *Status/Error Codes Overview*
ulCmd	UINT32	0x1AF9	EIP_OBJECT_CIP_SERVICE_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_OBJECT_CIP_SERVICE_CNF_T			
ulService	UINT32	1-31	CIP Service Code (see *Table 10: Service Codes according to the CIP specification*)
ulClass	UINT32	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	Valid Instance number	CIP Instance number
ulAttribute	UINT32	Valid Attribute number	CIP Attribute number (for get/set attribute only)
ulGRC	UINT32		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 128)
ulERC	UINT32		Additional error code.
abData[1520]	UINT8[ ]		CIP Service data Number of bytes provided in this field must be calculated using the packet header length field `ulLen`. Do the following to get the data size: number of bytes provided in abData = `tHead.ulLen - EIP_OBJECT_CIP_SERVICE_REQ_SIZE`

Table 130: `EIP_OBJECT_CIP_SERVICE_CNF` – Confirmation to CIP Service Request

6.2.18 EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES – CIP Object Change Indication

This indication will be received by the host application when a CIP object attribute is changed/set by service from the network or internally. Basically, changes to object attributes that are non-volatile are indicated. Where meaningful, the response to the change/set service will not be sent by the Protocol Stack until the host application has responded to the change indication.

The new attribute value will be indicated in the service data field of the indication. Whether this value has already been set as the new attribute value or still is about to be set after the host replies to the indication depends on the semantics of the attribute.

Object change indications are subject to a timeout value: If the host does not reply within an interval of 10 seconds after the indication was generated by the Protocol Stack, the causal service request will be replied to with an error status “embedded service error”.

Figure 53 and *Figure 54* below display a sequence diagram for the `EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES` packet in case the host application uses the Basic, Extended or Stack Packet Set (see 4.3 “*Configuration Using the Packet API*”).

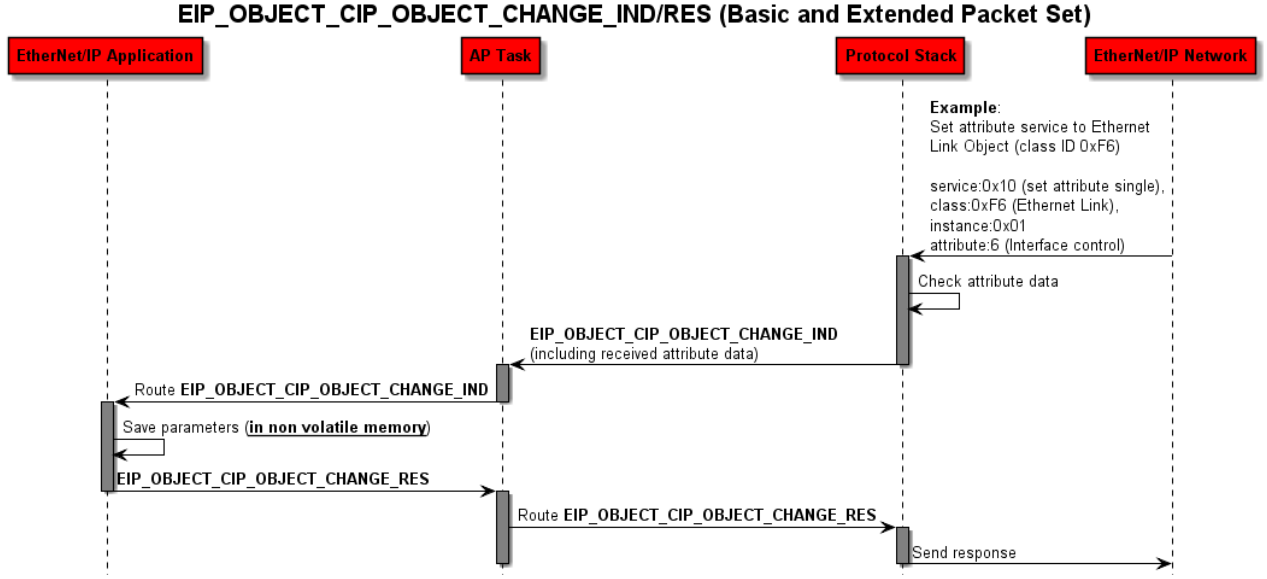

Figure 53: Sequence Diagram for the `EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES` Packet for the Basic and Extended Packet Set

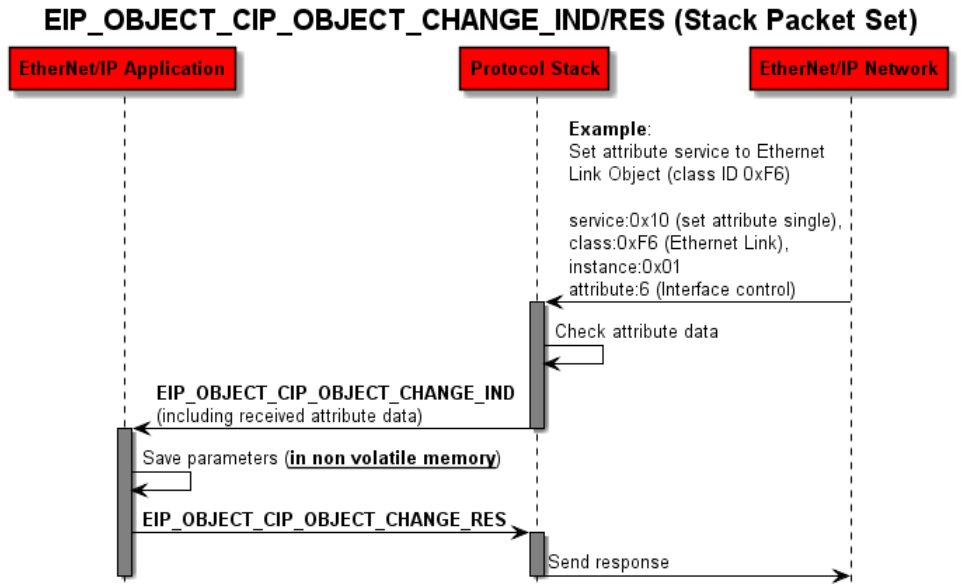

Figure 54: Sequence Diagram for the `EIP_OBJECT_CIP_OBJECT_CHANGE_IND/RES` Packet for the Stack Packet Set

Packet Structure Reference

```
#define EIP OBJECT MAX PACKET LEN   1400

typedef struct EIP_OBJECT_CIP_OBJECT_CHANGE_IND_Ttag
{
  TLR UINT32    ulInfoFlags;                        /*!< Information flags     */
  TLR UINT32    ulService;                          /*!< CIP service code      */
  TLR UINT32    ulClass;                            /*!< CIP class ID          */
  TLR UINT32    ulInstance;                         /*!< CIP instance number   */
  TLR_UINT32    ulAttribute;                        /*!< CIP attribute number  */
  TLR_UINT8     abData[EIP_OBJECT_MAX_PACKET_LEN];  /*!< Service Data          */
} EIP OBJECT CIP OBJECT CHANGE IND T;

typedef struct EIP OBJECT PACKET CIP OBJECT CHANGE IND Ttag
{
  TLR_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

structure EIP_OBJECT_PACKET_CIP_OBJECT_CHANGE_IND_T				
Type: Indication				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure `TLR_PACKET_HEADER_T`			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	20+n	Packet Data Length in bytes n = Number of bytes in abData[]
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See chapter *Status/Error Codes Overview*
	ulCmd	UINT32	0x1AFA	EIP_OBJECT_CIP_OBJECT_CHANGE_IND - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
tData	structure `EIP_OBJECT_CIP_OBJECT_CHANGE_IND_T`			
	ulInfoFlags	UINT32	0 … 3 (Bit mask)	Information flags See Table 132
	ulService	UINT32	0x10	CIP service code (see *Table 10: Service Codes according to the CIP specification*) Currently only the *SetAttributeSingle* service is used in this indication.
	ulClass	UINT32		CIP class ID
	ulInstance	UINT32		CIP instance number
	ulAttribute	UINT32		CIP attribute number
	abData[]	UINT8		Attribute Data Number of bytes n provided in abData = `tHead.ulLen - EIP_OBJECT_CIP_OBJECT_CHANGE_IND_SIZE`

Table 131: `EIP_OBJECT_CIP_OBJECT_CHANGE_IND` – CIP Object Change Indication

Information Flags – ulInfoFlags

Bit	Description
0	**`EIP_CIP_OBJECT_CHANGE_FLAG_STORE_REMANENT`** Signals that the attached attribute data must be stored in non-volatile memory.
1	**`EIP_CIP_OBJECT_CHANGE_FLAG_INTERNAL`** Signals that the object change was done internally. So no service from the network has triggered the change indication. E.g.: This flag is used when the IP configuration has been applied the first time on startup.

Table 132: Information Flags – ulInfoFlags

Packet Structure Reference

```
typedef struct EIP OBJECT PACKET CIP OBJECT CHANGE RES Ttag
{
  TLR PACKET HEADER T             tHead;
} EIP OBJECT PACKET CIP OBJECT CHANGE RES T;

#define EIP_OBJECT_CIP_OBJECT_CHANGE_RES_SIZE    0
```

Packet Description

structure EIP_OBJECT_PACKET_CIP_OBJECT_CHANGE_RES_T				
Type: Response				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	structure `TLR_PACKET_HEADER_T`			
	ulDest	UINT32		Destination Queue-Handle
	ulSrc	UINT32		Source Queue-Handle
	ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
	ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
	ulLen	UINT32	0	Packet Data Length in bytes
	ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
	ulSta	UINT32		See chapter *Status/Error Codes Overview*
	ulCmd	UINT32	0x1AFB	EIP_OBJECT_CIP_OBJECT_CHANGE_RES - Command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch

Table 133: `EIP_OBJECT_CIP_OBJECT_CHANGE_RES` – Response to CIP Object Change Indication

6.2.19 EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ/CNF – CIP Object Attribute Activate Request

This packet can be sent by the host application in order to activate an optional CIP object attribute within the EtherNet/IP stack.

The following Table 134 holds a list of all optional CIP Object attributes that can be activated within the Hilscher EtherNet/IP Stack.

For more information regarding these attributes please have a look at the object description in section *Available CIP Classes in the Hilscher EtherNet/IP Stack* on page 42.

Class		Instance	Attribute	
ID	**Name**	**ID**	**ID**	**Name**
0xF5	TCP/IP Interface Object (Description in section *TCP/IP Interface Object (Class Code: 0xF5)* on page 49)	1	7	SNN (Safety Network Number) **Note:** Activation of the SSN implicitly deactivates the support of the identity object's reset service. All reset services that address the Identity Object will then be rejected with general status code 0x08 (Service not supported).
			8	TTL Value
			9	Mcast Config
			12	EtherNet/IP Quick Connect

Table 134: Overview of optional CIP objects attributes that can be activated

Figure 55 and Figure 56 below display a sequence diagram for the `EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ/CNF` packet in case the host application uses the Extended or Stack Packet Set (see section Configuration Using the Packet API on page 77).

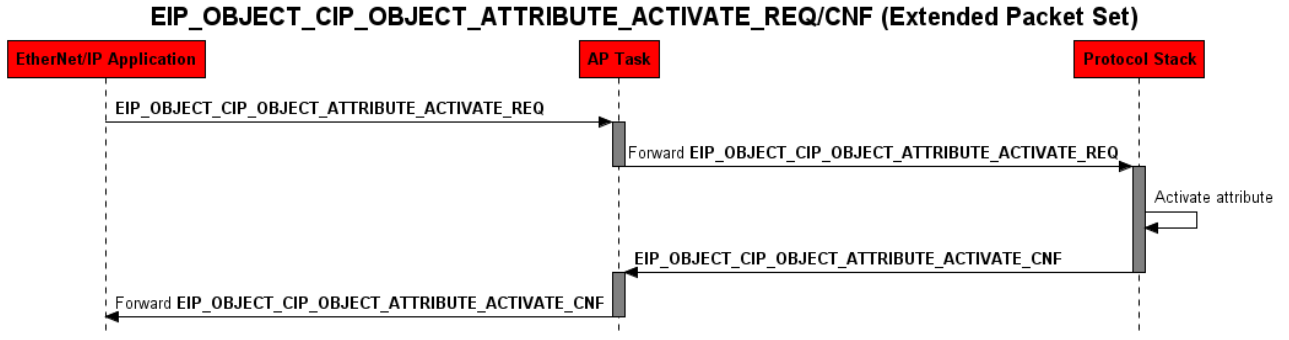

Figure 55: Sequence Diagram for the `EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ/CNF` Packet for the Extended Packet Set

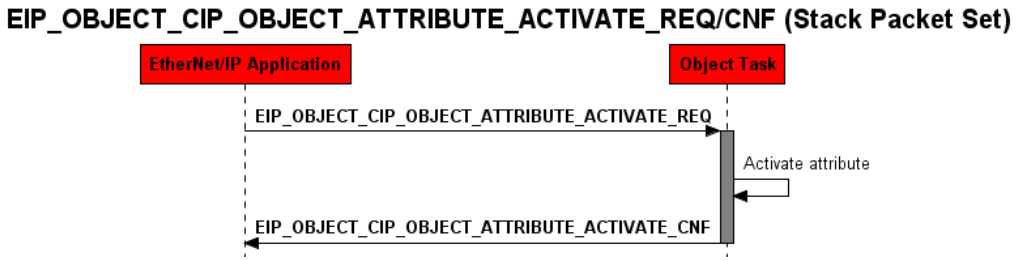

Figure 56: Sequence Diagram for the `EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ/CNF` Packet for the Stack Packet Set

Packet Structure Reference

```
typedef struct EIP OBJECT CIP OBJECT ATTRIBUTE ACTIVATE REQ Ttag
{
  TLR_UINT32    ulEnable;                          /*!< Specifies activation/deactivation
                                                        0: deactivates attribute
                                                        1: activates attribute    */
  TLR UINT32    ulClass;                           /*!< CIP class ID              */
  TLR UINT32    ulInstance;                        /*!< CIP instance number       */
  TLR_UINT32    ulAttribute;                       /*!< CIP attribute number      */

}EIP OBJECT CIP OBJECT ATTRIBUTE ACTIVATE REQ T;

typedef struct EIP OBJECT PACKET CIP OBJECT ATTRIBUTE ACTIVATE REQ Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  EIP OBJECT CIP OBJECT ATTRIBUTE ACTIVATE REQ T   tData;
} EIP OBJECT PACKET CIP OBJECT ATTRIBUTE ACTIVATE REQ T;

#define EIP OBJECT CIP OBJECT ATTRIBUTE ACTIVATE REQ SIZE
sizeof(EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ_T)
```

Packet Description

Structure EIP_OBJECT_PACKET_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of application task process queue
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier
ulSrcId	UINT32	0 ... 2^{32} -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	16	Packet data length in bytes. Depends on number of parameters
ulId	UINT32	0 ... 2^{32} -1	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32		Status not used for request.
ulCmd	UINT32	0x1AFC	EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ – Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ_T			
ulEnable	UINT32	0,1	Specifies activation/deactivation 0: deactivates attribute 1: activates attribute
ulClass	UINT32	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 possible values see Table 134.
ulInstance	UINT32	Valid Instance number	CIP Instance number For possible values see Table 134.
ulAttribute	UINT32	Valid Attribute number	CIP Attribute number (of attribute to be activated/deactivated) For possible values see Table 134.

Table 135: EIP_OBJECT_CIP_OBJECT_ATTRIBUT E_ACTIVATE_REQ – Activate/ Deactivate Slave Request

Packet Structure Reference

```
typedef struct EIP OBJECT PACKET CIP OBJECT ATTRIBUTE ACTIVATE CNF Ttag
{
  TLR PACKET HEADER T  tHead;
} EIP_OBJECT_PACKET_CIP_OBJECT_ATTRIBUTE_ACTIVATE_CNF_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_CIP_OBJECT_ATTRIBUTE_ACTIVATE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of application task process queue
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier
ulSrcId	UINT32	0 ... 2^{32} -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	0	Packet data length in bytes. Depends on number of parameters
ulId	UINT32	0 ... 2^{32} -1	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32		Status not used for request.
ulCmd	UINT32	0x1AFD	EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_CNF – Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 136: EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_CNF – Confirmation to Activate/ Deactivate Slave Request

6.2.20 RCX_LINK_STATUS_CHANGE_IND/RES – Link Status Change

This indication informs the application about the current Link status. This is informative for the application. Information from any earlier received Link Status Changed Indication is invalid at this point of time.

Note: This indication is also sent directly after the host application has registered at the EtherNet/IP Stack (RCX_REGISTER_APP_REQ – 0x2F10).

Packet Structure Reference

```
typedef struct RCX_LINK_STATUS_Ttag
{
  TLR_UINT32  ulPort;           /*!< Port number\n\n
                                     \valueRange \n
                                     0: Port 1 \n
                                     1: Port 2 */

  TLR BOOLEAN fIsFullDuplex;    /*!< Duplex mode\n\n
                                     \valueRange \n
                                     0: Half duplex \n
                                     1: Full Duplex */

  TLR_BOOLEAN fIsLinkUp;        /*!< Link status\n\n
                                     \valueRange \n
                                     0: Link is down \n
                                     1: Link is up */

  TLR_UINT32  ulSpeed;          /*!< Port speed\n\n
                                     \valueRange \n
                                     0:   (No link) \n
                                     10:  10MBit    \n
                                     100: 100MBit   \n */
} RCX LINK STATUS T;


typedef struct RCX LINK STATUS CHANGE IND DATA Ttag
{
  RCX LINK STATUS T  atLinkData[2]; /*!< Link status data */

} RCX_LINK_STATUS_CHANGE_IND_DATA_T;

typedef struct RCX LINK STATUS CHANGE IND Ttag
{
  TLR PACKET HEADER T               tHead;
  RCX_LINK_STATUS_CHANGE_IND_DATA_T tData;
} RCX_LINK_STATUS_CHANGE_IND_T;

#define RCX_LINK_STATUS_CHANGE_IND_SIZE (sizeof(RCX_LINK_STATUS_CHANGE_IND_DATA_T))
```

Packet Description

Structure RCX_LINK_STATUS_CHANGE_IND_T				Type: Indication
Area	**Variable**	**Type**	**Value / Range**	**Description**
Head	structure TLR_PACKET_HEADER_T			
	ulDest	UINT32		Destination queue handle of application task process queue
	ulSrc	UINT32		Source queue handle of AP-task process queue
	ulDestId	UINT32	0	Destination End Point Identifier not in use, set to zero for compatibility reasons
	ulSrcId	UINT32	0 ... 2^{32} -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
	ulLen	UINT32	32	Packet data length in bytes

Structure RCX_LINK_STATUS_CHANGE_IND_T				Type: Indication
Area	**Variable**	**Type**	**Value / Range**	**Description**
	ulId	UINT32	0 ... 2^{32} -1	Packet identification as unique number generated by the source process of the packet
	ulSta	UINT32	0	Status not in use for indication.
	ulCmd	UINT32	0x2FA8	RCX_LINK_STATUS_CHANGE_IND-command
	ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
	ulRout	UINT32	x	Routing, do not touch
Data	structure RCX_LINK_STATUS_CHANGE_IND_DATA_T			
	atLinkData[2]	RCX_LINK_STATUS_T		Link status information for two ports. If only one port is available, ignore second entry.

Table 137: RCX_LINK_STATUS_CHANGE_IND_T - Link Status Change Indication

structure RCX_LINK_STATUS_T				
Area	**Variable**	**Type**	**Value / Range**	**Description**
	ulPort	UINT32	0, 1	The port-number this information belongs to.
	fIsFullDuplex	BOOL32	FALSE (0) TRUE	Is the established link full Duplex? Only valid if fIsLinkUp is TRUE.
	fIsLinkUp	BOOL32	FALSE (0) TRUE	Is the link up for this port?
	ulSpeed	UINT32	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 138: Structure RCX_LINK_STATUS_CHANGE_IND_DATA_T

Packet Structure Reference

```
typedef struct RCX LINK STATUS CHANGE RES Ttag
{
  TLR_PACKET_HEADER_T             tHead;
} RCX_LINK_STATUS_CHANGE_RES_T;

#define RCX_LINK_STATUS_CHANGE_RES_SIZE   (0)
```

Packet Description

Structure RCX_LINK_STATUS_CHANGE_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle of application task process queue
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier
ulSrcId	UINT32	0 ... 2^{32} -1	Source End Point Identifier, specifying the origin of the packet inside the Source Process.
ulLen	UINT32	0	Packet data length in bytes. Depends on number of parameters
ulId	UINT32	0 ... 2^{32} -1	Packet identification as unique number generated by the source process of the packet
ulSta	UINT32		Status not used for request.
ulCmd	UINT32	0x2FA9	RCX_LINK_STATUS_CHANGE_RES – Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 139: RCX_LINK_STATUS_CHANGE_RES_T - Link Status Change Response

6.2.21 EIP_OBJECT_FWD_OPEN_FWD_IND/RES – Indication of a Forward_Open

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).

This indication will be sent to the host application when a Forward_Open request has been received by the protocol stack from the network. The protocol stack forwards the Forward_Open request without performing any processing on it. The host application now has the possibility to check/modify parameters and/or attach “Application Reply“ data (“Application Reply” data will be sent to the originator by attaching it to the Forward_Open response message).

Upon reception of EIP_OBJECT_FWD_OPEN_FWD_RES, the protocol stack processes the Forward_Open request data that comes with this response packet. It will be handled as if it came directly from the network. After 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 or not the connection could be established.

The host application also has the possibility to reject the Forward_Open request right away by setting the corresponding status field in the EIP_OBJECT_FWD_OPEN_FWD_RES packet.

Please have a look at Figure 57 on page 213 to get an overview about the possible packet sequences.

To attach “Application Reply” data, just add it at the end of the connection path (abConnPath) within the Forward_Open data and set the size and offset (ulAppReplyOffset, ulAppReplySize) correspondingly.

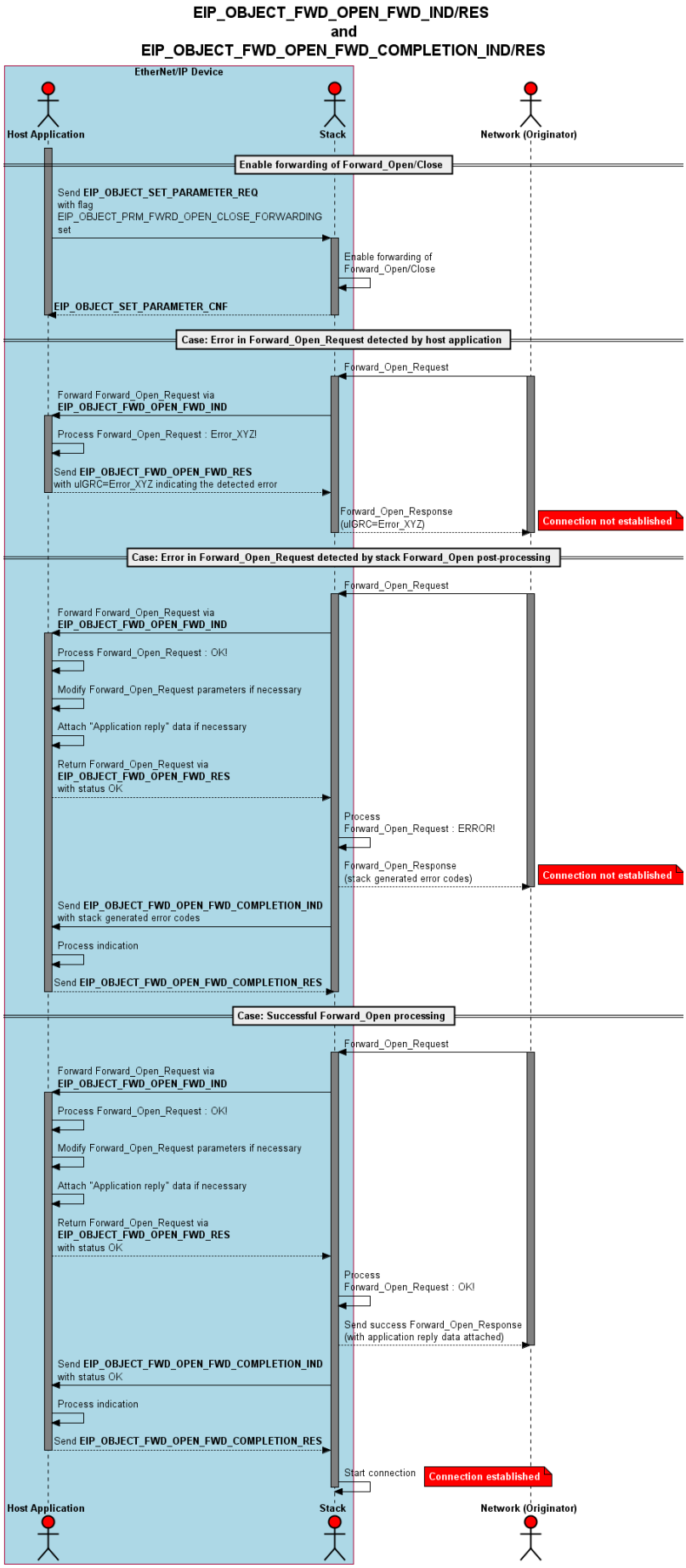

Figure 57: Packet sequence for Forward_Open forwarding functionality

Packet Structure Reference

```
#define EIP OBJECT MAX PACKET LEN   1400

typedef struct EIP CM APP FWOPEN IND Ttag
{
  TLR UINT8   bPriority;
  TLR UINT8   bTimeOutTicks;
  TLR UINT32  ulOTConnID;
  TLR_UINT16  usConnSerialNum;
  TLR UINT16  usVendorId;
  TLR UINT32  ulOSerialNum;
  TLR UINT8   bTimeoutMultiple;
  TLR UINT8   abReserved1[3];
  TLR UINT32  ulOTRpi;
  TLR_UINT16  usOTConnParam;
  TLR UINT32  ulTORpi;
  TLR UINT16  usTOConnParam;
  TLR UINT8   bTriggerType;
  TLR_UINT8   bConnPathSize;
  TLR_UINT8   abConnPath[EIP_OBJECT_MAX_PACKET_LEN];

} EIP_CM_APP_FWOPEN_IND_T;

typedef struct EIP OBJECT FWD OPEN FWD IND Ttag
{
  TLR_UINT32              ulRouteMsg;
  TLR UINT32              aulReserved[4];
  EIP CM APP FWOPEN IND T  tFwdOpenData;
} EIP OBJECT FWD OPEN FWD IND T;

typedef struct EIP_OBJECT_PACKET_FWD_OPEN_FWD_IND_Ttag
{
  TLR PACKET HEADER T                 tHead;
  EIP OBJECT FWD OPEN FWD IND T       tData;
} EIP OBJECT PACKET FWD OPEN FWD IND T;

#define EIP_OBJECT_FWD_OPEN_FWD_IND_SIZE  sizeof(EIP_OBJECT_FWD_OPEN_FWD_IND_T)\
- EIP_OBJECT_MAX_PACKET_LEN
```

Packet Description

Structure EIP_OBJECT_PACKET_FWD_OPEN_FWD_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue-Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	52 + n	EIP_OBJECT_FWD_OPEN_FWD_IND_SIZE + n - Packet Data Length in bytes n: Length of connection path (abConnPath) in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		Status
ulCmd	UINT32	0x1A4A	EIP_OBJECT_FWD_OPEN_FWD_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_OBJECT_FWD_OPEN_FWD_IND_T			
ulRouteMsg	TLR_UINT32		Link to remember underlying encapsulation request (must not be modified by app)
aulReserved[4]	TLR_UINT32		Place holder to be filled by response parameters, see EIP_OBJECT_FWD_OPEN_FWD_RES_T
tFwdOpenData	EIP_CM_APP_FW OPEN_IND_T		Forward Open data (See *Table 141*)

Table 140: EIP_OBJECT_FWD_OPEN_FWD_IND – Forward_Open indication

Structure EIP_CM_APP_ FWOPEN_IND_T		
		Description
bPriority	TLR_UINT8	Used to calculate request timeout information
bTimeOutTicks	TLR_UINT8	Used to calculate request timeout information
ulOTConnID	TLR_UINT32	Network connection ID originator to target
ulTOConnID	TLR_UINT32	Network connection ID target to originator
usConnSerialNum	TLR_UINT16	Connection serial number
usVendorId	TLR_UINT16	Originator Vendor ID
ulOSerialNum	TLR_UINT32	Originator serial number
bTimeoutMultiple	TLR_UINT8	Connection timeout multiplier
abReserved1[3]	TLR_UINT8	reserved
ulOTRpi	TLR_UINT32	Originator to target requested packet rate in microseconds
usOTConnParam	TLR_UINT16	Originator to target connection parameter
ulTORpi	TLR_UINT32	target to originator requested packet rate in microseconds
usTOConnParam	TLR_UINT16	target to originator connection parameter
bTriggerType	TLR_UINT8	Transport type/trigger
bConnPathSize	TLR_UINT8	Connection path size in 16 bit words
abConnPath[1400]	TLR_UINT8	Connection path

Table 141: EIP_CM_APP_FWOPEN_IND_T - Forward_Open request data

Packet Structure Reference

```
typedef struct EIP OBJECT FWD OPEN FWD RES Ttag
{
  TLR UINT32               ulRouteMsg;
  TLR UINT32               ulGRC;
  TLR UINT32               ulERC;
  TLR UINT32               ulAppReplyOffset;
  TLR UINT32               ulAppReplySize;
  EIP_CM_APP_FWOPEN_IND_T  tFwdOpenData;
} EIP OBJECT FWD OPEN FWD RES T;

typedef struct EIP OBJECT PACKET FWD OPEN FWD RES Ttag
{
  TLR PACKET HEADER T                      tHead;
  EIP_OBJECT_FWD_OPEN_FWD_RES_T            tData;
} EIP OBJECT PACKET FWD OPEN FWD RES T;

#define EIP_OBJECT_FWD_OPEN_FWD_RES_SIZE  sizeof(EIP OBJECT FWD OPEN FWD RES T) - \
                                               EIP_OBJECT_MAX_PACKET_LEN
```

Packet Description

structure EIP_OBJECT_PACKET_FWD_OPEN_FWD_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
tHead - Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue Handle
ulDestId	UINT32		Destination Queue Reference
ulSrcId	UINT32		Source Queue Reference
ulLen	UINT32		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
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		
ulCmd	UINT32	0x1A4B	EIP_OBJECT_FWD_OPEN_FWD_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_OBJECT_FWD_OPEN_FWD_RES_T			
ulRouteMsg	TLR_UINT32		Link to underlying Encapsulation request
ulGRC	TLR_UINT32		General Error Code
ulERC	TLR_UINT32		Extended Error Code
ulAppReplyOffset	TLR_UINT32		Offset of “Application Reply” data
ulAppReplySize	TLR_UINT32		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_APP_FWOPEN_IND_T		Forward Open data (See *Table 141*)

Table 142: EIP_OBJECT_FWD_OPEN_FWD_RES – Response of Forward_Open indication

6.2.22 EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND/RES – Indication of Forward_Open Completion Result

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).

This indication will be sent to the host application during processing of a Forward_Open request by the protocol stack from the network.

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 or not the connection could be established.

Please have a look at Figure 57 on page 213 to get an overview about the possible packet sequences.

Packet Structure Reference

```
typedef struct EIP OBJECT FWD OPEN FWD COMPLETION IND Ttag
{
  TLR UINT16  usCmInstance;
  TLR UINT16  usConnSerialNum;
  TLR_UINT16  usVendorId;
  TLR UINT32  ulOSerialNum;
  TLR UINT32  ulGRC;
  TLR UINT32  ulERC;
} EIP OBJECT FWD OPEN FWD COMPLETION IND T;

typedef struct EIP_OBJECT_PACKET_FWD_OPEN_FWD_COMPLETION_IND_Ttag
{
  TLR 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

Structure EIP_OBJECT_PACKET_FWD_OPEN_FWD_COMPLETION_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue-Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	16	EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_SIZE - Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		Status
ulCmd	UINT32	0x1A4C	EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND_T			
usCmInstance	TLR_UINT16	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	TLR_UINT16		Connection serial number
usVendorId	TLR_UINT16		Originator Vendor ID
ulOSerialNum	TLR_UINT32		Originator serial number
ulGRC	TLR_UINT32		General Error Code
ulERC	TLR_UINT32		Extended Error Code

Table 143: EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_IND – Forward_Open completion indication

Packet Structure Reference

```
typedef struct EIP OBJECT PACKET FWD OPEN FWD COMPLETION RES Ttag
{
  TLR 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

Structure EIP_OBJECT_PACKET_FWD_OPEN_FWD_COMPLETION_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue-Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES _SIZE - Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		Status
ulCmd	UINT32	0x1A4D	EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch

Table 144: EIP_OBJECT_FWD_OPEN_FWD_COMPLETION_RES – Response of Forward_Open completion indication

6.2.23 EIP_OBJECT_FWD_CLOSE_FWD_IND - Indication of a Forward_Close

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).

This indication will be sent to the host application when a Forward_Close request was received by the protocol stack from 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.

Upon reception of EIP_OBJECT_FWD_CLOSE_FWD_RES, the protocol stack processes the Forward_Close request data that comes with this response packet. It will be handled as if it came directly from the network.

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.

Please have a look at Figure 58 to get a better understanding of how these packets are used.

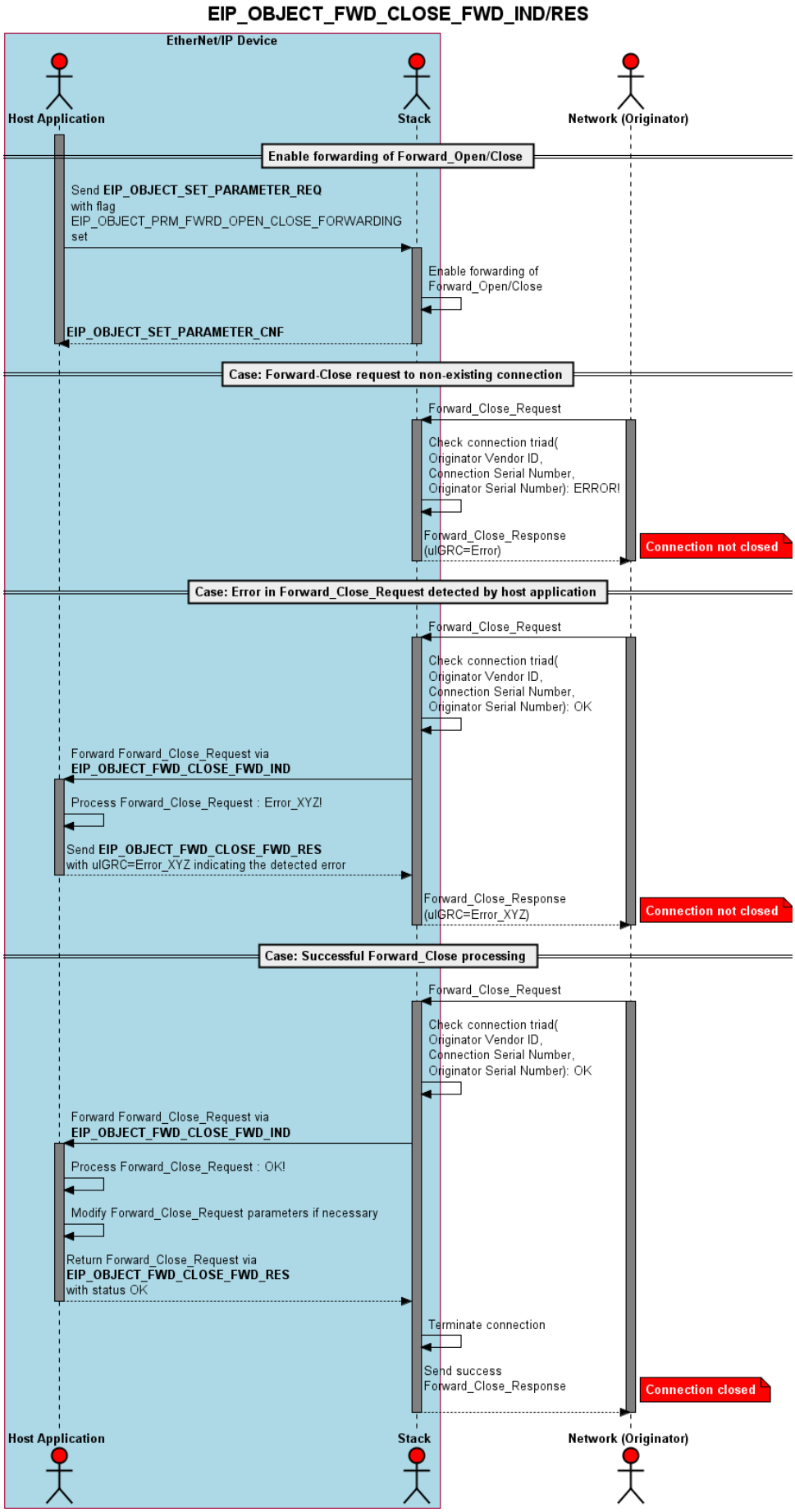

Figure 58: Packet sequence for Forward_Close forwarding functionality

Packet Structure Reference

```
#define EIP OBJECT MAX PACKET LEN   1400

typedef struct EIP CM APP FWCLOSE IND Ttag
{
  TLR UINT8   bPriority;
  TLR UINT8   bTimeOutTicks;
  TLR UINT16  usConnSerialNum;
  TLR_UINT16  usVendorId;
  TLR UINT32  ulOSerialNum;
  TLR UINT8   bConnPathSize;
  TLR UINT8   bReserved1;
  TLR UINT8   bConnPath[EIP OBJECT MAX PACKET LEN];
} EIP CM APP FWCLOSE IND T;

typedef struct EIP OBJECT FWD CLOSE FWD IND Ttag
{
  TLR UINT32               ulRouteMsg;
  TLR_UINT32               aulReserved[2];
  EIP_CM_APP_FWCLOSE_IND_T  tFwdCloseData;
} EIP OBJECT FWD CLOSE FWD IND T;

typedef struct EIP OBJECT PACKET FWD CLOSE FWD IND Ttag
{
  TLR_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) - \
```

Packet Description

Structure EIP_OBJECT_PACKET_FWD_CLOSE_FWD_IND_T			**Type: Indication**
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue-Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32		Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	24 + n	EIP_OBJECT_FWD_CLOSE_FWD_IND_SIZE + n - Packet Data Length in bytes n: Length of connection path (abConnPath) in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		Status
ulCmd	UINT32	0x1A4E	EIP_OBJECT_FWD_CLOSE_FWD_IND - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_OBJECT_FWD_CLOSE_FWD_IND_T			
ulRouteMsg	TLR_UNIT32		Link to remember underlying Encapsulation request (must not be modified by app)
aulReserved[2]	TLR_UINT32		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 146)*

Table 145 : EIP_OBJECT_FWD_CLOSE_FWD_IND – Forward_Close request indication

Structure EIP_CM_APP_FWCLOSE_IND_T		
Variable	**Type**	**Description**
bPriority	TLR_UINT8	Used to calculate request timeout information
bTimeOutTicks	TLR_UINT8	Used to calculate request timeout information
usConnSerialNum	TLR_UINT16	Connection serial number
usVendorId	TLR_UINT16	Originator Vendor ID
ulOSerialNum	TLR_UINT32	Originator serial number
bConnPathSize	TLR_UINT8	Connection path size in 16 bit words
bReserved1	TLR_UINT8	Reserved
abConnPath[1400]	TLR_UINT8	Connection path

Table 146: EIP_CM_APP_FWCLOSE_IND_T - Forward_Close request data

Packet Structure Reference

```
typedef struct EIP OBJECT FWD CLOSE FWD RES Ttag
{
  TLR UINT32               ulRouteMsg;
  TLR UINT32               ulGRC;
  TLR UINT32               ulERC;
  EIP CM APP FWCLOSE IND T tFwdCloseData;
} EIP OBJECT FWD CLOSE FWD RES T;

typedef struct EIP OBJECT PACKET FWD CLOSE FWD RES Ttag
{
  TLR 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

structure EIP_OBJECT_PACKET_FWD_CLOSE_FWD_RES_T			**Type: Response**
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/ DPMINTF_QUE	Destination Queue Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue Handle
ulDestId	UINT32		Destination Queue Reference
ulSrcId	UINT32		Source Queue Reference
ulLen	UINT32	24 + n	EIP_OBJECT_FWD_CLOSE_FWD_RES_SIZE + n - Packet Data Length in bytes n: Length of connection path (abConnPath) in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		
ulCmd	UINT32	0x1A4F	EIP_OBJECT_FWD_CLOSE_FWD_RES - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing, do not touch
tData - Structure EIP_OBJECT_FWD_CLOSE_FWD_RES_T			
ulRouteMsg	TLR_UINT32		Link to underlying Encapsulation request
ulGRC	TLR_UINT32		General Error Code
ulERC	TLR_UINT32		Extended Error Code
tFwdCloseData	EIP_CM_APP_FWCLOSE_IND_T		*Forward Close data (See Table 146: EIP_CM_APP_FWCLOSE_IND_T - Forward_Close request data*)

Table 147: EIP_OBJECT_FWD_CLOSE_FWD_RES – Response of Forward_Close indication

6.2.24 EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ – Create Time Sync Object/Configuration of the Synchronization Mode

In order to activate the Time Sync object within the EtherNet/IP stack, the command EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ needs to be sent to the stack.

Some synchronization-related parameters are required to adjust the interval and offset times for the hardware synchronization signals Sync 0 and Sync 1.

The Sync 0 signal also triggers an interrupt that the host application will receive in order to retrieve the current CIP Sync system time.

In case of a loadable firmware, on each occurrence of the event the EtherNet/IP stack writes the current CIP Sync system time into the extended data area of the Dual Port Memory interface. In case of linkable object module, the host task needs to handle the interrupt by itself.

If the confirmation packet is received with ulSta=0 then the Create Time Sync Object Request has been processed correctly.

Note: Currently, only Sync 0 can be used.

Packet Structure Reference

```
/*****************************************************************************/
/* Request packet definition */
/*****************************************************************************/

typedef struct EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ_Ttag
{
  TLR_UINT32  ulSync0Interval;
  TLR_UINT32  ulSync0Offset;
  TLR_UINT32  ulSync1Interval;
  TLR_UINT32  ulSync1Offset;
  TLR_UINT32  ulPulseLength;
} EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ_T;

#define EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ_SIZE  sizeof(EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ_T)

typedef struct  EIP_OBJECT_PACKET_CREATE_OBJECT_TIMESYNC_REQ_Ttag
{
  TLR_PACKET_HEADER_T                       tHead;
  EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ_T  tData;

}EIP_OBJECT_PACKET_CREATE_OBJECT_TIMESYNC_REQ_T;
```

Packet Description

Structure EIP_OBJECT_PACKET_CREATE_OBJECT_TIMESYNC_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0, 0x20	Destination Queue-Handle. Set to 0: Destination is operating system rcX 32 (0x20): Destination is the protocol stack
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle. Set to: 0: when working with loadable firmware. Queue handle returned by TLR_QUE_IDENTIFY(): when working with loadable firmware.
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0, will not be changed
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process. This variable may be used for low-level addressing purposes.
ulLen	UINT32	20	Packet Data Length (In Bytes);
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Status/Error
ulCmd	UINT32	0x1A58	EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	×	Routing, do not change
tData - Structure EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ_T			
ulSync0Interval	UINT32	0, 5000 … 500000000	Sync0 Interval 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	Sync 0 Offset This parameter specifies a nanosecond offset for the Sync 0 signal relative to the second overrun of the system time (Time of the Sync Master).
ulSync1Interval	UINT32	0, 5000 … 500000000	Sync1 Interval 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	Sync 1 Offset This parameter specifies a nanosecond offset for the Sync 1 signal to the second overrun of the system time (Time of the Sync Master).
ulPulseLength	UINT32	4 … 500	This parameter specifies the pulse length of sync pins 0 and 1 in microseconds. The default value is 4.

Table 148: EIP_OBJECT_CREATE_OBJECT_TIMESYNC_REQ – Create Time Sync Object Request

Packet Structure Reference

```
/*****************************************************************************/
/* Confirmation packet definition */
/*****************************************************************************/

typedef struct  EIP OBJECT PACKET CREATE OBJECT TIMESYNC CNF Ttag
{
  TLR PACKET HEADER T            tHead;
}EIP_OBJECT_PACKET_CREATE_OBJECT_TIMESYNC_CNF_T;

#define EIP_OBJECT_CREATE_OBJECT_TIMESYNC_CNF_SIZE  0
```

Packet Description

Structure EIP_OBJECT_PACKET_CREATE_OBJECT_TIMESYNC_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
tHead – Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0, 0x20	Destination Queue Handle
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue Handle
ulDestId	UINT32	0	Destination End Point Identifier
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier
ulLen	UINT32	16	Packet Data Length (In Bytes);
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	=0 <>0	correct execution an error has occurred
ulCmd	UINT32	0x1A59	EIP_OBJECT_CREATE_OBJECT_TIMESYNC_CNF - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	×	Routing, do not change

Table 149: EIP_OBJECT_CREATE_OBJECT_TIMESYNC_CNF – Confirmation of Create Time Sync Object Request

6.3 The Encapsulation Task

The encapsulation task (EIS_ENCAP task) acts as an encapsulation layer between the high-level CIP protocol and the protocols of the TCP/IP family which are on levels 3 and 4 of the OSI model. It is used for packing CIP messages into TCP, UDP or IP frames according to the EtherNet/IP specification.

The encapsulation task is only used for internal purposes of the EtherNet/IP Adapter protocol stack, you do not require accessing its functionality directly.

6.4 The EIS_CL1-Task

The EIS_CL1-Task does not provide any packet interface.

6.5 The EIS_DLR-Task

The EIS_DLR-Task is only used for internal purposes of the EtherNet/IP Adapter protocol stack, you do not require accessing its functionality directly.

6.6 The TCP_IP-Task

As EtherNet/IP uses protocols of the TCP/IP family as lower level protocols (which are located on levels 3 and 4 of the OSI model for network connections), these protocols need to be handled by a separate task, namely the TCP/IP task. For instance, the TCPIP_IP_CMD_SET_CONFIG_REQ/CNF packet of this task might be of interest in conjunction with EtherNet/IP.

7 Special topics

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

7.1 Getting the Receiver Task Handle of the Process Queue

To get the handle of the process queue of a specific task the Macro `TLR_QUE_IDENTIFY()` needs to be used. It is described in detail within section 10.1.9.3 of the Hilscher Task Layer Reference Model Manual. This macro delivers a pointer to the handle of the intended queue to be accessed (which is returned within the third parameter, `phQue`), if you provide it with the name of the queue (and an instance of your own task). The correct ASCII-queue names for accessing the desired task which you have to use as current value for the first parameter (`pszIdn`) is

ASCII Queue name	Description
"OBJECT_QUE"	Name of the EipObject-Task process queue
"ENCAP_QUE"	Name of the EipEncap-Task process queue
"QUE_EIP_CL1"	Name of the CL1-Task process queue
"QUE_EIP_DLR"	Name of the DLR-Task process queue
"EN_TCPUDP_QUE"	Name of the TCP/IP-Task process queue
"DPMINTF_QUE"	Name of the EIP_APS-Task process queue

Table 150: Names of Queues in EtherNet/IP Firmware

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

8 Status/Error Codes Overview

8.1 Status/Error Codes EipObject-Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC01F0002	TLR_E_EIP_OBJECT_OUT_OF_MEMORY System is out of memory
0xC01F0003	TLR_E_EIP_OBJECT_OUT_OF_PACKETS Task runs out of empty packets at the local packet pool
0xC01F0004	TLR_E_EIP_OBJECT_SEND_PACKET Sending a packet failed
0xC01F0010	TLR_E_EIP_OBJECT_AS_ALLREADY_EXIST Assembly instance already exists
0xC01F0011	TLR_E_EIP_OBJECT_AS_INVALID_INST Invalid Assembly Instance
0xC01F0012	TLR_E_EIP_OBJECT_AS_INVALID_LEN Invalid Assembly length
0xC01F0020	TLR_E_EIP_OBJECT_CONN_OVERRUN No free connection buffer available
0xC01F0021	TLR_E_EIP_OBJECT_INVALID_CLASS Object class is invalid
0xC01F0022	TLR_E_EIP_OBJECT_SEGMENT_FAULT Segment of the path is invalid
0xC01F0023	TLR_E_EIP_OBJECT_CLASS_ALLREADY_EXIST Object Class is already used
0xC01F0024	TLR_E_EIP_OBJECT_CONNECTION_FAIL Connection failed.
0xC01F0025	TLR_E_EIP_OBJECT_CONNECTION_PARAM Unknown format of connection parameter
0xC01F0026	TLR_E_EIP_OBJECT_UNKNOWN_CONNECTION Invalid connection ID
0xC01F0027	TLR_E_EIP_OBJECT_NO_OBJ_RESSOURCE No resource for creating a new class object available
0xC01F0028	TLR_E_EIP_OBJECT_ID_INVALID_PARAMETER Invalid request parameter
0xC01F0029	TLR_E_EIP_OBJECT_CONNECTION_FAILED General connection failure. See also General Error Code and Extended Error Code for more details.
0xC01F0031	TLR_E_EIP_OBJECT_READONLY_INST Access denied. Instance is read only
0xC01F0032	TLR_E_EIP_OBJECT_DPM_USED DPM address is already used by another instance.
0xC01F0033	TLR_E_EIP_OBJECT_SET_OUTPUT_RUNNING Set Output command is already running

Hexadecimal Value	Definition Description
0xC01F0034	TLR_E_EIP_OBJECT_TASK_RESETING EtherNet/IP Object Task is running a reset.
0xC01F0035	TLR_E_EIP_OBJECT_SERVICE_ALREADY_EXIST Object Service already exists
0xC01F0036	TLR_E_EIP_OBJECT_DUPLICATE_SERVICE The service is rejected by the application due to a duplicate sequence count.

Table 151: Status/Error Codes EipObject-Task

8.1.1 Diagnostic Codes

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC01F0001	TLR_DIAG_E_EIP_OBJECT_NO_SERVICE_RES_PACKET No free packet available to create a response of the request.
0xC01F0002	TLR_DIAG_E_EIP_OBJECT_NO_GET_INP_PACKET No free packet available to send the input data.
0xC01F0003	TLR_DIAG_E_EIP_OBJECT_ROUTING_SEND_PACKET_FAIL Routing a request to an object failed. An error occurred at the destination packet queue.
0xC01F0004	TLR_DIAG_E_EIP_OBJECT_ROUTING_SEND_PACKET_CNF_FAIL Sending the confirmation of a request failed. An error occurred at the packet queue.
0xC01F0005	TLR_DIAG_E_EIP_OBJECT_GETTING_UNKNOWN_CLASS_ID Getting a confirmation of a request from an unknown object.
0xC01F0006	TLR_DIAG_E_EIP_OBJECT_NO_CC_INSTANCE_MEMORY Instance of the CC object could not be created. No free memory available.
0xC01F0007	TLR_DIAG_E_EIP_OBJECT_CLOSE_SEND_PACKET_FAIL Completing a connection close command failed. Sending the command to the packet queue failed.
0xC01F0008	TLR_DIAG_E_EIP_OBJECT_OPEN_SEND_PACKET_FAIL Completing a connection open command failed. Sending the command to the packet queue failed.
0xC01F0009	TLR_DIAG_E_EIP_OBJECT_DEL_TRANSP_SEND_PACKET_FAIL Sending the delete transport command failed. Encap Queue signal an error.
0xC01F000A	TLR_DIAG_E_EIP_OBJECT_FW_OPEN_SEND_PACKET_FAIL Sending the forward open command failed. Encap Queue signal an error.
0xC01F000B	TLR_DIAG_E_EIP_OBJECT_START_TRANSP_SEND_PACKET_FAIL Sending the start transport command failed. Encap Queue signal an error.
0xC01F000C	TLR_DIAG_E_EIP_OBJECT_CM_UNKNOWN_CNF Connection manager received a confirmation of unknown service.
0xC01F000D	TLR_DIAG_E_EIP_OBJECT_FW_CLOSE_SEND_PACKET_FAIL Sending the forward close command failed. Encap Queue signal an error.
0xC01F000E	TLR_DIAG_E_EIP_OBJECT_NO_RESET_PACKET Fail to complete reset command. We did not get an empty packet.

Table 152: Diagnostic Codes EipObject-Task

8.2 Status/Error Codes EipEncap-Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC01E0002	TLR_E_EIP_ENCAP_NOT_INITIALIZED Encapsulation layer is not initialized
0xC01E0003	TLR_E_EIP_ENCAP_OUT_OF_MEMORY System is out of memory
0xC01E0010	TLR_E_EIP_ENCAP_OUT_OF_PACKETS Task runs out of empty packets at the local packet pool
0xC01E0011	TLR_E_EIP_ENCAP_SEND_PACKET Sending a packet failed
0xC01E0012	TLR_E_EIP_ENCAP_SOCKET_OVERRUN No free socket is available
0xC01E0013	TLR_E_EIP_ENCAP_INVALID_SOCKET Socket ID is invalid
0xC01E0014	TLR_E_EIP_ENCAP_CEP_OVERRUN Connection could not be opened. No resource for a new Connection Endpoint available
0xC01E0015	TLR_E_EIP_ENCAP_UCMM_OVERRUN Message could not send. All Unconnected Message Buffers are in use
0xC01E0016	TLR_E_EIP_ENCAP_TRANSP_OVERRUN Connection could not be opened. All transports are in use
0xC01E0017	TLR_E_EIP_ENCAP_UNKNOWN_CONN_TYP Received message includes an unknown connection type
0xC01E0018	TLR_E_EIP_ENCAP_CONN_CLOSED Connection was closed
0xC01E0019	TLR_E_EIP_ENCAP_CONN_RESETED Connection is reset from remote device
0x001E001A	TLR_S_EIP_ENCAP_CONN_UNREGISTER We closed the connection successful. With an unregister command
0xC01E001B	TLR_E_EIP_ENCAP_CONN_STATE Wrong connection state for this service
0xC01E001C	TLR_E_EIP_ENCAP_CONN_INACTIV Encapsulation session was deactivated
0xC01E001D	TLR_E_EIP_ENCAP_INVALID_IPADDR received an invalid IP address
0xC01E001E	TLR_E_EIP_ENCAP_INVALID_TRANSP Invalid transport type
0xC01E001F	TLR_E_EIP_ENCAP_TRANSP_INUSE Transport is in use
0xC01E0020	TLR_E_EIP_ENCAP_TRANSP_CLOSED Transport is closed
0xC01E0021	TLR_E_EIP_ENCAP_INVALID_MSGID The received message has an invalid message ID
0xC01E0022	TLR_E_EIP_ENCAP_INVALID_MSG invalid encapsulation message received

Hexadecimal Value	Definition Description
0xC01E0023	TLR_E_EIP_ENCAP_INVALID_MSGLEN Received message with invalid length
0xC01E0030	TLR_E_EIP_ENCAP_CL3_TIMEOUT Class 3 connection runs into timeout
0xC01E0031	TLR_E_EIP_ENCAP_UCMM_TIMEOUT Unconnected message gets a timeout
0xC01E0032	TLR_E_EIP_ENCAP_CL1_TIMEOUT Timeout of a class 3 connection
0xC01E0033	TLR_W_EIP_ENCAP_TIMEOUT Encapsulation service is finished by timeout
0xC01E0034	TLR_E_EIP_ENCAP_CMDRUNNING Encapsulation service is still running
0xC01E0035	TLR_E_EIP_ENCAP_NO_TIMER No empty timer available
0xC01E0036	TLR_E_EIP_ENCAP_INVALID_DATA_IDX The data index is unknown by the task. Please ensure that it is the same as at the indication.
0xC01E0037	TLR_E_EIP_ENCAP_INVALID_DATA_AREA The parameter of the data area are invalid. Please check length and offset.
0xC01E0039	TLR_E_EIP_ENCAP_TASK_RESETING Ethernet/IP Encapsulation Layer runs a reset.
0xC01E003A	TLR_E_EIP_ENCAP_DUPLICATE_SERVICE The service is rejected by the application due to a duplicate sequence count.

Table 153: Status/Error Codes EipEncap-Task

8.2.1 Diagnostic Codes

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC01E0001	TLR_DIAG_E_EIP_ENCAP_NO_LIDENTITY_PACKET No free packet available to indicate the received List Identity information.
0xC01E0002	TLR_DIAG_E_EIP_ENCAP_NO_ENCAP_CMD_PACKET No free packet available to send a request to the Ethernet interface.
0xC01E0003	TLR_DIAG_E_EIP_ENCAP_NO_REGISTER_PACKET No free packet available to send a register session request to the Ethernet interface.
0xC01E0004	TLR_DIAG_E_EIP_ENCAP_CMD_TCP_SEND_PACKET_FAIL Send packet to the Ethernet interface failed.
0xC01E0005	TLR_DIAG_E_EIP_ENCAP_NO_LSERVICE_PACKET No free packet available to indicate the received List Service information.
0xC01E0006	TLR_DIAG_E_EIP_ENCAP_NO_LINTERFACE_PACKET No free packet available to indicate the received List Interface information.
0xC01E0007	TLR_DIAG_E_EIP_ENCAP_NO_MULTICAST_JOIN_PACKET No free packet available to join the multicast group.

Hexadecimal Value	Definition Description
0xC01E0008	TLR_DIAG_E_EIP_ENCAP_NO_MULTICAST_DROP_PACKET No free packet available to drop the multicast group.
0xC01E0009	TLR_DIAG_E_EIP_ENCAP_CONNECTING_INVALID_PACKET_ID By establishing a new connection an invalid packet ID was received.
0xC01E000A	TLR_DIAG_E_EIP_ENCAP_WAIT_CONN_INVALID_PACKET_ID By waiting for a connection an invalid packet ID was received.
0xC01E000B	TLR_DIAG_E_EIP_ENCAP_CEP_OVERRUN No free connection endpoints are available.
0xC01E000C	TLR_DIAG_E_EIP_ENCAP_CONNECTION_INACTIVE Receive data from an inactive or unknown connection.
0xC01E000D	TLR_DIAG_W_EIP_ENCAP_CONNECTION_CLOSED Connection is closed.
0xC01E000E	TLR_DIAG_W_EIP_ENCAP_CONNECTION_RESET Connection is reset
0xC01E000F	TLR_DIAG_E_EIP_ENCAP_RECEIVED_INVALID_DATA Receive invalid data, Connection is closed.
0xC01E0010	TLR_DIAG_E_EIP_ENCAP_UNKNOWN_CONNECTION_TYP Receive data from an unknown connection type
0xC01E0011	TLR_DIAG_E_EIP_ENCAP_CEP_STATE_ERROR Command is not allowed at the actual connection endpoint state.
0xC01E0012	TLR_DIAG_E_EIP_ENCAP_NO_INDICATION_PACKET No free packet available to send an indication of the received data.
0xC01E0013	TLR_DIAG_E_EIP_ENCAP_REVEIVER_OUT_OF_MEMORY No memory for a receive buffer is available, data could not received.
0xC01E0014	TLR_DIAG_E_EIP_ENCAP_NO_ABORT_IND_PACKET No free packet available to send an abort transport indication.
0xC01E0015	TLR_DIAG_E_EIP_ENCAP_START_CONNECTION_FAIL Starting the connection failed. Connection endpoint is invalid.
0xC01E0016	TLR_DIAG_E_EIP_ENCAP_NO_GET_TCP_CONFIG_PACKET No free packet for requesting the actual configuration from the TCP task

Table 154: Diagnostic Codes EipEncap-Task

8.3 Status/Error Codes EIS_APS-Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0590001	TLR_E_EIP_APS_COMMAND_INVALID Invalid command.
0xC0590002	TLR_E_EIP_APS_PACKET_LENGTH_INVALID Invalid packet length.
0xC0590003	TLR_E_EIP_APS_PACKET_PARAMETER_INVALID Invalid packet parameter.
0xC0590004	TLR_E_EIP_APS_TCP_CONFIG_FAIL TCP/IP configuration failed. The TCP/IP task reports an error: IP address, gateway address, network mask or configuration flags are invalid.
0xC0590007	TLR_E_EIP_APS_ACCESS_FAIL Unregister application command rejected, because another task then the registered task has send an unregister application command. Only the registered task can send the unregister application command.
0xC0590008	TLR_E_EIP_APS_STATE_FAIL In normal state: clear watchdog command received. This command can’t be processed in this state and is rejected. In watchdog error state: the received command can’t be processed in this state and is rejected.
0xC0590009	TLR_E_EIP_APS_IO_OFFSET_INVALID Invalid I/O offset.
0xC059000A	TLR_E_EIP_APS_FOLDER_NOT_FOUND Expected folder contains the configuration file(s) not found.
0xC059000B	TLR_E_EIP_APS_CONFIG_DBM_INVALID The configuration file ‘config.nxd’ does not contain the expected configuration parameters.
0xC059000C	TLR_E_EIP_APS_NO_CONFIG_DBM Configuration file named ‘config.nxd’ not found. As a result, EtherNet/IP configuration parameters are missing.
0xC059000D	TLR_E_EIP_APS_NWID_DBM_INVALID The configuration file named ‘nwid.nxd’ does not contain the expected configuration parameters.
0xC059000E	TLR_E_EIP_APS_NO_NWID_DBM Configuration file ‘nwid.nxd’ not found. As a result, TCP/IP configuration parameters are missing.
0xC059000F	TLR_E_EIP_APS_NO_DBM Configuration file missing.
0xC0590010	TLR_E_EIP_APS_NO_MAC_ADDRESS_AVAILABLE No MAC address available.

Table 155: Error Codes EIS_APS-Task

8.3.1 Diagnostic Codes EIS_APS-Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0590001	TLR_DIAG_E_EIP_APS_TCP_CONFIG_FAIL TCP stack configuration failed.
0xC0590002	TLR_DIAG_E_EIP_APS_CONNECTION_CLOSED Existing connection is closed.

8.4 Status/Error Codes Eip_DLR-Task

Hexadecimal Value	Definition Description
0x00000000	TLR_S_OK Status ok
0xC0950001	TLR_E_EIP_DLR_COMMAND_INVALID Invalid command received.
0xC0950002	TLR_E_EIP_DLR_NOT_INITIALIZED DLR task is not initialized.
0xC0950003	TLR_E_EIP_DLR_FNC_API_INVALID_HANDLE Invalid DLR handle at API function call.
0xC0950004	TLR_E_EIP_DLR_INVALID_ATTRIBUTE Invalid DLR object attribute.
0xC0950005	TLR_E_EIP_DLR_INVALID_PORT Invalid port.
0xC0950006	TLR_E_EIP_DLR_LINK_DOWN Port link is down.
0xC0950007	TLR_E_EIP_DLR_MAX_NUM_OF_TASK_INST_EXCEEDED Maximum number of EthernetIP task instances exceeded.
0xC0950008	TLR_E_EIP_DLR_INVALID_TASK_INST Invalid task instance.
0xC0950009	TLR_E_EIP_DLR_CALLBACK_NOT_REGISTERED Callback function is not registered.
0xC095000A	TLR_E_EIP_DLR_WRONG_DLR_STATE Wrong DLR state.
0xC095000B	TLR_E_EIP_DLR_NOT_CONFIGURED_AS_SUPERVISOR Not configured as supervisor.
0xC095000C	TLR_E_EIP_DLR_INVALID_CONFIG_PARAM Configuration parameter is invalid.
0xC095000D	TLR_E_EIP_DLR_NO_STARTUP_PARAM_AVAIL No startup parameters available.

Table 156: Status/Error Codes `Eip_DLR-Task`

8.5 General EtherNet/IP Error Codes

The following table contains the possible General Error Codes defined within the EtherNet/IP standard.

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 where 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 157: General Error Codes according to CIP Standard

9 Appendix

9.1 Module and Network Status

This section describes the LED signaling of EtherNet/IP Adapter devices. 2 LEDs display status information namely the Module Status LED denominated as MS and the network Status LED denominated as NS.

9.1.1 Module Status

Table 158 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 minor fault.
`EIP_MS_MAJOR_UNRECOVERABLE_FAULT`	5	**Major unrecoverable fault** If the device has detected a major unrecoverable fault, the module status indicator is steady red.

Table 158: Possible values of the Module Status

9.1.2 Network Status

Table 159 lists the possible values of the Network Status and their meanings (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.
`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.
`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.
`EIP_NS_TIMEOUT`	3	**Connection timeout** An Exclusive Owner connection for which this device is the target has timed out. The network status indicator returns to steady green only when all timed out Exclusive Owner connections are reestablished. The Network LED turns from flashing red to steady green only when all connections to the previously timed-out O->T connection points are reestablished. Timeout of connections other than Exclusive Owner connections do not cause the indicator to flash red. The Flashing Red state applies to target connections only.
`EIP_NS_DUPIP`	4	**Duplicate IP** The device has detected that its IP address is already in use.
`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 159: Possible values of the Network Status

There are 3 packets provided by the EIS_AP task dealing with the Module Status and the Network Status:

- `EIP_APS_SET_PARAMETER_REQ/CNF` described in section 6.1.3 on page 107

  This packet allows enabling notifications on changes of Module Status and Network Status.

- `EIP_APS_MS_NS_CHANGE_IND/RES` described in section 6.1.4 on page110

  This packet notifies the host application about changes of the Module Status and the Network Status.

- `EIP_APS_GET_MS_NS_REQ/CNF` described in section 6.1.5 on page113

  This packet allows the application to retrieve the current Module Status and Network Status.

9.2 Quality of Service (QoS)

9.2.1 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 also mandatory for DLR (see section 9.3”DLR”).

In TCP/IP-based protocols, there are two standard mechanisms available for implementing QoS. These are:

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

which are both described in more detail below.

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

9.2.2 DiffServ

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

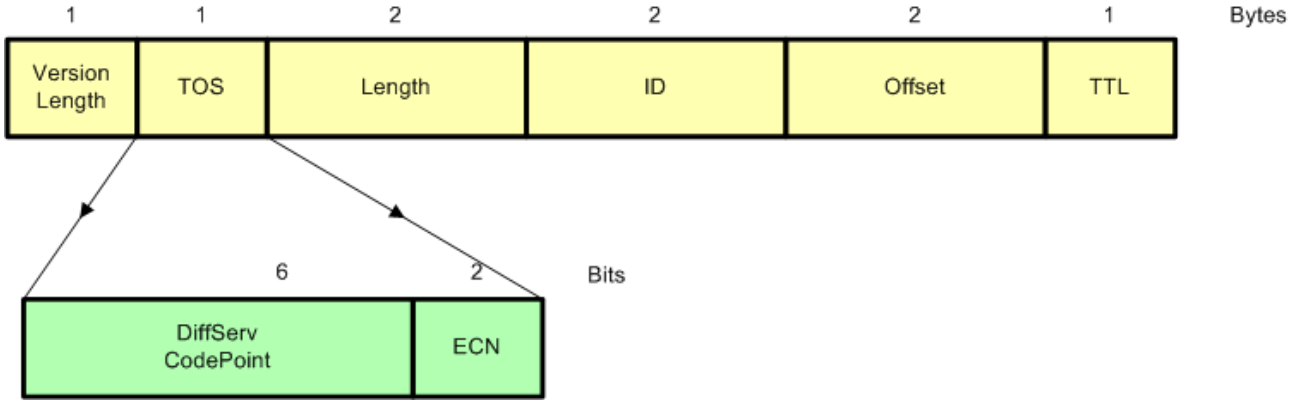

Figure 59: TOS Byte in IP v4 Frame Definition

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 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). A lot of 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 160: Default Assignment of DSCPs in EtherNet/IP

9.2.3 802.1D/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)

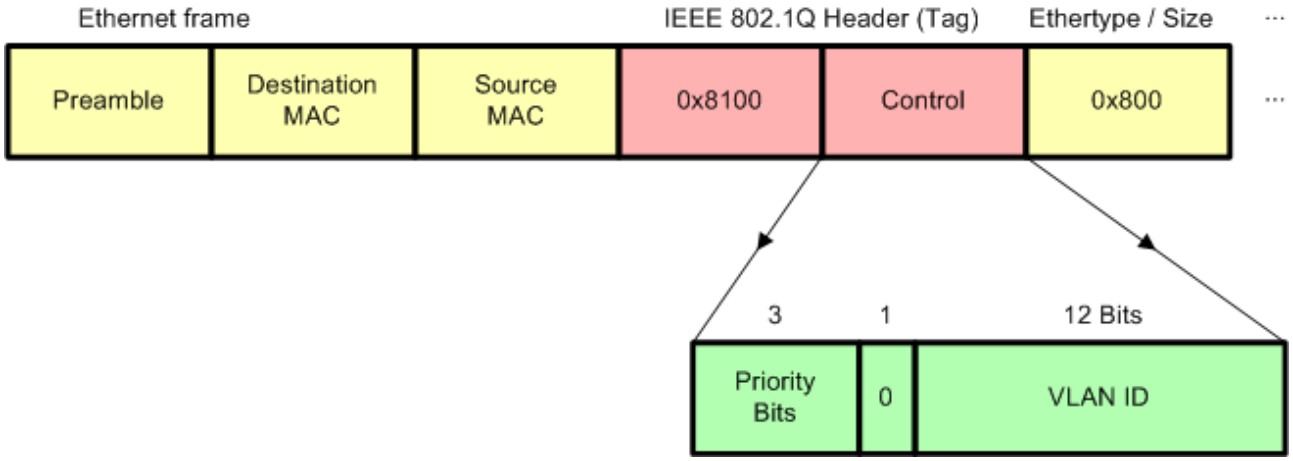

Figure 60: Ethernet Frame with IEEE 802.1Q Header

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 161: Default Assignment of 802.1D/Q Priorities in EtherNet/IP

9.2.4 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 3.10 „*Quality of Service Object (Class Code: 0x48)* “ of this document.

9.2.4.1 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.

9.3 DLR

This section intends to give a brief and compact overview about the basic facts and concepts of the DLR (Device level Ring) networking technology supported by Hilscher's EtherNet/IP Adapter protocol stack.

DLR is a technology (based on a special protocol additionally to Ethernet/IP) 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 capabilities).

In general, there are two kinds of nodes in the network:

- Ring supervisors
- Ring nodes

DLR requires all modules (both supervisors and normal ring nodes) to be equipped with two Ethernet ports and internal switching technology.

Each module within the DLR network checks the target address of the currently received DLR frame whether it matches its own MAC address.

- If yes, it keeps the packet and processes it. It will not be propagated any further.
- If no, it propagates the packet via the other port which did not receive the packet.

There is a ring topology so that all devices in the DLR network are each connected to two different neighbors with their two Ethernet ports. In order to avoid looping, one port of the (active) supervisor is blocked.

9.3.1 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

An active 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.

At least one active ring supervisor is required within a DLR network.

Back-up supervisors

It is recommended but not necessary that each DLR network should have at least one back-up supervisor. If the active supervisor of the network fails, the back-up supervisor will take over the duties of the active supervisor.

9.3.2 Precedence Rule for Multi-Supervisor Operation

Multi-Supervisor Operation is allowed for DLR networks. If more than one supervisor is configured as active on the ring, the following rule applies in order to determine the supervisor which is relevant:

Each supervisor contains an internal precedence number which can be configured. The supervisor within the ring carrying the highest precedence number will be the active supervisor, the others will behave passively and switch back to the state of back-up supervisors.

9.3.3 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, however.

Frequency

- Beacon frames are always sent every beacon interval. Usually, a beacon interval is defined to have an interval of 400 microseconds. However, beacon frames may be sent even faster up to an interval of 100 microseconds.
- Announce frames are always sent in time intervals of one 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 of the ring on its own.

9.3.4 Ring Nodes

This subsection deals with modules in the ring, which does 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 on one port a beacon frame from a supervisor is received, 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 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.

  The FAULT_STATE provides partial ring support, but the ring is still not fully operative in FAULT_STATE. If the beacon frames have a time-out on both ports, the state will change to the IDLE_STATE. If on both ports a beacon frame is received 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 fully 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.

9.3.5 Normal Network Operation

In normal operation, the supervisor sends beacon and, if configured, 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 may send announce frames once per second and additionally, if an error is detected.

9.3.6 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.

9.3.7 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 fully 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 [5], section “9-5 Device Level Ring”.

9.4 Quick Connect

9.4.1 Introduction

In many automotive applications, robots, tool changers and framers are required to quickly exchange tooling fixtures which contain a section or segment of an industrial network. This requires the network and nodes to be capable of quickly connecting and disconnecting, both mechanically, and logically.

While the mechanical means for connecting and disconnecting tooling exists, achieving a quick re-establishment of a logical network connection between a network controller and a fully powered-down node on Ethernet can take as much as 10 or more seconds. This is too slow for applications that require very short cycle times.

The time in which a robot arm first makes electrical contact with a new tool, until the mechanical lock being made, is typically 1 second. In applications where the tools are constantly being connected and disconnected, the nodes need to be able to achieve a logical connection to the controller and test the position of the tool in less than 1 second from the time the tool and the robot make an electrical connection. This means that the node needs to be able to power up and establish a connection in approximately 500 ms.

It should be noted that controller and robotic application behavior is outside the scope of this specification.

The Quick Connect feature is an option enabled on a node-by-node basis. When enabled, the Quick Connect feature will direct the EtherNet/IP target device to quickly power up and join an EtherNet/IP network.

In order for Quick Connect devices to power up as quickly as possible, manufacturers should minimize the hardware delay at power-up and reset as much as possible.

The Quick Connect feature is enabled within the device through the non-volatile EtherNet/IP Quick Connect attribute (12) in the TCP/IP object. A device shall have this feature disabled as the factory default.

The goal for Quick Connect connection time is 500ms. Specifically, this is defined as the guaranteed repeatable time between the electrical contact of power and Ethernet signals at the tool changer, and when the newly connected devices are ready to send the first CIP I/O data packet.

Quick Connect connection time is comprised of several key time durations. The majority of the Quick Connect connection time is due to the Quick Connect target devices' power-up time. Also contributing to the connection time is the amount of time it takes a controller to detect the newly attached device and send a Forward Open to start the connection process. The overall 500ms Quick Connect connection time is additive, and consists of the Quick Connect devices' power-up time, the controller's connection establishment time, and actual network communication time. Also, the network communication time is dependent on the network topology. For instance, in a linear topology, the network communication time will be dependent on all devices powering up, plus the delay through all of the devices. The final application connection time assumes that connections to ALL of the I/O devices on the tool have been established.

The following figure shows the events, states, and sequence in which a controller shall discontinue communications with a device on a given tool and then establish a connection to a device on a new tool. Note: There can be multiple I/O devices on the tool. This sequence is repeated for each connection from the controller to the I/O devices on the tool.

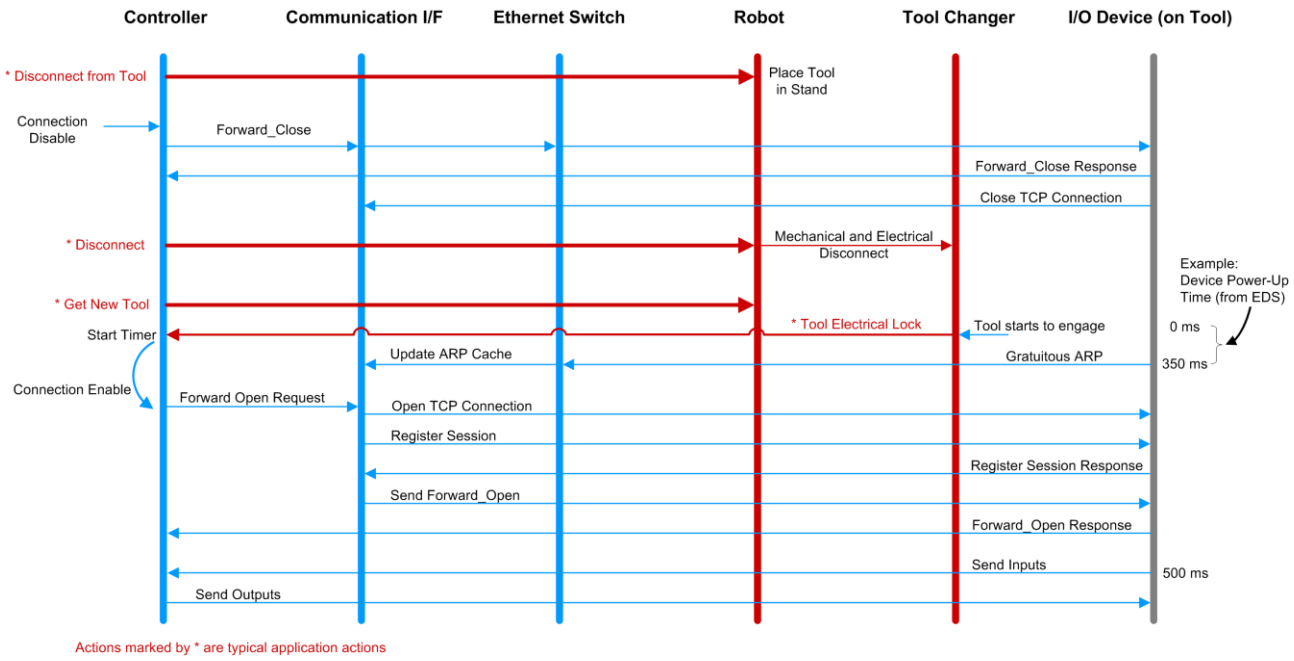

Figure 61: Quick Connect System Sequence Diagram

There are two classes of Quick Connect devices.

- Class A Quick Connect target devices is able to power-up, send the first Gratuitous ARP packet, and be ready to accept a TCP connection in less than 350ms.
- Class B Quick Connect target devices shall be able to power-up, send the first Gratuitous ARP packet, and be ready to accept a TCP connection in less than 2 seconds.

9.4.2 Requirements

EtherNet/IP target devices supporting QuickConnect must adhere to the following requirements:

- In order to be able to establish a physical link as fast as possible all Ethernet ports shall be set to 100 MBit/s and full duplex
- When in Quick Connect mode Quick Connect devices shall not use Auto-MDIX (detection of the required cable connection type)
- To enable the use of straight-thru cables when Auto-MIDX is disabled, the following rules shall be applied:

  A. On a device with only one port: the port shall be configured as MDI.

  B. On devices with 2 external Ethernet ports:

     The labels for the 2 external ports shall include an ordinal indication (e.g.: Port 1 and Port 2, or A and B)

     The port with the lower ordinal indication shall be configured as MDI.

     The port with the upper ordinal indication shall be configured as MDIX.

- The target device shall support EtherNet/IP Quick Connect attribute (12) in the TCP/IP Object that enables the Quick Connect feature.
  This optional attribute 12 can be activated using the command `EIP_OBJECT_CIP_OBJECT_ATTRIBUTE_ACTIVATE_REQ/CNF` – CIP Object Attribute Activate Request)
- The target device shall have the Quick Connect keywords and values included in the device's EDS file.

9.5 Non-Null Forward Open and Null Forward Open

9.5.1 Introduction

The Forward_Open service (Service Code = 0x54) is used to establish a connection with a target device. This service results in local connection establishment on each link along the path.

A Forward Open can be either

- a non-Null Forward Open or
- a Null Forward Open.

Non-Null Forward Open: A non-Null Forward Open is a Forward_Open service request for which at least one of the Connection Types in the O2T or T2O network connection parameter field is not 00 (NULL).

Null Forward Open: A Null Forward Open is a Forward_Open service request for which the Connection Type in both the O2T and T2O network connection parameter fields are both 00 (NULL) and results in no connection being established.

A Forward Open (both Null and non-Null) can be either **not matching** or **matching**. A **matching** Forward_Open service request received by the target device is one where the Connection Triad matches an existing connection. The Connection Triad relates to the combination of “Connection Serial Number”, “Originator Vendor ID” and “Originator Serial Number” parameters, which are all part of the forward open request.

9.5.2 Use cases

The following table lists the use cases of each combination of non-Null/Null and not matching/matching Forward Opens:

Use case	Type and description
1	**Non-Null / not matching → Open a connection** This type is used to open implicit connections (Exclusive Owner, Input Only, Listen Only) and explicit connections. This is what the Hilscher EtherNet/IP protocol stack support by default. There is no need to activate this functionality through the protocol stack's API.
2	**Non-Null / matching → Error** The Hilscher EtherNet/IP protocol stack will reject this type of forward open as it is typically happens when in case the same forward open request ist received a seconds time while the connection has already been established.
3a	**Null / not matching – Ping a device** A Null Forward Open for which the Connection Triad does not match an existing connection's parameters can be used to "ping" a device. The following characteristics apply: ▪ Single application path "20 01 24 01" (Identity Object). ▪ An electronic key segment may be included. ▪ No data segment is included. ▪ No connection is established.
3b	**Null / not matching – Configure** A Null Forward Open for which the Connection Triad does not match an existing connection's parameters can be used to configure an application of the device. The following characteristics apply: ▪ A configuration application path and data segment is included in the request. The data is sent to the application specified by the path and applied. ▪ If the entire configuration cannot be applied by the application then none of the configuration shall be applied and the appropriate error code returned. ▪ An electronic key segment may be included. ▪ No connection is established.
4	**Null / matching – Reconfigure** A Null Forward Open for which the Connection Triad matches an existing parameters of the connection can be used to reconfigure a target application of the device. The following characteristics apply: ▪ A configuration application path and data segment is included in the request and they are sent to the application to change the application configuration. ▪ If the entire configuration cannot be applied by the application then none of the configuration shall be applied and the appropriate error code returned. ▪ The connection is not interrupted due to this request. **Note:** If the interpretation of the consume/produce data changes as a result of the reconfigure operation, care must be taken in the producing and consuming applications. The CIP specification does not provide a mechanism to coordinate between the producing and consuming applications, so a change in the meaning of the real time data can result in unexpected operation. Devices have the option to reject a reconfiguration request with an Object State Conflict error (General Status = 0x0C), to prevent this situation.

Table 162: Use cases of Forward Open

9.5.3 Using the Null Forward Open Feature

Typically, EtheNet/IP devices support only use case 1. In this case, a configuration assembly instance is available to transport device-specific configuration data to the host application. In this case, the configuration data is attached to the Forward Open request and is provided to the host application (also realized via packet EIP_OBJECT_CONNECTION_CONFIG_IND - 0x1A40). This way of configuration does not require the support of the Null Forward Open feature.

In addition to use case 1, the Null Forward Open feature adds the following functionality to the device:

1. Configure the host application of the device without opening a connection.
2. Re-configure the host application of the device while the IO connection is already running. E.g. the originator of the connection (PLC) can change the configuration of the device during run-time

9.5.3.1 Activatation

The Null Forward Open use cases 3a, 3b and 4 (see section *Use cases* on page 257) are not supported by default. In order to support them, the host application must enable the Null Forward Open feature by sending the packet EIP_OBJECT_SET_PARAMETER_REQ with bit 8 set (see section *EIP_OBJECT_SET_PARAMETER_REQ/CNF – Set Parameter* on page 181).

9.5.3.2 Handling of use cases

Use case 3a: Ping

The protocol stack handles use case 3a (“Ping”). There is not need for the host application to do anything.

Use case 1, 3b and 4: Configure and Re-Configure

Whenever the protocol stack receives a Null Forward Open that matches the above mentioned characteristics (3a or 4), it will send the packet EIP_OBJECT_CONNECTION_CONFIG_IND (0x1A40) to the host application (see also section *EIP_OBJECT_CONNECTION_CONFIG_IND/RES – Indication of Configuration Data received during Connection Establishment* on page 171).

Note: The protocol stack sends this indication also for use case 1 when a configuration assembly instances is addressed in the Forward Open request.

When processing the indication packet, the host application now must determine whether it can apply the provided configuration data in its current state. The host application can determine the actual Forward Open use case by taking the following parameters into account:

- tConnectionTriad
- ulOTParameter
- ulTOParameter

```
If ( "Connection Type" inside ulOTParameter or ulTOParameter is != 0 )
{
Use case 1
}
else if ( "Connection Type" inside ulOTParameter and ulTOParameter  is 0 )
{
If (tConnectionTriad.fConnectionTriadMatch == false)
{
Use case 3B
}
else
{
Use case 4
}
}
```

The CIP specification does not define any specific behavior for the host application with respect to use cases 1, 3b and 4. It is entirely up to the device manufacturer what to do in each use case. The easiest way will be to always accept the received configuration data. Of course, this also depends on the configuration data itself.

9.5.3.3 Preparing the EDS file for the Null Forward Open Support

In order to make the Null Forward Open also available with the EDS file, the following adaptions can be made using the EDS file editor "EZ-EDS" (Freeware tool can be downloaded on https://www.odva.org)

For the Null Forward Open "Ping"

Add a new entry to the connection manager section and chose type "Ping a device". EZ-EDS will configure all connection properties for you.

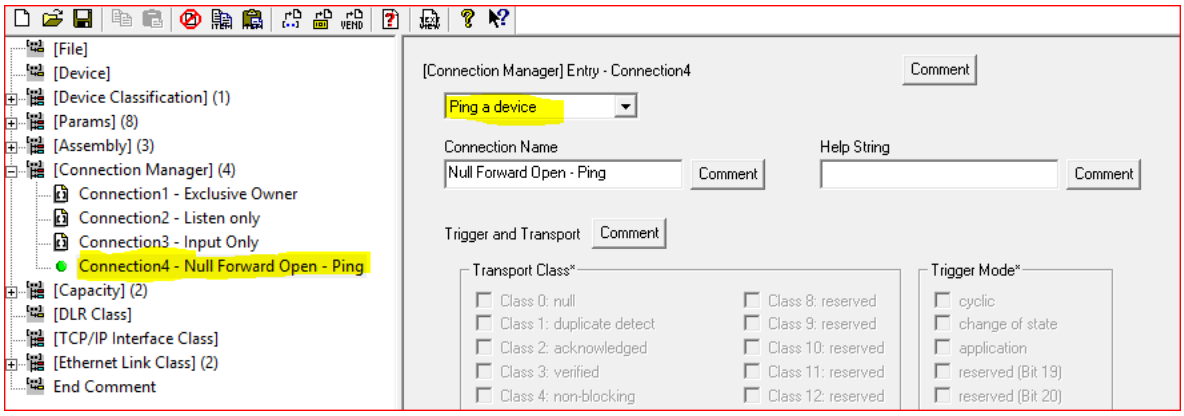

Figure 62: EDS: connection entry for Null Forward Open - "Ping"

For the Null Forward Open “Configure” / “Re-Configure”

Chose the connection entry that addresses a configuration assembly and set the check mark for “allow Reconfiguration”.

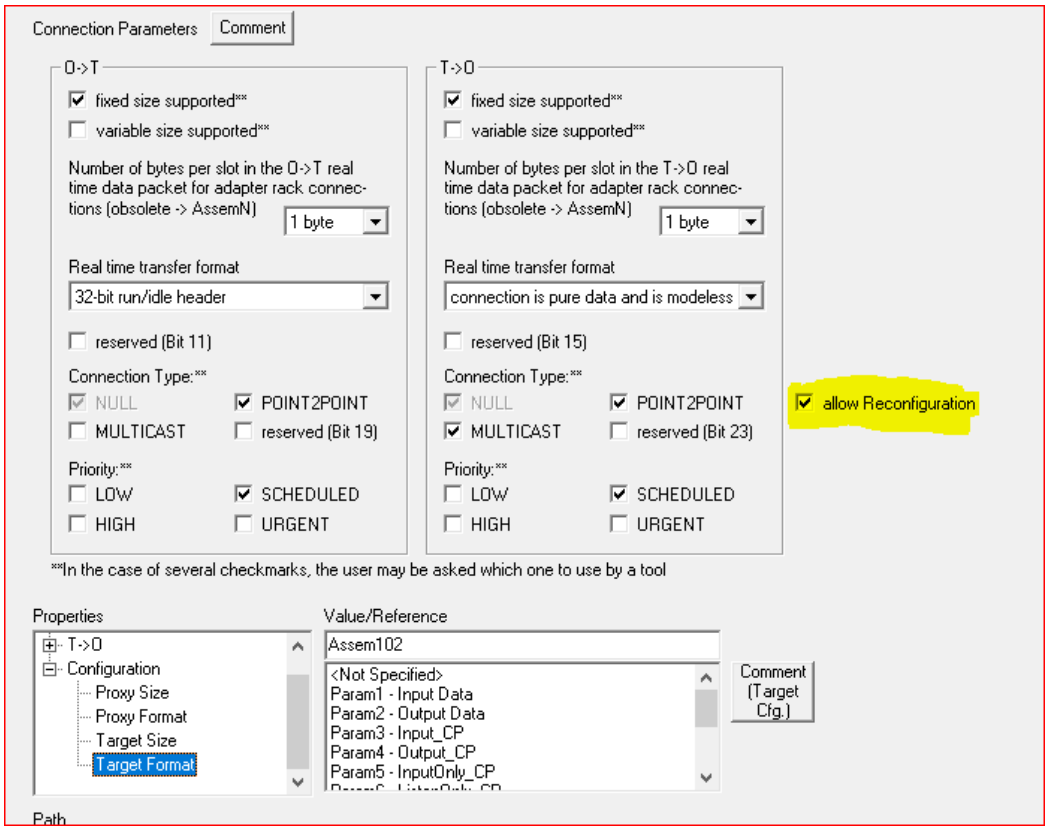

Figure 63: EDS: connection entry for Null Forward Open - "Re-Configuration"

9.5.3.4 Preparing the STC file for the Null Forward Open Support

In order to make the Null Forward Open also available SCT file of the Conformance Test tool, the following adaptions need to be made:

For the Null Forward Open “Ping”

Add a new connection entry to the connection manager with the following properties:

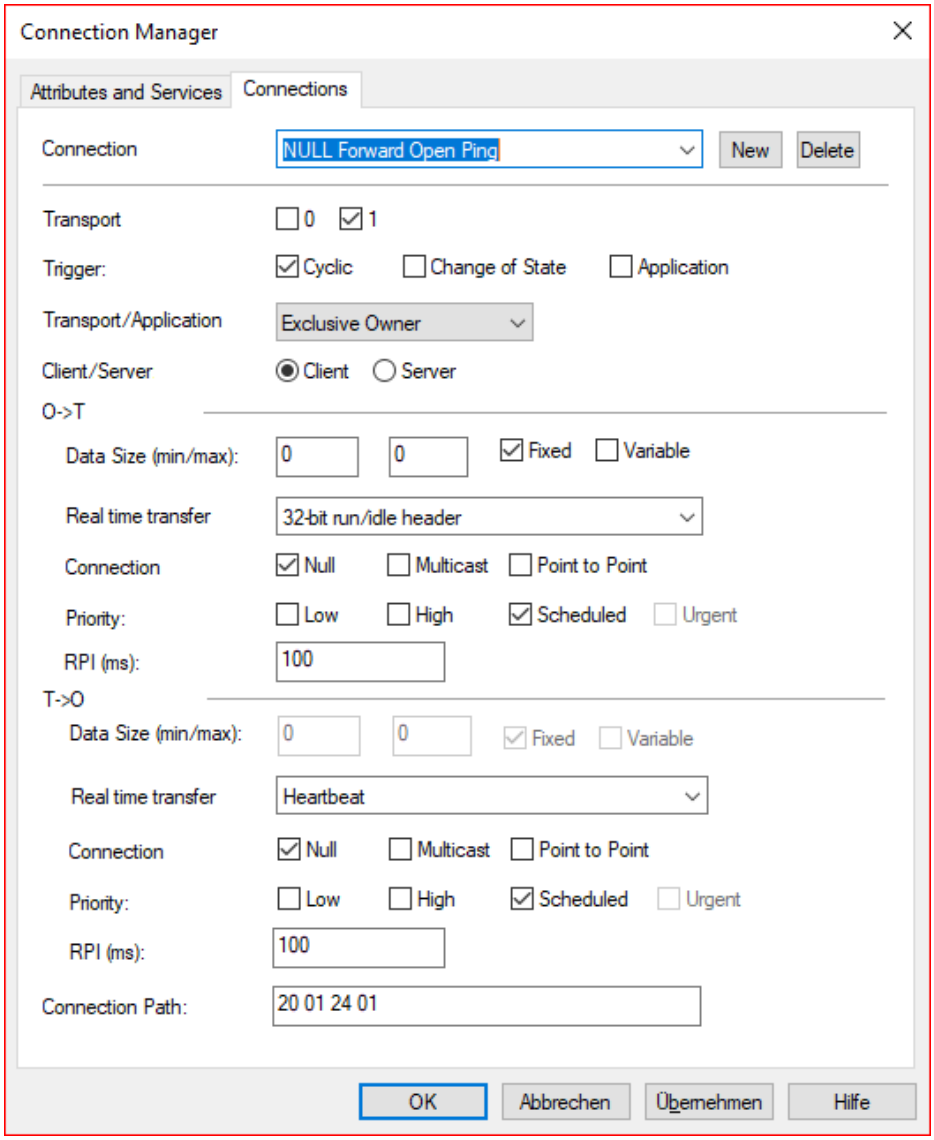

Figure 64: STC: connection entry for Null Forward Open - "Ping"

For the Null Forward Open “Configure” / “Re-Configure”

Chose the connection entry that shall be “reconfigurable” and set the following check marks in the connection properties:

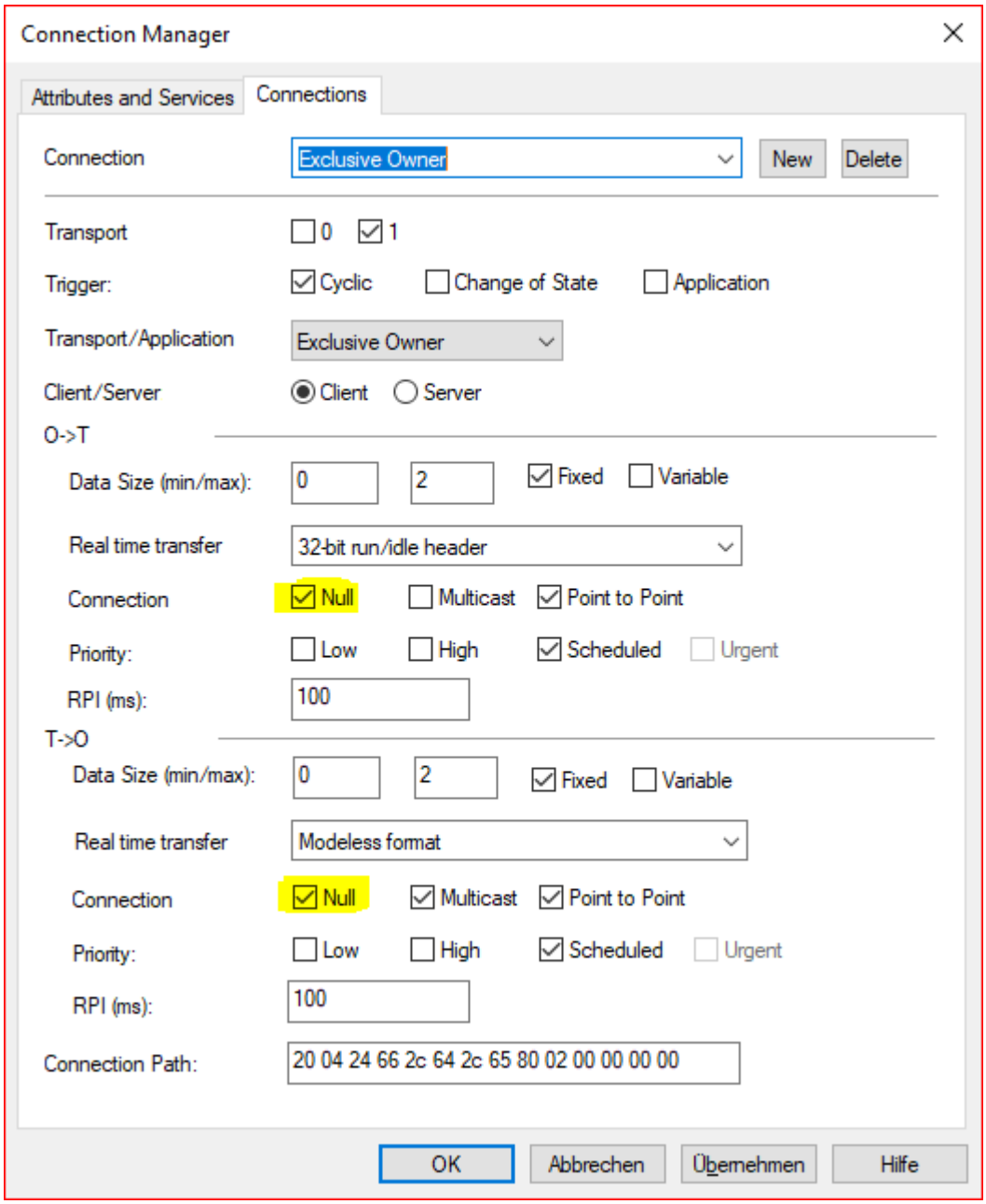

Figure 65: STC: connection entry for Null Forward Open - "Re-Configuration"

9.6 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.

9.7 List of Tables

9.8 List of Figures

9.9 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