BS EN 13321-2:2012

BSI Standards Publication

Open Data Communication in Building Automation, Controls and Building Management — Home and Building Electronic Systems

Part 2: KNXnet/IP Communication

bsi.

...making excellence a habit.™

National foreword

This British Standard is the UK implementation of EN 13321-2:2012. It supersedes BS EN 13321-2:2006 which is withdrawn.

The UK participation in its preparation was entrusted to Technical Committee RHE/16, Performance requirements for control systems.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Published by BSI Standards Limited 2013

ISBN 978 0 580 77057 9

ICS 35.240.99; 97.120

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 31 January 2013.

Amendments issued since publication

Date	Text affected

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 13321-2

December 2012

ICS 35.240.99; 97.120

Supersedes EN 13321-2:2006

English Version

Open Data Communication in Building Automation, Controls and Building Management - Home and Building Electronic Systems - Part 2: KNXnet/IP Communication

Réseau ouvert de communication de données pour l'automatisation, la régulation et la gestion technique du bâtiment - Systèmes électroniques pour la maison et le bâtiment - Partie 2: Communication KNXnet/IP

Offene Datenkommunikation für die Gebäudeautomation und Gebäudemanagement - Elektrische Systemtechnik für Heim und Gebäude - Teil 2: KNXnet/IP-Kommunikation

This European Standard was approved by CEN on 30 September 2012.

CEN members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration. Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CEN member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CEN member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CEN members are the national standards bodies of Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.

EUROPEAN COMMITTEE FOR STANDARDIZATION
COMITÉ EUROPÉEN DE NORMALISATION
EUROPÄISCHES KOMITEE FÜR NORMUNG

Management Centre: Avenue Marnix 17, B-1000 Brussels

Ref. No. EN 13321-2:2012: E

Contents

Page

Foreword

This document (EN 13321-2:2012) has been prepared by Technical Committee CEN/TC 247 “Building Automation, Controls and Building Management”, the secretariat of which is held by SNV.

This European Standard shall be given the status of a national standard, either by publication of an identical text or by endorsement, at the latest by June 2013, and conflicting national standards shall be withdrawn at the latest by June 2013.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. CEN [and/or CENELEC] shall not be held responsible for identifying any or all such patent rights.

This document supersedes EN 13321-2:2006.

Whereas ENV 13321-2:2000 described the transmission of EIB packets over Ethernet including the frame encoding, this document describes the transmission of HBES packets using the Internet Protocol. Details of the HBES packet frames are covered in part 1 of EN 13321, removing the need to explicitly describe the HBES frames in this document.

This document is Part 2 of the EN 13321 series of European Standards under the general title *Open data communication in building automation, controls and building management — Home and building electronic systems*, which consists of the following parts:

— *Part 1: Product and system requirements;*

— *Part 2: KNXnet/IP communication*.

According to the CEN/CENELEC Internal Regulations, the national standards organisations of the following countries are bound to implement this European Standard: Austria, Belgium, Bulgaria, Croatia, Cyprus, Czech Republic, Denmark, Estonia, Finland, Former Yugoslav Republic of Macedonia, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and the United Kingdom.

Introduction

This European Standard is intended for the design of new buildings and the retrofit of existing buildings in terms of acceptable indoor environment, practical energy conservation and efficiency.

This standard defines the integration of KNX protocol implementations within the Internet Protocol (IP) named KNXnet/IP.It defines a standard protocol, which is implemented within KNX devices, Engineering Tool Software (ETS) and other implementations to support KNX data exchange over IP networks. In fact, KNXnet/IP provides a general framework, which accommodates several specialised “Service Protocols” in a modular and extendible fashion.

The KNXnet/IP standard consists of the following clauses:

— Clause 1, Overview

— Clause 2, Core Specification

— Clause 3, Device Management

— Clause 4, Tunnelling

— Clause 5, Routing

— Clause 6, Remote Diagnosis and Configuration

Additional clauses may be added to the KNXnet/IP standard in the future at which time Clause 1 “Overview” as well as Annex A will need to be updated.

KNXnet/IP supports different software implementations on top of the protocol. More specifically, these software implementations can be Building Management, Facility Management, Energy Management, or simply Data Base and SCADA (Supervision, Control and Data Acquisition) packages.

Most of these packages need to be configured for the specific user application. In order to simplify this process and cut costs for engineering, KNXnet/IP provides simple engineering interfaces, namely a description “language” for the underlying KNX system. This may be done off-line, e.g. generated as an ETS export file, or on-line by a mechanism that self-describes the underlying KNX system (reading data from the system itself).

In conjunction with the EIB/KNX-to-BACnet mapping described in EN ISO 16484-5, EIB/KNX installations can very easily be integrated into BACnet system environments.

KNXnet/IP supports:

— on-the-fly change-over between Operational modes (configuration, operation);

— event driven mechanisms;

— connections with a delay time greater than $t_{EIB\_transfer\_timeout}$ (e.g. network connection via satellite).

Clause 1, Overview

Clause 1 “Overview” provides a general overview of KNXnet/IP and covers security considerations.

Clause 2, Core specification

Clause 2 “Core Specification” defines a standard protocol which is implemented within KNXnet/IP devices and Engineering Tool Software to support KNX data exchange over IP networks.

This specific implementation of the protocol over the Internet Protocol (IP) is called KNXnet/IP.

This standard addresses:

— definition of data packets sent over the IP host protocol network for KNXnet/IP communication;

— discovery and self-description of KNXnet/IP servers;

— configuration and establishment of a communication channel between a KNXnet/IP client and a KNXnet/IP server.

Clause 3, Device Management

Clause 3 “Device Management” defines services for remote configuration and remote management of KNXnet/IP servers.

Clause 4, Tunnelling

Clause 4 “Tunnelling” defines services for point-to-point exchange of KNX telegrams over an IP network between a KNXnet/IP device acting as a server and a KNXnet/IP Client. This point-to-point exchange may be established by a super ordinate system for building automation or management functions or by an Engineering Tool Software. It supports all ETS functions for download, test, and analysis of KNX devices on KNX networks connected via KNXnet/IP servers. This includes changes of single KNX device object properties.

Tunnelling assumes that a data transmission round-trip between a KNXnet/IP Tunnelling client and KNXnet/IP servers takes less than $t_{KNX\_transfer\_timeouts}$.

Clause 5, Routing

Clause 5 “Routing” defines services for a point-to-multipoint exchange of KNX telegrams over an IP network between KNXnet/IP routers and/or KNX/IP devices.

Clause 6, Remote Diagnosis and Configuration

Clause 6 “Remote Diagnosis and Configuration” defines services for a point-to-point exchange of KNX telegrams over an IP network between KNXnet/IP routers and/or KNX/IP devices. The services provide means for diagnosing communication settings and for changing these remotely.

1 Scope

This European Standard defines the integration of KNX protocol implementations on top of Internet Protocol (IP) networks, called KNXnet/IP. It describes a standard protocol for KNX devices connected to an IP network, called KNXnet/IP devices. The IP network acts as a fast (compared to KNX transmission speed) backbone in KNX installations.

Widespread deployment of data networks using the Internet Protocol (IP) presents an opportunity to expand building control communication beyond the local KNX control bus, providing:

— remote configuration;

— remote operation (including control and annunciation);

— fast interface from LAN to KNX and vice versa;

— WAN connection between KNX systems (where an installed KNX system is at least one line).

A KNXnet/IP system contains at least these elements:

— one EIB line with up to 64 (255) EIB devices;
OR
one KNX segment (KNX-TP1, KNX-TP0, KNX-RF, KNX-PL110, KNX-PL132);

— a KNX-to-IP network connection device (called KNXnet/IP server);

and typically additional

— software for remote functions residing on e.g. a workstation (may be data base application, BACnet Building Management System, browser, etc.).

Figure 1 shows a typical scenario where a KNXnet/IP client (e.g. running ETS) accesses multiple KNX installed systems or KNX subnetworks via an IP network. The KNXnet/IP client may access one or more KNXnet/IP servers at a time. For subnetwork, routing server-to-server communication is possible.

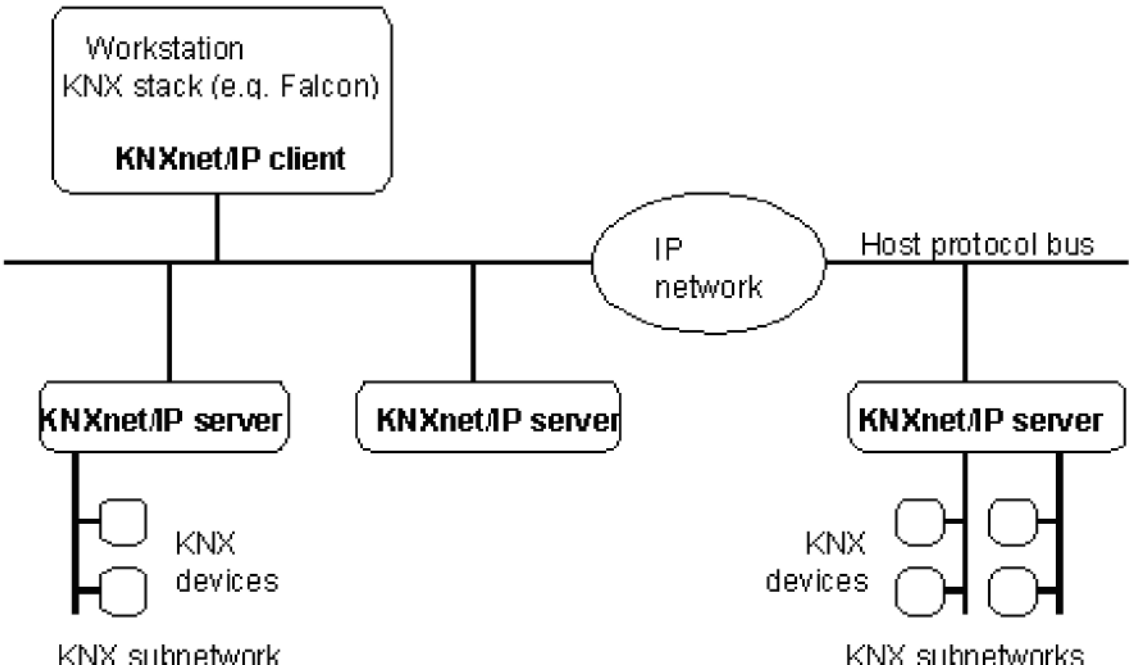


Figure 1 — Device types and configuration examples

2 Normative references

Not applicable.

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
subnet
portion of a network that shares a common address component known as the "subnet address"

Note 1 to entry: Different network protocols specify the subnet address in different ways.

3.2
Engineering Tool Software
ETS
software used to configure KNX devices

3.3
Host Protocol Address Information
HPAI
structure holding the IP host protocol address information used to address a KNXnet/IP endpoint on another KNXnet/IP device

3.4
communication channel
logical connection between a KNXnet/IP client and a KNXnet/IP server (or, in case of routing, between two or more KNXnet/IP servers)

Note 1 to entry: A communication channel consists of one or more connections on the definition of the host protocol used for KNXnet/IP.

3.5
KNX node
device implementing a KNX protocol stack and fulfilling the requirements for certification according to the KNX standard

3.6
KNXnet/IP server
KNX device with physical access to a KNX network implementing the KNXnet/IP server protocol to communicate with KNXnet/IP clients or other KNXnet/IP servers (in case of routing) on an IP network channel

Note 1 to entry: A KNXnet/IP server is by design always also a KNX node.

3.7
KNXnet/IP client
application using the KNXnet/IP client protocol to get access to a KNX subnetwork over an IP network channel

3.8
KNXnet/IP device
implementation of KNXnet/IP services on a KNX node (KNXnet/IP server) or any other hardware (KNXnet/IP client)

3.9
KNXnet/IP router
special type of KNXnet/IP device that routes KNX protocol packets between KNX sub-networks

3.10
Time To Live
TTL
maximum number of IP routers a multicast UDP/IP datagram may be routed through

Note 1 to entry: Each IP router the datagram passes decrements the TTL by one; the local host adapter also does this. When the TTL has reached zero, the router discards the datagram. When sending a datagram from the local host adapter, a TTL of zero means that the datagram never leaves the host. A TTL of one means that the datagram never leaves the local network (it is not routed).

3.11
KNXnet/IP Tunnelling
services for point-to-point exchange of KNX telegrams over an IP network between a KNXnet/IP device acting as a server and a KNXnet/IP client

3.12
Internet Control Message Protocol
ICMP
extension to the Internet Protocol (IP) for error, control, and informational messages

Note 1 to entry: ICMP is defined by RFC [1)] 92 and supports packet containing error, control, and informational messages. The PING command, for example, uses ICMP to test an Internet connection.

3.13
Internet Group Management Protocol
IGMP
extension to the Internet Protocol (IP) for management of IP multicasting in the Internet

Note 1 to entry: IGMP is defined in RFC 1112 as the standard for IP multicasting in the Internet. It is used to establish host memberships in particular multicast groups on a single network. By using Host Membership Reports, the

1) Request for Comment: Internet Standards defined by the Internet Engineering Task Force (IETF) are firstly published as RFCs.

mechanisms of the protocol allow a host to inform its local router that it wants to receive messages addressed to a specific multicast group. All hosts conforming to level 2 of the IP multicasting specification require IGMP.

3.14
IP channel
logical connection between two IP host/port endpoints

Note 1 to entry: IP channels are either a guaranteed, reliable TCP (transmission control protocol) or an unreliable point-to-point or multicast (in case of routing) UDP (user datagram protocol) connection.

3.15
communication channel
as defined by the KNXnet/IP Core specification, this is represented by one or two IP channels

3.16
common External Message Interface
cEMI
generic structure for medium independent KNX messages

Note 1 to entry: cEMI (common EMI) frames are used to encapsulate KNX messages within Internet Protocol (IP) packets.

4 Symbols and abbreviations

For the purpose of this document, the symbols, abbreviations and acronyms used are listed below.

Tables listing implementation requirements use the following abbreviations.

Symbol	Description
M	Mandatory
C^n	Conditions are specified under note "n"
O	Optional
X	Not allowed
n/a	Not applicable
R	Required
MC	Message code
AddIL	length of additional information

4.1
DHCP
Dynamic Host Configuration Protocol
communication protocol for automatic assignment of IP address settings

4.2
DNS
Domain Name Service
assigns Internet names to IP addresses

4.3
EIB
European Installation Bus
Standard for Building Controls (EN 50090)

4.4
IP
Internet Protocol

4.5
KNX
Standard for Building Controls (EN 50090)

4.6
TCP/IP
Transmission Control Protocol over Internet Protocol
connection-oriented communication over the Internet

4.7
UDP/IP
User Datagram Protocol over Internet Protocol
connection-less communication over the Internet

5 Requirements

5.1 Clause 1: Overview

5.1.1 KNXnet/IP Document Clauses

5.1.1.1 General

This European Standard defines the integration of KNX protocol implementations within the Internet Protocol (IP) named KNXnet/IP or EIBnet/IP for continuity with ENV 13321-2:2000 (EIBnet) as defined by CEN/TC 247. EIBnet was introduced as an expansion of EIB into the information technology realm and was incorporated as a building controls technology in CEN/TC 247. EIBnet/IP is the logical successor to EIBnet. As EIB has become a part of KNX, this standard is called KNXnet/IP.

This European Standard defines a standard protocol, which is implemented within KNX devices, Engineering Tool Software and other implementations to support KNX data exchange over IP networks. In fact, KNXnet/IP provides a general framework, which accommodates several specialised “Service Protocols” in a modular and extendible fashion.

The KNXnet/IP standard consists of the following clauses:

— Clause 1, Overview;

— Clause 2, Core Specification;

— Clause 3, Device Management;

— Clause 4, Tunnelling;

— Clause 5, Routing.

Additional clauses may be added to the KNXnet/IP standard in the future at which time this Clause 1 “Overview” shall be updated.

KNXnet/IP supports different software implementations on top of the protocol. More specifically, these software implementations can be Building Management, Facility Management, Energy Management, or simply Data Base and SCADA (Supervision, Control and Data Acquisition) packages.

Most of these packages need to be configured for the specific user application. In order to simplify this process and cut costs for engineering, KNXnet/IP provides simple engineering interfaces, namely a

description “language” for the underlying KNX system. This may be done off-line, e.g. generated as an ETS export file, or on-line by a mechanism that self-describes the underlying KNX system (reading data from the system itself).

KNXnet/IP supports:

— on-the-fly change-over between operational modes (configuration, operation);

— event driven mechanisms;

— connections with a delay time greater than $t_{EIB\_transfer\_timeout}$ (e.g. network connection via satellite).

5.1.1.2 Clause 1, Overview

Clause 1 "Overview" is this document.

5.1.1.3 Clause 2, Core specification

Clause 2 “Core Specification” defines a standard protocol, which is implemented within KNXnet/IP devices and the Engineering Tool Software (ETS) to support KNX data exchange over IP networks.

This specific implementation of the protocol over the Internet Protocol (IP) is called KNXnet/IP.

This standard addresses:

— definition of data packets sent over the IP host protocol network for KNXnet/IP communication;

— discovery and self-description of KNXnet/IP servers;

— configuration and establishment of a communication channel between a KNXnet/IP client and a KNXnet/IP server;

5.1.1.4 Clause 3, Device Management

Clause 3 “Device Management” defines services for remote configuration and remote management of KNXnet/IP servers.

5.1.1.5 Clause 4, Tunnelling

Clause 4 “Tunnelling” defines services for point-to-point exchange of KNX telegrams over an IP network between a KNXnet/IP device acting as a server and a KNXnet/IP Client. This point-to-point exchange may be established by a super ordinate system for building automation or management functions or by an Engineering Tool Software. It supports all ETS functions for download, test, and analysis of KNX devices on KNX networks connected via KNXnet/IP servers. This includes changes of single KNX device object properties.

Tunnelling assumes that a data transmission round-trip between ETS or a KNXnet/IP Tunnelling client and KNXnet/IP servers takes less than $t_{KNX\_transfer\_timeouts}$.

5.1.1.6 Clause 5, Routing

Clause 5 “Routing” defines services, which route KNX telegrams between KNXnet/IP servers through the IP network.

5.1.2 Mandatory and optional implementation of IP protocols

5.1.2.1 General

KNXnet/IP uses existing IP protocols where possible unless their use implies an undue burden with regard to memory and implementation requirements for the intended service.

The following table shows mandatory (M) and optional (O) implementation of IP protocols by KNXnet/IP service types. Although this table refers to the KNXnet/IP server, it also indicates which IP protocols shall be implemented by the KNXnet/IP client. Any non-applicable IP protocol is marked as "na".

Table 1 — KNXnet/IP service types and IP protocols

IP protocol	Service Type			
	Core	Device Management	Tunnelling	Routing
ARP	M	M	M	M
RARP	O	O	O	O
Support of fixed IP address	M	M	M	M
BootP (Client)[a]	M	M	M	M
DHCP (Client)[a]	M	M	M	M
UDP	M	M	M	M
TCP	O	O	O	na
ICMP	M	M	M	M
IGMP	M	M	na	M

[a] BootP/DHCP: It is essential that either one be implemented by a KNXnet/IP device.

Other Internet protocols like NTP (network time protocol), FTP (file transfer protocol), HTTP (hypertext transfer protocol), SMTP (simple message transfer protocol), DNS (domain name system), and SNMP (simple network management protocol) may be used but are not within the scope of the KNXnet/IP protocol.

5.1.2.2 Minimum KNXnet/IP device requirements

KNXnet/IP service types as defined in this standard require the implementation of a minimal set of IP protocols for interworking.

KNXnet/IP servers shall implement these IP protocols: ARP, BootP, UDP, ICMP and IGMP. Other IP protocols may be required for specific services.

5.1.2.3 Network environment

Because KNXnet/IP servers use IP, this standard does not require any specific medium carrying the IP datagrams.

It is RECOMMENDED to use a medium, which carries at least twice the bit rate of all KNXnet/IP routers connected to this medium. For a point-to-point, e.g. PPP, connection this would be at least 19 200 bit/s; for a network with up to 400 KNXnet/IP servers, this would be least 8 Mbit/s.

10BaseT is RECOMMENDED as a minimum for KNXnet/IP servers using Ethernet as physical medium.

5.1.2.4 Addressing

KNXnet/IP servers are typically connected to one KNX subnetwork and to an IP network. Hence, KNXnet/IP servers have one distinct address for each network they are connected to: one KNX/EIB Individual Address and one IP address.

Additionally, KNXnet/IP servers are assigned to a KNXnet/IP project-installation using the same IP multicast address for all KNXnet/IP servers in a KNXnet/IP project-installation.

5.1.2.5 KNXnet/IP device classes

A KNXnet/IP device can be a software implementation running on a PC. Hence, ETS or any other software implementing KNXnet/IP services is viewed as a KNXnet/IP device.

Definition of KNXnet/IP device classes ensures interoperability between KNXnet/IP devices as well as a minimum set of supported KNXnet/IP services for a given KNXnet/IP device class.

Table 2 lists mandatory and optional service types for device classes.

Device Class A encompasses configuration and system maintenance tools. Except for Object Server services, all other KNXnet/IP services shall be implemented. ETS is a member of this device class.

Device Class B defines the minimum set of KNXnet/IP services for KNXnet/IP routers.

Device Class C defines the minimum set of KNXnet/IP services for any other KNXnet/IP device. Building, energy and facilities management systems are members of this KNXnet/IP device class.

Table 2 — KNXnet/IP device classes

Device Class	**Service Type**			
	Core	**Device Management**	**Tunnelling**	**Routing**
A (Configuration and System Maintenance Tools)	M	M	M	M
B (KNXnet/IP Router)	M	M	M	M
C (any other KNXnet/IP device)	M	M	O	O

5.1.3 Security considerations

5.1.3.1 Introduction

For KNX, security was and is a minor concern, as any breach of security requires local access to the network. In the case of KNX TP (EIB), and KNX PL networks, this requires even physical access to the network wires, which in nearly all cases is impossible as the wires are inside a building or buried underground.

Hence, security aspects are less of a concern for KNX field level media.

The intention of KNXnet/IP is to provide a means of using existing data networking technology, i.e. the Internet Protocol, for fast communication between different KNX subnetworks in the same or different locations.

Unless the data network is completely isolated and only used for KNXnet/IP, a number of potential threats need to be considered.

5.1.3.2 Categories of threats

5.1.3.2.1 General

This section briefly enumerates various types of security threats and mentions typical countermeasures.

5.1.3.2.2 Passive attacks

Passive attacks involve a violation of privacy or leak of information to an unauthorised party that “wiretaps” or listens in on the network. Passive attacks are difficult to detect, since they do not modify the network traffic in any way. The primary defence against passive attacks is isolation of network traffic (one cannot eavesdrop or analyse traffic that one cannot see):

— Eavesdropping – interception and examination of packet contents by unauthorised third parties. In addition to network isolation, this threat can be countered by encrypting packets.

— Traffic analysis – interception and analysis of traffic characteristics without examining the contents of individual packets. This threat exists even if packets are encrypted, and it is very difficult to counter except by isolation of network traffic.

5.1.3.2.3 Active attacks

Active attacks involve some type of modification to network traffic: adding, deleting, or delaying packets, or changing the contents of packets in some way. Types of active attack are listed below:

— Masquerade – unauthorised entities forging traffic to make it appear to originate from a legitimate source. This threat can be countered by authentication mechanisms.

— Access control violation – a user accesses a resource that they do not have proper authorisation to use. This threat can be countered by message integrity, authentication, and authorisation mechanisms (the access control policies shall also be secured).

— Modification – intercepting and changing the contents of packets on the network. This threat can be countered by message integrity mechanisms.

— Deletion – preventing packets from arriving at their intended destination (for example, by maliciously configuring a router to drop packets or by deliberately interfering with packet transmissions on the wire). This threat can be very difficult to counter, especially if the attacker can jam packet transmissions on the physical network.

— Delay – delaying arrival of packets at their intended destination. This attack can be achieved by compromising or overloading a router or bridge or by performing a time-limited attack on the packet transmission (thereby causing retransmission delays). This threat is also very difficult to counter.

— Replay – intercepting packets and later resending them on the network. This threat can be countered by authentication, message integrity, and timestamps or message counters.

— Denial of service – this type of attack typically involves injecting a flood of useless traffic on the network to consume resources such as bandwidth or processing time so that the resources are unavailable to service legitimate traffic. This threat is difficult to counter, especially if the attack traffic cannot be easily distinguished from legitimate traffic.

5.1.3.3 Countermeasures

The following is a list of countermeasures for KNXnet/IP:

a) Typically, building control data is only exchanged in a closed or tightly surveillance network. Use KNXnet/IP in an Intranet or over Virtual Private Network only.

b) Filtering KNXnet/IP datagrams from the network requires network analysis tools and expertise. The content of a KNXnet/IP message is not self-descriptive but requires semantic knowledge residing in ETS. Access to ETS is limited to authorised personnel only.

c) Use authentication when opening point-to-point connections e.g. for tunnelling or remote logging.

d) Do not publish default KNXnet/IP IP multicast address.

e) Use sequence counter on all point-to-point connections e.g. for tunnelling or remote logging.

f) Use UDP/IP or TCP/IP port numbers, which are filtered by a firewall.

Table 3 — Threats and countermeasures

Threat…	**Is countered by measure(s)**
Eavesdropping	a), b), d)
Traffic analysis	a), d)
Masquerade	a), c), d)
Access control violation	a), c), d)
Modification	a), e)
Deletion	a)
Delay	a)
Replay	c), e)
Denial of service	a)

5.1.3.4 Conclusion

Most of the listed threats are countered by staying within a network, which tightly supervises and restricts network access from outside the network.

Using authentication when establishing point-to-point connections is an additional means to this end.

It is quite unlikely that legitimate users of a network would have the means to intercept, decipher, and then tamper with the KNXnet/IP without excessive study of the KNX standards.

Thus the remaining security threat is considered to be very low and does not justify mandating encryption, which would require considerable computing resources.

5.2 Clause 2: Core

5.2.1 Scope

This Clause "Core" of the KNXnet/IP standard provides a general host protocol-independent framework, which accommodates several specialised “Service Protocols” in a modular and extendible fashion.

This standard addresses:

— definition of data packets sent over the non-KNX “host protocol” network for KNXnet/IP communication;

— discovery and self-description of KNXnet/IP servers; and

— configuration, establishment and maintenance of a communication channel between a KNXnet/IP client and a KNXnet/IP server.

5.2.2 KNXnet/IP frames

5.2.2.1 General definitions

5.2.2.1.1 Data format

The KNXnet/IP frames described within this standard are coded binary.

The data structure specifications are always noted in binary format.

5.2.2.1.2 Byte order

For binary structures, if not explicitly stated otherwise, the byte order shall be big endian mode (Motorola, non-swapped). For plain text formats, the byte order and formatting shall be according to the related protocol specifications.

5.2.2.1.3 Structures

All KNXnet/IP structures follow a common rule: the first octet is always the length of the complete structure (as some structures may have fields of variable length e.g. strings) and the second octet is always an identifier that specifies the type of the structure. From the third octet on, the structure data follows. If the amount of data exceeds 252 octets, the length octet shall be FFh and the next two octets will contain the length as a 16 Bit value. Then the structure data starts at the fifth octet.

5.2.2.2 Frame format

The communication between KNXnet/IP devices is based on KNXnet/IP frames. A KNXnet/IP frame is a data packet sent over the non-KNX network protocol that consists of a header, comparable to the IP header of an internet protocol data packet and an optional body of variable length.

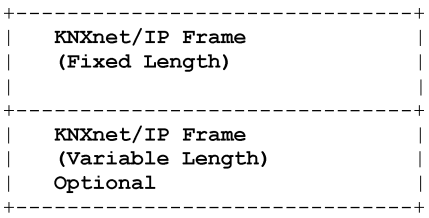


Figure 2 — KNXnet/IP frame binary format

The type of KNXnet/IP frame is described by a KNXnet/IP service type identifier in the header. KNXnet/IP services include, but are not limited to, information regarding discovery and description, communication channel (connection) management and KNX data transfer.

5.2.2.3 Header

5.2.2.3.1 Description

Every KNXnet/IP frame, without any exception, consists at least of the common KNXnet/IP header that contains information about the protocol version, the header and total packet length and the KNXnet/IP service type identifier. The KNXnet/IP header may be followed by a KNXnet/IP body, depending on the KNXnet/IP service.

Timestamp information and frame counters are not included in the common KNXnet/IP frame header as this information is closely linked with certain KNXnet/IP service types and will therefore be included in the body of these services as additional information for certain communication channel types.

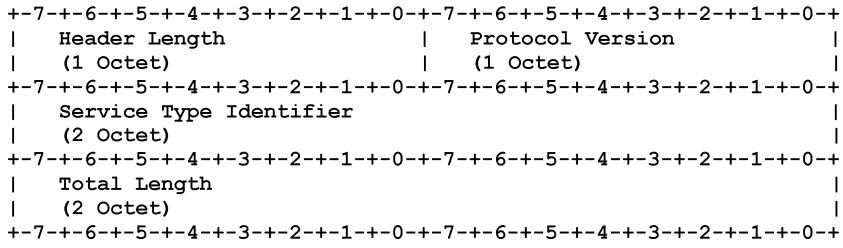


Figure 3 — KNXnet/IP header binary format

5.2.2.3.2 Header length

Although the length of the header is always fixed, it is possible that the size of the header changes with a new version of the protocol. The header length can be used as an index into the KNXnet/IP frame data to find the beginning of the KNXnet/IP body.

5.2.2.3.3 Protocol version

The protocol version information states the revision of the KNXnet/IP protocol that the following KNXnet/IP frame is subject to. It will be stored in binary coded decimal format. The only valid protocol version at this time is 1.0 (represented as hexadecimal 10h).

5.2.2.3.4 KNXnet/IP service

The KNXnet/IP service type identifier defines the kind of action to be performed and the type of the data payload contained in the KNXnet/IP body if applicable. The high octet of the KNXnet/IP service type identifier denotes the service type family and the low octet the actual service type in that family. For a detailed description of the services, see below.

5.2.2.3.5 Total length

The total length is expressing the total KNXnet/IP frame length in octets. The length includes the complete KNXnet/IP frame, starting with the header length of the KNXnet/IP header and including the whole KNXnet/IP body.

5.2.3 Host protocol independence

5.2.3.1 Host protocol

The KNXnet/IP core specification defines the integration of KNX protocol implementations on top of the Internet Protocol (IP). It describes the general and host protocol independent as well as the host protocol specific parts of the KNXnet/IP communication.

5.2.3.2 Endpoints

KNXnet/IP defines the Host Protocol Address Information (HPAI) structure, which is the combination of IP address and port number. The HPAI is the data required to send a KNXnet/IP frame to another device. The KNXnet/IP standard uses the term KNXnet/IP endpoint as a logical view of a HPAI to address another KNXnet/IP device for certain well-defined purposes:

Every KNXnet/IP device shall support exactly one device related, bi-directional and connectionless endpoint for discovery if the host protocol requires discovery services. It shall support at least one bi-directional and connectionless endpoint for controlling and at least one bi-directional and connection oriented endpoint for service type related data transmission.

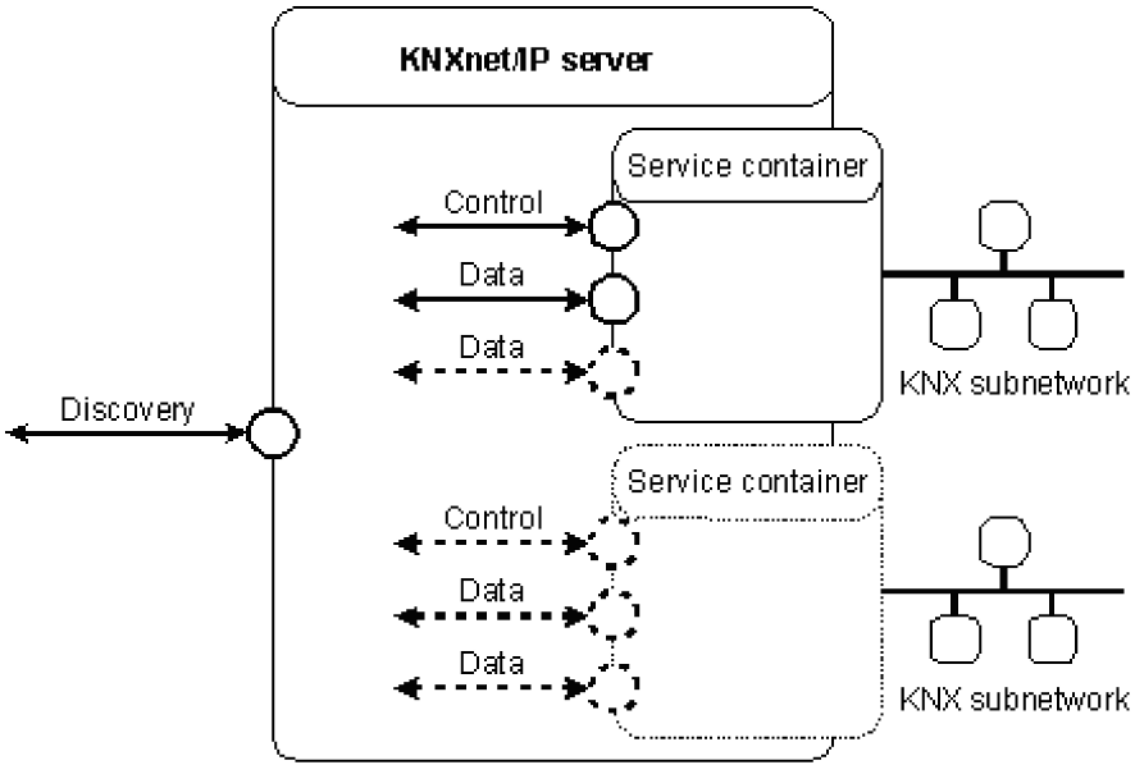


Figure 4 — KNXnet/IP server endpoints sample configuration

The control endpoint uniquely addresses one entity inside the KNXnet/IP server device that shall be capable of providing at least one KNXnet/IP service type.

This entity, called service container, may be connected to a KNX Subnetwork. If the KNXnet/IP server device supports more than one KNX Subnetwork connection, it is REQUIRED that every KNX Subnetwork shall be

represented by a different control endpoint. The KNXnet/IP client therefore considers every service container represented by a control endpoint as one independent entity no matter if they are implemented in only one or two separate KNXnet/IP server devices.

These KNXnet/IP endpoints present a logical view to the communication of a KNXnet/IP device. The actual implementation of these endpoints with different host protocols may use transport medium dependent approaches that differ from this logical view. For example, the bi-directional KNXnet/IP endpoints could be implemented using two unidirectional channels with the host protocol. Therefore, it is possible for one KNXnet/IP endpoint to be represented by multiple HPAIs.

5.2.4 Discovery and self description

5.2.4.1 Introduction

Particularly for networks supporting hot-plug, and where even the address assignment may take place at runtime (e.g. IP address assignment via BootP or DHCP), it is of significant importance to search for devices within a subnetwork without having the need to retrieve network parameters through a non-standardised way and manually input them in the client tool to establish a connection. Furthermore, to get a precise picture of the services supported by the KNXnet/IP server without implementing trial and error, a self-description mechanism is an important feature.

5.2.4.2 Discovery

Any KNXnet/IP server shall implement discovery according to this procedure. If applicable for the host protocol, it is recommended that a KNXnet/IP client implementation supports searching for KNXnet/IP servers instead of requiring manual input.

The discovery operation consists of a SEARCH_REQUEST data packet, sent via multicast using the discovery endpoint, which contains the HPAI of the KNXnet/IP client's discovery endpoint. The HPAI may contain a unicast IP address to receive the answers from the different KNXnet/IP servers directly in a point-to-point manner. Typically, it should contain the KNXnet/IP System Setup multicast address to ensure reception from KNXnet/IP servers that are on a different subnetwork.

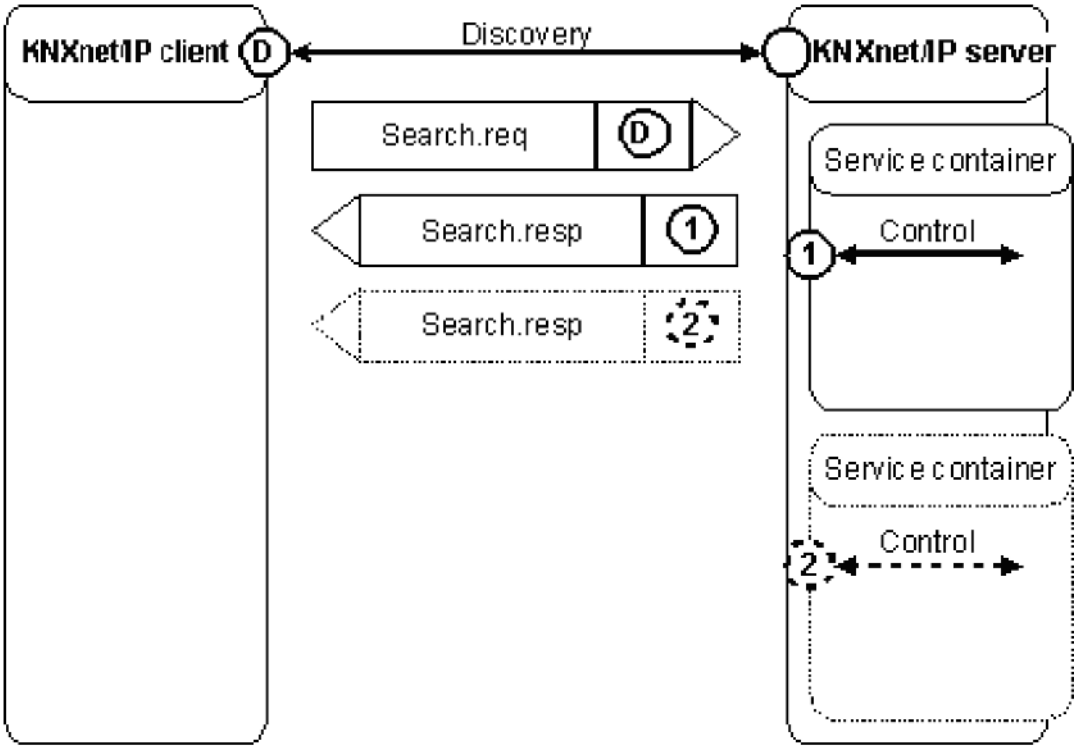


Figure 5 — Discovery procedure

After sending the request, the KNXnet/IP client shall wait for time SEARCH_TIMEOUT for SEARCH_RESPONSE frames from KNXnet/IP servers. After that period of time, any received SEARCH_RESPONSE frame shall be ignored by that client until it starts another discovery cycle. SEARCH_REQUEST frames received by clients from other clients shall be ignored.

Any KNXnet/IP server receiving a SEARCH_REQUEST service shall respond immediately with a SEARCH_RESPONSE data packet to the given HPAI using its discovery endpoint. Such a response contains only the HPAI of the KNXnet/IP server's control endpoint for all further communication.

Any KNXnet/IP server shall support discovery by processing search requests and sending correct responses. A KNXnet/IP server may support links to more than one KNX Subnetwork, it shall however send a SEARCH_RESPONSE data packet for the control endpoint of each KNX Subnetwork it supports connections to, even if it supports only one data connection at a time.

5.2.4.3 Self-description

Typically, after discovering a KNXnet/IP server, the KNXnet/IP client sends a DESCRIPTION_REQUEST through a unicast or point-to-point connection to all control endpoints of the KNXnet/IP server. It is REQUIRED that every KNXnet/IP implementation supports description requests. Furthermore, before a KNXnet/IP client communicates with a KNXnet/IP server, it should check if the server supports the services requested by the client using the self-description mechanism.

If a KNXnet/IP server receives a valid description request, it shall reply with a DESCRIPTION_RESPONSE packet providing information on the supported protocol version range, its own capabilities, state information and optionally a friendly name for this KNXnet/IP server's KNX connection. As a KNXnet/IP server may support links to more than one KNX Subnetwork, it shall support responding to discovery requests for each potential KNX Subnetwork connection announced by the discovery responses.

5.2.5 Communication Channels

5.2.5.1 Introduction

A communication channel is the data endpoint connection between a KNXnet/IP client and a KNXnet/IP server. Data endpoint connections are established for services requiring point-to-point communication e.g. tunnelling or device management. Any point-to-point connection between a KNXnet/IP client and a KNXnet/IP server shall be initiated by the client. Any KNXnet/IP server shall support at least one client connection at a time. It may support more than one client connection at a time, however it shall ensure that existing connections are not affected by accepting new connections (e.g. a KNXnet/IP server shall not accept a tunnelling connection on the same physical access to a KNX Subnetwork in different modes, link or busmonitor layer).

5.2.5.2 Establishing a link

To establish a link between a KNXnet/IP client and a KNXnet/IP server, the client shall send a CONNECT_REQUEST frame to the control endpoint of the server. This request provides information on the requested connection type (e.g. data tunnelling or remote logging), general and connection type specific options (e.g. link layer or busmonitor mode[2)]) and the data endpoint HPAI that the client wants to use for this communication channel.

Before sending a connection request of a specific type (with specific options), the KNXnet/IP client should check against the self-description information received from the KNXnet/IP server if the server supports the requested connection type and/or all the requested options.

The KNXnet/IP server shall then send a CONNECT_RESPONSE frame in any case back to the KNXnet/IP client requesting to establish the connection, providing the status of the request (ACK/NACK with extended status information if applicable). If the request could be accepted by the server, the CONNECT_RESPONSE

2) The list of supported layers and services is supposed to be extended in further versions.

frame contains additionally an identifier as well as the HPAI of the data endpoint that the server now prepared for this communication channel[3)]

After sending the connection request, the KNXnet/IP client shall wait for the host protocol dependent time CONNECT_REQUEST_TIMEOUT (= 10 s) for the response frame from KNXnet/IP server. After that period of time, any received response frames shall be ignored by that client until it starts another connection request.

The current protocol standard assumes that a connection shall not be shared by multiple clients. A KNXnet/IP server shall therefore not accept multiple connect requests of the same type on e.g. the same physical KNX connection, though it may of course implement numerous physical connections, exposing each logically as an independent KNXnet/IP server, if supported by the used host protocol. The client implementation can rely on that restriction and is not required to handle such a connection sharing scenario. Service family exceptions to this general rule are described in the corresponding service family KNXnet/IP chapter.

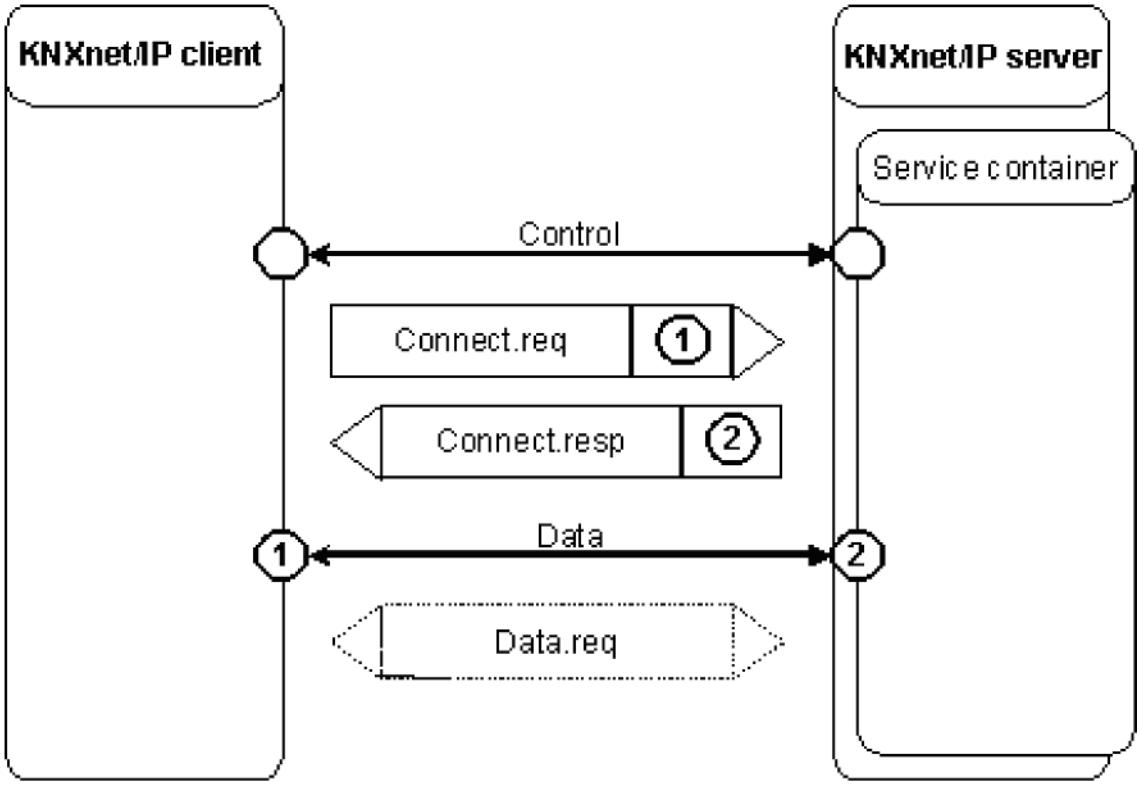


Figure 6 — Establishing a data connection

5.2.5.3 Connection Header

5.2.5.3.1 Description

The body of every KNXnet/IP frame sent over an established communication channel starts with data fields that contain additional general information about the data connection. The amount of this data and what type of information is included is determined by several options during the connection phase of a communication channel. The total of these data fields is called connection header and its appearance varies greatly depending on the already mentioned connection options. Only the order in which the different data fields are stored in the connection header is fixed.

3) It should be noted that an EIBnet/IP connection may consist due to technical reasons of multiple logical connections at the host protocol layer.

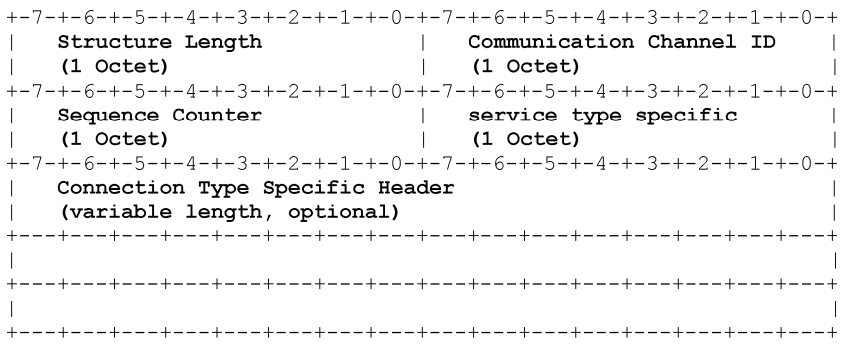


Figure 7 — Common connection header

5.2.5.3.2 Structure length

Structure Length is the total length of the connection header.

5.2.5.3.3 Communication Channel ID

The KNXnet/IP server assigns a Communication Channel ID to each established communication channel.

5.2.5.3.4 Sequence Counter

The sequence counter is maintained for each communication channel. It shall be incremented by one independently for every communication channel for each KNXnet/IP frame sent over the data connection. Both KNXnet/IP devices maintaining the communication channel have independent sequence counters.

Every time a connection is established, the counter for this connection shall be set to 0, so the first KNXnet/IP frame sent on an established communication channel has a sequence counter of 0.

5.2.5.3.5 Connection Type specific Header Items

The connection type specific header items are optional and of variable length depending on the type of the connection.

5.2.5.4 Heartbeat monitoring

Host protocols not providing mechanisms for lifetime check like UDP/IP need a procedure to identify failure of communication, may it be on the KNX or the tunnelling network. To detect such situations heartbeat monitoring is defined and shall be implemented by both KNXnet/IP clients and servers.

The client shall send a CONNECTIONSTATE_REQUEST frame regularly, i.e. every 60 s, to the server's control endpoint which shall respond immediately with a CONNECTIONSTATE_RESPONSE frame (this counts as heartbeat response).

If the KNXnet/IP client does not receive the heartbeat response within the CONNECTIONSTATE_REQUEST_TIMEOUT (= 10 s) or the status of a received heartbeat response signalled any kind of error condition, the client shall repeat the CONNECTIONSTATE_REQUEST three times and then terminate the connection by sending a DISCONNECT_REQUEST to the server's control endpoint.

If the KNXnet/IP server does not receive a heartbeat request within 120 s of the last correctly received message frame, the server shall terminate the connection by sending a DISCONNECT_REQUEST to the client's control endpoint. The server shall not retrigger the timeout after messages received with unexpected sequence number.

5.2.5.5 Disconnecting

Typically, the client terminates the connection. During normal operation, the client shall send a DISCONNECT_REQUEST to the server's control endpoint to request termination of the data channel connection.

The client should try to disconnect gracefully if possible, even under error conditions. The server may disconnect from the client by sending a DISCONNECT_REQUEST in case of internal problems or if it receives invalid data packets; however it is recommended to let the client terminate the connection.

The KNXnet/IP device receiving the DISCONNECT_REQUEST from the communication partner shall acknowledge the operation with a DISCONNECT_RESPONSE frame. This data packet signals the final termination of a previously established communication channel.

5.2.6 General implementation guidelines

5.2.6.1 Introduction

This section defines programming guidelines which must be taken into account when implementing KNXnet/IP servers or clients, respectively. Compliance with these guidelines is a requirement for the certification of the protocol implementation.

5.2.6.2 KNXnet/IP servers

— If a server receives a data packet with an unsupported version field, it shall reply with a NACK message indicating in the status field E_VERSION_NOT_SUPPORTED.

— If an invalid data packet is received, the implementation shall ignore the data packet without taking any further action.

— If a connection is established, all data packets shall be sent with the same protocol version.

— If a connection is established and the protocol version changes within the received data packets, the server shall shut down the connection.

5.2.6.3 KNXnet/IP clients

— If a client receives a data packet with an unsupported version field, it shall reply with a NACK message indicating in the status field E_VERSION_NOT_SUPPORTED. If a connection to that server sending with an unsupported protocol version is established, the client shall disconnect. The client may try to reconnect then and re-establish the connection.

— If an invalid data packet is received, the implementation shall ignore the data packet without taking any further action.

— If a connection is established, all data packets shall be sent with the same protocol version.

— If a connection is established and the protocol version changes within the received data packets, the client shall disconnect from the server. The client may try to reconnect then and re-establish the connection.

5.2.6.4 KNXnet/IP router settings

5.2.6.4.1 KNXnet/IP router factory default settings

An important feature for KNXnet/IP routers is that they shall provide proper KNXnet/IP Routing without any user intervention. This plug and play Routing behaviour requires a standardised factory default configuration.

— Routers shall be shipped with a default Individual Address of FF00h.

— The Routing multicast address equals the System Setup multicast address.

— KNX broadcast telegrams shall be routed from one KNX Subnetwork to another even if KNXnet/IP Routing devices are still being used in their factory default configuration.

— KNXnet/IP Routing shall already work even if a valid unicast IP address has not been obtained by or assigned to the KNXnet/IP Routing device.

— Routing shall already work and Tunnelling shall be available for (further) configuration of the KNXnet/IP router if an Individual Address has been assigned to a KNXnet/IP router via KNX Subnetwork.

5.2.6.4.2 KNXnet/IP router IP address assignment

KNXnet/IP routers shall support plug and play KNXnet/P routing out of the box even if a valid unicast IP address was not acquired from a DHCP server, by manual input, or via ETS configuration. This may require that the IP stack can send and receive multicast IP messages although a valid unicast IP address has not been acquired.

If an IP stack does not support multicasting without an assigned unicast IP address then the KNXnet/IP router shall acquire a unicast IP address via AutoIP or by self-assigning the default KNXnet/IP source unicast IP address 0.0.0.0.

If a valid unicast IP address was not acquired the KNXnet/IP router shall use the default KNXnet/IP source unicast IP address 0.0.0.0 for Routing but shall not support KNXnet/IP Tunnelling.

5.2.6.5 Initial setup procedures for KNXnet/IP Servers

5.2.6.5.1 General

KNXnet/IP devices shall be configurable in the same way as traditional KNX devices. In an unconfigured state KNXnet/IP routers shall operate with default values enabling KNX telegrams to pass from one KNX Subnetwork to another, enabling KNXnet/IP routers to transparently replace KNX routers (Line and Backbone Couplers).

This requirement is set under the condition that KNX Individual Addresses are unique across the IP network. If KNXnet/IP devices of two independent installations are connected to the same IP network or a KNX project consists of multiple installations, the use of one factory default routing configuration cannot guarantee the delivery of KNX telegrams to the intended destination as the same KNX (Individual or Group) Address could be present in different installations.

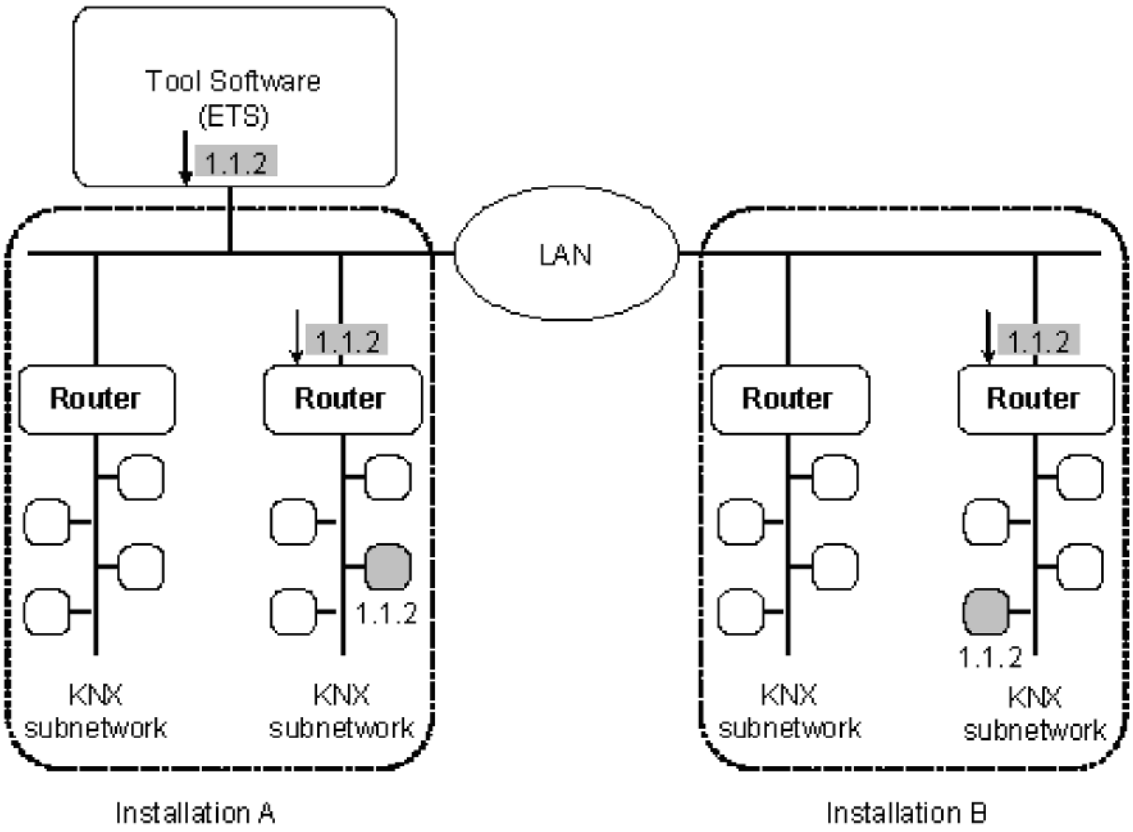


Figure 8 — KNX project with multiple installations

Under the precondition that only one installation is connected to the IP network the following rules apply:

1) Upon power-up, the KNXnet/IP device steps through the procedure described in KNXnet/IP Core, 5.2.8.5, IP Address Assignment, to retrieve an individual IP address.

2) If connected to a KNX Serial Interface device (RS232) or KNX USB Interface, the tool software can only access KNX devices on the local KNX subnetwork (including the local KNXnet/IP router) as long as not all the KNXnet/IP router devices belonging to one project are successfully configured.

3) The tool software SHOULD always attempt to configure KNXnet/IP router devices first to gain access to KNX devices behind possibly still unconfigured KNXnet/IP routers.

4) If directly connected to the IP network, the tool software can access all KNXnet/IP devices and through configured routers the corresponding KNX Subnetworks.

5) The tool software SHOULD use the IP network for the setup of projects containing KNXnet/IP devices.

6) It is possible to completely configure KNXnet/IP devices in one single configuration procedure (KNX Individual Address assignment and parameter download).

7) Assignment of KNX Individual Address: If the tool software is connected through a KNX Serial Interface device (RS232) or KNX USB Interface to the KNX network, the traditional KNX management procedure applies for KNXnet/IP devices as well as other KNX nodes. Devices behind unconfigured routers are not reachable. If the tool software is connected to the IP network, the configuration procedure for KNXnet/IP devices (described below) shall apply. Before setting up KNX field media devices the tool software shall firstly configure all KNXnet/IP router devices.

8) Parameter download: if the tool software is connected through a serial interface to the KNX network, the traditional KNX management procedure applies for KNXnet/IP devices as well as other KNX nodes. If the tool software is connected to the IP network, it establishes management connections to every KNXnet/IP device to download the device parameters (see configuration procedure). After having configured all KNXnet/IP router devices, the tool software can access all other KNX nodes of the project for further set-up and download through a KNXnet/IP tunnelling connection.

9) If the SEARCH_RESPONSE answers reveal that KNXnet/IP devices use different IP address spaces, the tool software shall present an error message.

10) If the SEARCH_RESPONSE answers reveal that the IP address of a KNXnet/IP device is different from the network settings of the tool software and the IP address of the KNXnet/IP device is an AutoIP address, the tool software shall present an error message and user control button that enables to reset the KNXnet/IP device via KNXnet/IP Routing with the KNX connectionless restart service.

5.2.6.5.2 Configuration Procedure for configuration via KNXnet/IP Routing

The tool software is connected to a KNX Subnetwork (1.1) via a KNX Serial Interface device (RS232) or KNX USB interface. A second KNX Subnetwork (1.2) is connected with the first via two KNXnet/IP routers and a LAN. All devices including the routers are initially unconfigured.

The tool software uses KNX management procedures and (indirectly) KNXnet/IP Routing to configure the two KNXnet/IP routers.

This is the configuration procedure for assignment of the KNX Individual Address and configuration of parameters of a KNXnet/IP device:

1) The tool software requests the user to activate the programming mode of the KNXnet/IP device.

NOTE This can be done by pressing the Programming Button on the KNXnet/IP device.

2) The tool software sends a KNX read (broadcast). The KNXnet/IP router on this KNX Subnetwork sends this broadcast to other KNXnet/IP routers which in turn forward it to their KNX Subnetworks and process the telegram themselves if the programming mode is active.

3) The tool software shall check if more than one device answers with "programming mode active" and if so shall abort procedure with a descriptive message about the reason for aborting the procedure. The KNXnet/IP router on this KNX Subnetwork forwards the answer(s) to this KNX Subnetwork.

4) The tool software shall send a KNX write (broadcast) to write the KNX Individual Address into the KNXnet/IP device. The KNXnet/IP router on this KNX Subnetwork sends this broadcast to other KNXnet/IP routers, which in turn forward it to their KNX Subnetworks and process the telegram themselves if the programming mode is active.

5) The tool software shall establish a KNX connection to the KNXnet/IP device to download the configuration parameters (properties). This requires that the KNXnet/IP router existing on the same KNX Subnetwork as the tool software shall be configured first before other KNXnet/IP devices can be configured.

6) After downloading the parameters the tool software shall reset the KNXnet/IP device for the parameters to become effective.

5.2.6.5.3 Configuration procedure for configuration via KNXnet/IP Device Management

The tool software is directly connected to a LAN. Two KNX Subnetworks, (1.1) and (1.2), are connected to the LAN via two KNXnet/IP routers. All devices including the routers are initially unconfigured.

The tool software shall use KNXnet/IP Device Management to configure the two KNXnet/IP routers.

This is the configuration procedure for assignment of the KNX Individual Address and configuration of parameters of a KNXnet/IP device:

1) The tool software shall request the user to activate the programming mode of the KNXnet/IP device.

NOTE This can be done by pressing the Programming Button on the KNXnet/IP device.

2) The tool software shall send a KNXnet/IP SEARCH_REQUEST frame.

3) The tool software shall check if more than one KNXnet/IP device (service container) answers with device status “programming mode active” and if so abort procedure.

4) The tool software shall use the IP address from the SEARCH_RESPONSE frame of the KNXnet/IP device to establish a device management connection to the device.

5) The tool software shall set the KNX Individual Address, project identifier and subnetwork information if applicable.

6) At this point, the tool software can download additional parameters if necessary.

7) After successfully disconnecting the device, management connection or sending a reset_req service to the device, the changed values shall be written and the device shall be restarted.

5.2.7 Data Packet structures

5.2.7.1 Introduction

All KNXnet/IP frames shall have a common header, consisting of header length information, the protocol version, the KNXnet/IP service type identifier, and the total length of the KNXnet/IP frame.

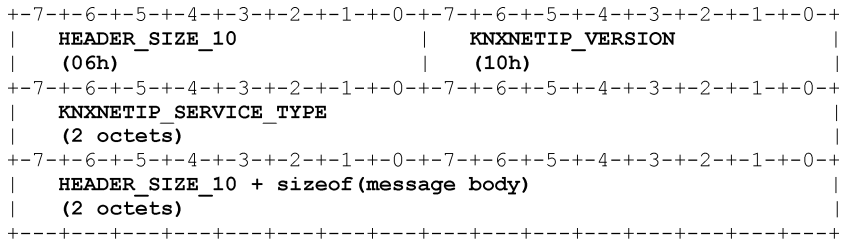


Figure 9 — KNXnet/IP message header

Requests sent to connectionless KNXnet/IP endpoints shall include information about the return address. This HPAI for the reception of the response information is always the first data of the KNXnet/IP body of all these requests. Additional KNXnet/IP service related data may follow. Response packets do not contain this kind of return address information.

5.2.7.2 Common constants

5.2.7.2.1 HEADER_SIZE_10

This constant with value 06h shall identify the KNXnet/IP header as defined in protocol version 1.0.

5.2.7.2.2 KNXNETIP_VERSION_10

This constant with value 10h shall identify the KNXnet/IP protocol version 1.0.

5.2.7.3 Common error codes

5.2.7.3.1 E_NO_ERROR

This constant with value 00h shall identify a successful operation.

5.2.7.3.2 E_HOST_PROTOCOL_TYPE

This constant with value 01h shall identify that the requested host protocol is not supported by the KNXnet/IP device.

5.2.7.3.3 E_VERSION_NOT_SUPPORTED

This constant with value 02h shall identify that the requested protocol version is not supported by the KNXnet/IP device.

5.2.7.3.4 E_SEQUENCE_NUMBER

This constant with value 04h shall identify that the received sequence number is out of order.

5.2.7.4 KNXnet/IP services

5.2.7.4.1 Core KNXnet/IP services

5.2.7.4.1.1 SEARCH_REQUEST

This constant with value 0201h shall identify the KNXnet/IP service type sent by KNXnet/IP client to search available KNXnet/IP servers.

5.2.7.4.1.2 SEARCH_RESPONSE

This constant with value 0202h shall identify the KNXnet/IP service type sent by KNXnet/IP server when responding to a KNXnet/IP SEARCH_REQUEST.

5.2.7.4.1.3 DESCRIPTION_REQUEST

This constant with value 0203h shall identify the KNXnet/IP service type sent by KNXnet/IP client to a KNXnet/IP server to retrieve information about capabilities and supported services.

5.2.7.4.1.4 DESCRIPTION_RESPONSE

This constant with value 0204h shall identify the KNXnet/IP service type sent by KNXnet/IP server in response to a DESCRIPTION_REQUEST to provide information about the server implementation sent.

5.2.7.4.1.5 CONNECT_REQUEST

This constant with value 0205h shall identify the KNXnet/IP service type sent by KNXnet/IP client to establish a communication channel with a KNXnet/IP server.

5.2.7.4.1.6 CONNECT_RESPONSE

This constant with value 0206h shall identify the KNXnet/IP service type sent by KNXnet/IP server in response to a CONNECT_REQUEST telegram.

5.2.7.4.1.7 CONNECTIONSTATE_REQUEST

This constant with value 0207h shall identify the KNXnet/IP service type sent by KNXnet/IP client requesting the connection state of an established connection with a KNXnet/IP server.

5.2.7.4.1.8 CONNECTIONSTATE_RESPONSE

This constant with value 0208h shall identify the KNXnet/IP service type sent by KNXnet/IP server when receiving a CONNECTIONSTATE_REQUEST for an established connection.

5.2.7.4.1.9 DISCONNECT_REQUEST

This constant with value 0209h shall identify the KNXnet/IP service type sent by KNXnet/IP device, typically the client, to terminate an established connection.

5.2.7.4.1.10 DISCONNECT_RESPONSE

This constant with value 020Ah shall identify the KNXnet/IP service type sent by KNXnet/IP device, typically the server, in response to a DISCONNECT_REQUEST.

5.2.7.5 Placeholders

5.2.7.5.1 Host Protocol Address Information (HPAI)

The Host Protocol Address Information structure (HPAI) shall be the address information required to uniquely identify a communication channel on the host protocol. Its size shall vary between different host protocols. For the specific definition of the HPAI, consult the host protocol dependent addendums of the KNXnet/IPstandard.

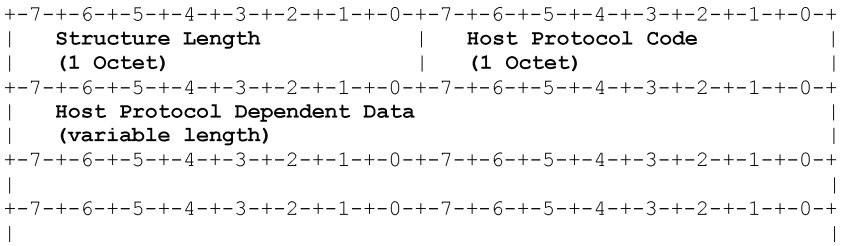


Figure 10 — HPAI structure binary format

5.2.7.5.2 Connection Request Information (CRI)

The Connection Request Information structure (CRI) shall be the additional information needed for different types of communication channels to fulfil a connection request. As this structure shall contain two substructures including host protocol independent data as well as host protocol dependent information, the specific definition of the CRI can be found in the description of the connection type with consultancy of the host protocol dependent parts of the KNXnet/IPstandard.

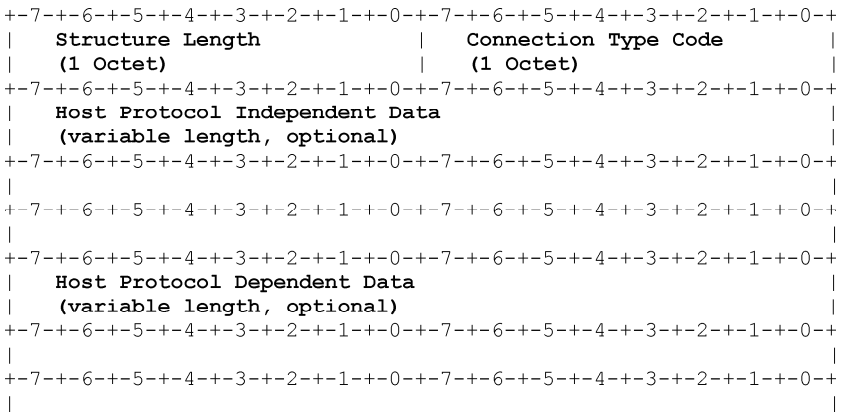


Figure 11 — CRI structure binary format

5.2.7.5.3 Connection Response Data Block (CRD)

The Connection Request Data Block structure (CRD) shall be the data block returned with the CONNECT_RESPONSE frame. As this structure shall contain two substructures including host protocol independent data as well as host protocol dependent information, the specific definition of the CRD can be found in the description of the connection type with consultancy of the host protocol dependent parts of the KNXnet/IP standard.

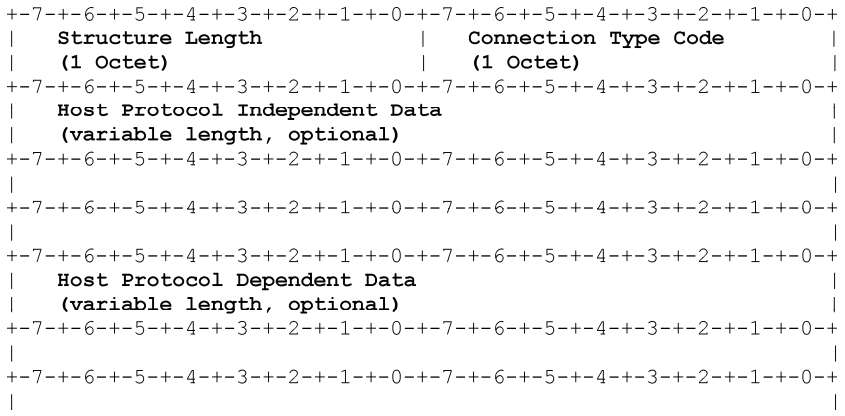


Figure 12 — CRD structure binary format

5.2.7.5.4 Description Information Block (DIB)

5.2.7.5.4.1 Use, format and general requirements

The Description Information Block structure (DIB) shall be used by a KNXnet/IP server to return a specific block of device information when responding to a DESCRIPTION_REQUEST.

At least two DIB structures shall be returned with information about the device capabilities on (1) device hardware and (2) supported service families.

More than two DIB structures may be returned in one DESCRIPTION_RESPONSE frame.

The first octet of each DIB shall contain the length of the DIB structure. The second octet shall declare the DIB structure type. Then the actual data of the DIB shall be appended. The structure shall always have an even number of octets which may have to be achieved by padding with 00h in the last octet of the DIB structure.

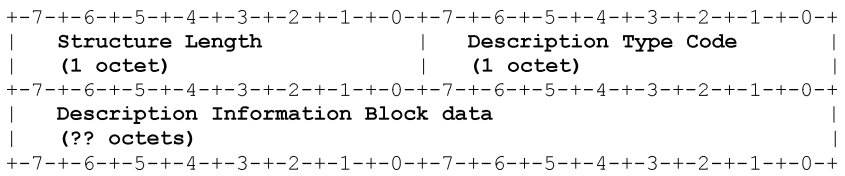


Figure 13 — Description structure binary format

Table 4 lists the valid description type codes.

Table 4 — Description type codes

Description type	Value	V.	Description
DEVICE_INFO	01h	1	Device information e.g. KNX medium
SUPP_SVC_FAMILIES	02h	1	Service families supported by the device
IP_CONFIG	03h	1	IP configuration of the device
IP_CUR_CONFIG	04h	1	Current IP configuration of the device
IP_CONFIG	05h	1	KNX addresses used by/assigned to the device
Reserved	06h – FDh	1	Reserved for future use
MFR_DATA	FEh	1	DIB structure for further data defined by device manufacturer
not used	FFh	1	Not used

5.2.7.5.4.2 Device information DIB

The device information DIB shall have the structure as given below.

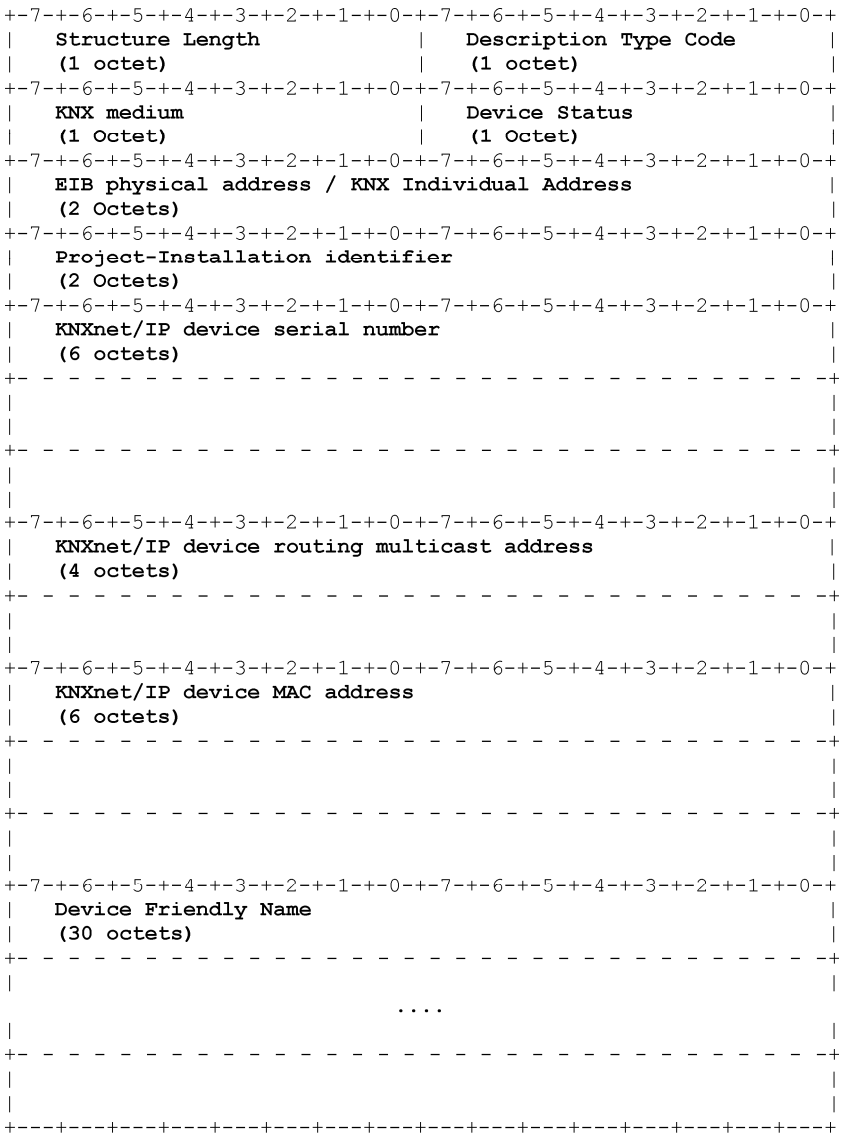


Figure 14 — Structure of device information DIB

— **KNX medium**

The KNX medium codes shall be as specified in Table 5 below.

Table 5 — KNX medium codes

KNX medium code	KNX medium
01h	reserved
02h	TP1
04h	PL110
08h	reserved
10h	RF
20h	KNX IP

Exactly one single bit shall be set.

— **Device Status**

The Device Status octet shall be as specified as:

Table 6

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
reserved	reserved	reserved	reserved	reserved	reserved	reserved	programming mode

— **Project-installation Identifier**

The Project-installation Identifier shall be defined as:

Table 7

bit 15 – 4	bit 3 – 0
Project number	Installation number

The Project-installation Identifier shall solely be assigned by ETS and shall be used to uniquely identify KNXnet/IP devices in a project with more than one KNX installation, i.e. more than 15 areas with 12 lines, or in an environment with more than one KNX project.

— **KNXnet/IP device KNX Serial Number**

The KNXnet/IP device serial number shall be the KNX serial number of the KNXnet/IP device. This information may be used to identify the device or set the Individual Address.

— **KNXnet/IP device routing multicast address**

The KNXnet/IP device routing multicast address shall be the multicast address used by a KNXnet/IP router for KNXnet/IP Routing. KNXnet/IP devices that do not implement KNXnet/IP Routing shall set this value to 0.0.0.0. KNX serial number of the KNXnet/IP device. This information may be used if KNXnet/IP Routing messages need to be sent to KNXnet/IP routers that do not use the default KNXnet/IP routing multicast address, which shall be equal to the KNXnet/IP system setup multicast address.

— **KNXnet/IP device MAC address**

The KNXnet/IP device MAC address is the Ethernet MAC address of the KNXnet/IP device. This information may be used to identify the device on the Ethernet to a server allocating network resources, specifically the unicast IP address for the KNXnet/IP device.

— **Device Friendly Name**

The device friendly name may be any NULL (00h) terminated ISO/IEC 8859-1 character string with a maximum length of 30 octets. This name may be used to identify the device to a user. Unused octets are filled with the NULL (00h) character.

5.2.7.5.4.3 Supported service families DIB

The supported service families DIB shall have this structure:

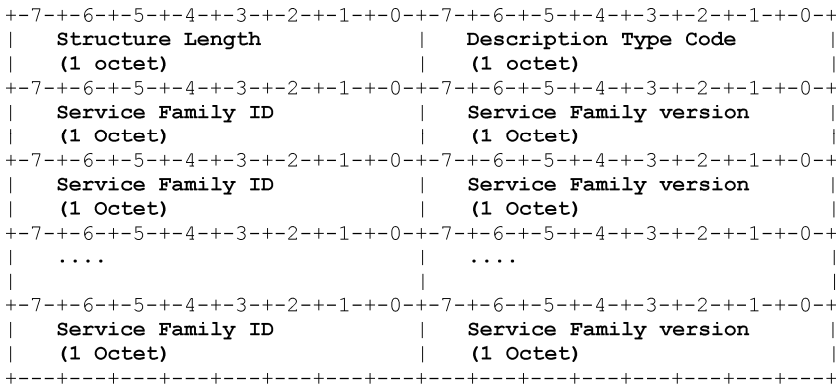


Figure 15 — Supported services families DIB

The service family ID's shall be the high octet of the service type ID. A list of service type ID's can be found in Annex A, Table A.2.

The version of a service family shall refer to the version of the corresponding KNXnet/IP chapter document. This version is only updated when the document itself is updated after it has gone through the KNX process. Any version of a service family shall be backwards compatible with previous versions, i.e. all services shall be implemented and supported. The service family version is an integer. It does not represent the manufacturer's implementation version.

5.2.7.5.4.4 Manufacturer data DIB

The manufacturer data DIB has this structure:

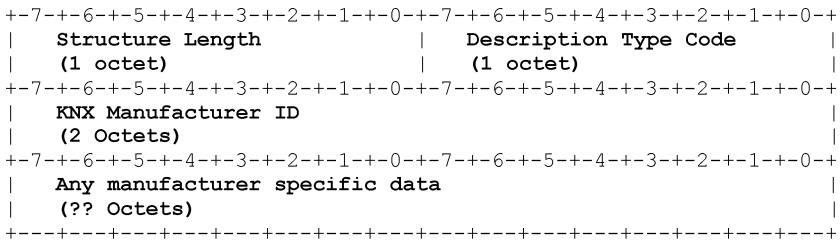


Figure 16 — Manufacturer data DIB

The KNX manufacturer ID shall be added to clearly identify the manufacturer. This information is not necessarily encoded in the KNXnet/IP serial number (6 octets).

The manufacturer data DIB may contain any manufacturer specific data.

5.2.7.6 Discovery

5.2.7.6.1 SEARCH_REQUEST

The SEARCH_REQUEST frame shall be sent by a KNXnet/IP client via multicast to the discovery endpoints of any listening KNXnet/IP server. As communication with the discovery endpoint shall be connection- and stateless, the client's discovery endpoint address information shall be included in the KNXnet/IP body.

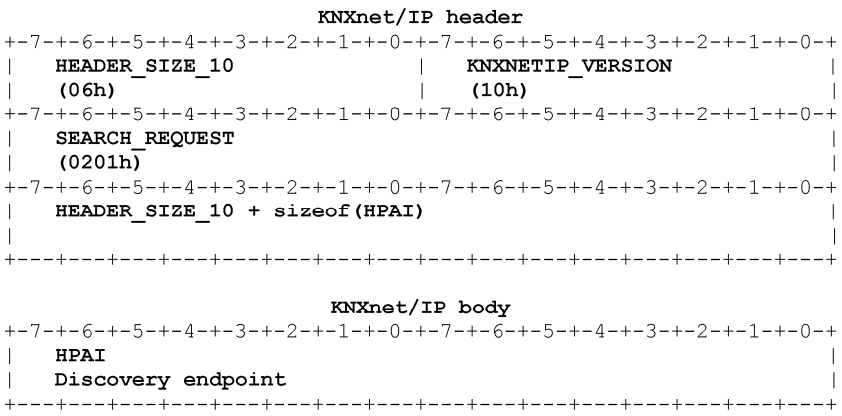


Figure 17 — SEARCH_REQUEST frame binary format

5.2.7.6.2 SEARCH_RESPONSE

The SEARCH_RESPONSE frame shall be sent by the KNXnet/IP server as an answer to a received SEARCH_REQUEST frame. It shall be addressed to the client's discovery endpoint using the HPAI included in the received frame.

The HPAI of the server's own control endpoint shall be carried in the KNXnet/IP body of the SEARCH_RESPONSE frame along with the description of the device hardware and the supported service families. If the KNXnet/IP server supports more than one KNX connection, the server shall announce each of its own control endpoints in a single SEARCH_RESPONSE frame.

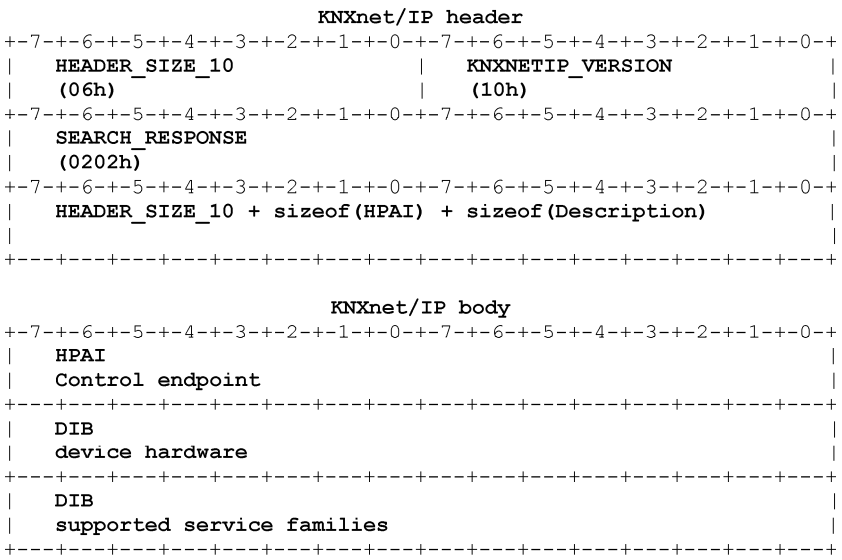


Figure 18 — SEARCH_RESPONSE frame binary format

5.2.7.7 Self-description

5.2.7.7.1 DESCRIPTION_REQUEST

The DESCRIPTION_REQUEST frame shall be sent by the KNXnet/IP client to the control endpoint of the KNXnet/IP server to obtain a self-description of the KNXnet/IP server device.

The KNXnet/IP body shall contain the return address information of the client's control endpoint.

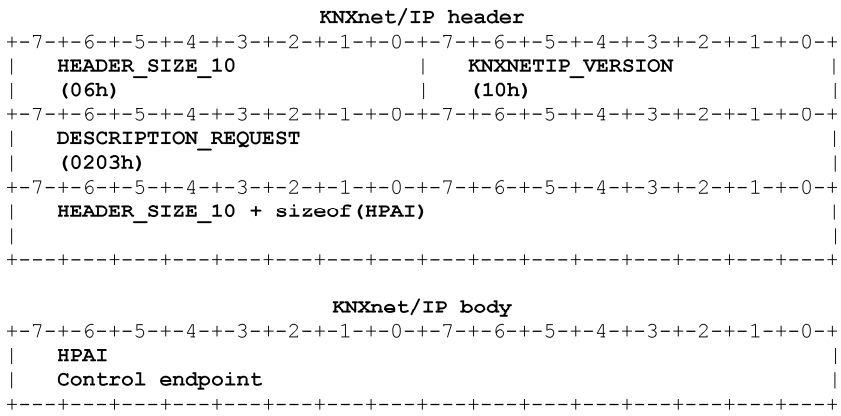


Figure 19 — DESCRIPTION_REQUEST frame binary format

The DESCRIPTION_RESPONSE frame shall be sent by the KNXnet/IP server as an answer to a received DESCRIPTION_REQUEST frame. It shall be addressed to the client's control endpoint using the HPAI included in the received frame.

The size of the KNXnet/IP body varies depending on the number of DIB structures sent by the server in response to the client's DESCRIPTION_REQUEST.

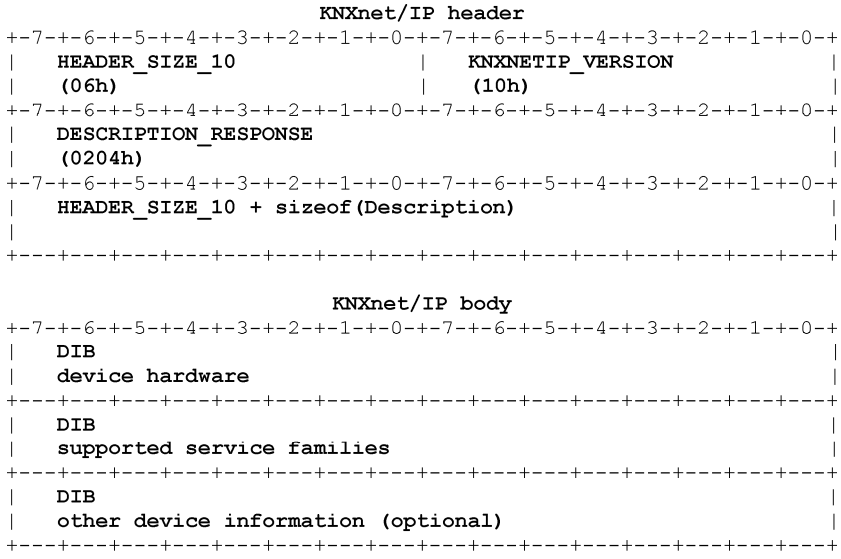


Figure 20 — DESCRIPTION_RESPONSE frame binary format

5.2.7.8 Connection management

5.2.7.8.1 CONNECT_REQUEST

The CONNECT_REQUEST frame shall be sent by the KNXnet/IP client to the control endpoint of the KNXnet/IP server. As for every request using control communication, the KNXnet/IP body shall begin with the return address information of the client's control endpoint.

Next follows the CRI, a variable data structure that shall include all additional information that is specific to the requested connection type (and to the underlying host protocol). The exact definition of this structure can be found in the description of the specific connection type.

Table 8 — Connection types

Connection type	Value	V.	Description
DEVICE_MGM_CONNECTION	03h	1	Data connection used to configure a KNXnet/IP device
TUNNEL_CONNECTION	04h	1	Data connection used to forward EIB telegrams between two KNXnet/IP devices
REMLOG_CONNECTION	06h	1	Data connection used for configuration and data transfer with a remote logging server
REMCONF_CONNECTION	07h	1	Data connection used for data transfer with a remote configuration server
OBJSVR_CONNECTION	08h	1	Data connection used for configuration and data transfer with an Object Server in a KNXnet/IP device

Inside the CRI one octet shall determine the type of communication channel requested by this frame and two octets are reserved for the options for this channel. Additional KNXnet/IP documents may define more connection types and additional connection type specific connection options.

The host protocol address information of the client's data endpoint meant for the requested data connection shall complete the body of the CONNECT_REQUEST frame.

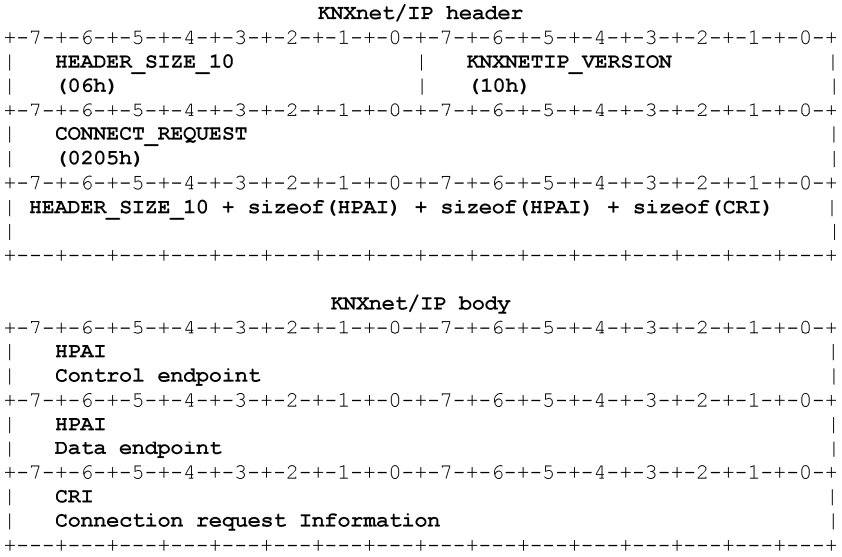


Figure 21 — CONNECT_REQUEST frame binary format

5.2.7.8.2 CONNECT_RESPONSE

The CONNECT_RESPONSE frame shall be sent by the KNXnet/IP server as an answer to a received CONNECT_REQUEST frame. It shall be addressed to the client's control endpoint using the HPAI included in the received frame.

The size of the KNXnet/IP body varies according to the success or failure of the client's CONNECT_REQUEST.

If the connection request could be successfully fulfilled with all the requested options, the body shall contain a communication channel ID that uniquely identifies this connection with the KNXnet/IP server. The communication channel ID shall be the first octet of the body.

The second octet of the body shall contain the status information of the connection request. This status information can contain error information regarding the request itself or regarding the connection type specific information.

Table 9 — Common CONNECT_RESPONSE status codes

Error constant	Value	V.	Description
E_NO_ERROR	00h	1	The connection was established successfully
E_CONNECTION_TYPE	22h	1	The requested connection type is not supported by the KNXnet/IP server device
E_CONNECTION_OPTION	23h	1	One or more requested connection options are not supported by the KNXnet/IP server device
E_NO_MORE_CONNECTIONS	24h	1	The KNXnet/IP server device could not accept the new data connection because its maximum amount of concurrent connections are already busy

The Host Protocol Address Information of the server's data endpoint prepared for this data connection shall be the next block of data in the body of a successful CONNECT_RESPONSE frame.

The Connection Response Data Block containing connection type specific response data shall complete the body of a successful CONNECT_RESPONSE frame.

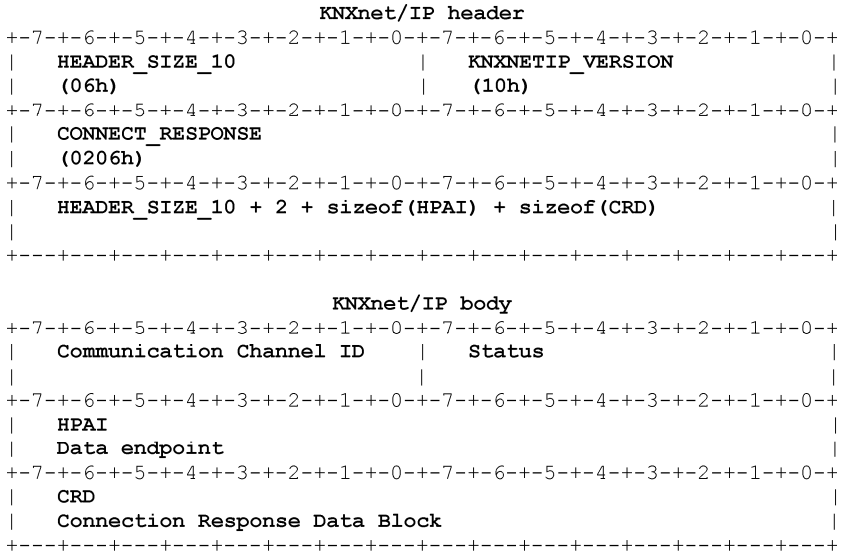


Figure 22 — CONNECT_RESPONSE frame binary format

5.2.7.8.3 CONNECTIONSTATE_REQUEST

The CONNECTIONSTATE_REQUEST frame shall be sent by the KNXnet/IP client to the control endpoint of the KNXnet/IP server. The first octet of the KNXnet/IP body shall contain the communication channel ID that the KNXnet/IP server uses to uniquely identify the data connection for this connection state request. The second octet shall be reserved for future use.

The HPAI with the return address information of the client's control endpoint shall be added after the communication channel ID.

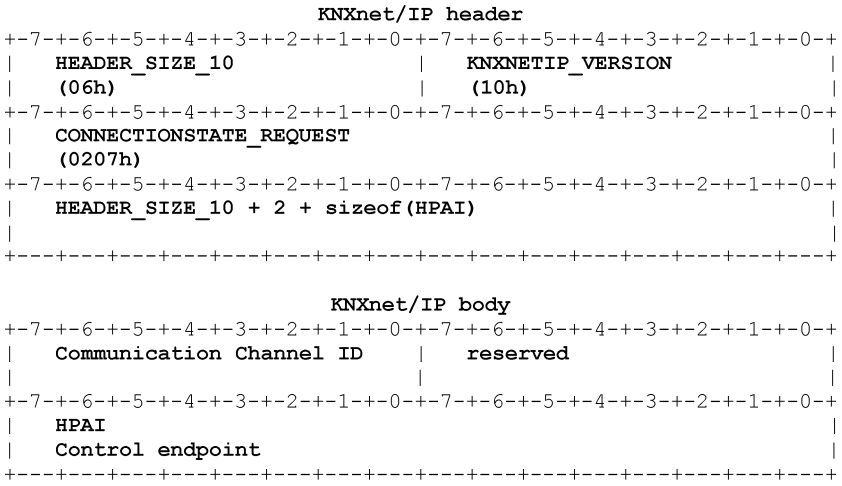


Figure 23 — CONNECTIONSTATE_REQUEST frame binary format

5.2.7.8.4 CONNECTIONSTATE_RESPONSE

The CONNECTIONSTATE_RESPONSE frame shall be sent by the KNXnet/IP server as an answer to a received CONNECTIONSTATE_REQUEST frame. It shall be addressed to the client's control endpoint using the HPAI included in the received frame.

The first octet of the KNXnet/IP body shall contain the communication channel ID that the KNXnet/IP client passed to the KNXnet/IP with the CONNECTIONSTATE_REQUEST frame.

The second octet of the KNXnet/IP body shall contain the status information of the connection state request. The following table lists the status codes that are defined for the CONNECTIONSTATE_RESPONSE frame:

Table 10 — CONNECTIONSTATE_RESPONSE status codes

Error constant	Value	V.	Description
E_NO_ERROR	00h	1	The connection state is normal
E_CONNECTION_ID	21h	1	The KNXnet/IP server device could not find an active data connection with the specified ID
E_DATA_CONNECTION	26h	1	The KNXnet/IP server device detected an error concerning the data connection with the specified ID
E_KNX_CONNECTION	27h	1	The KNXnet/IP server device detected an error concerning the EIB bus/KNX subsystem connection with the specified ID

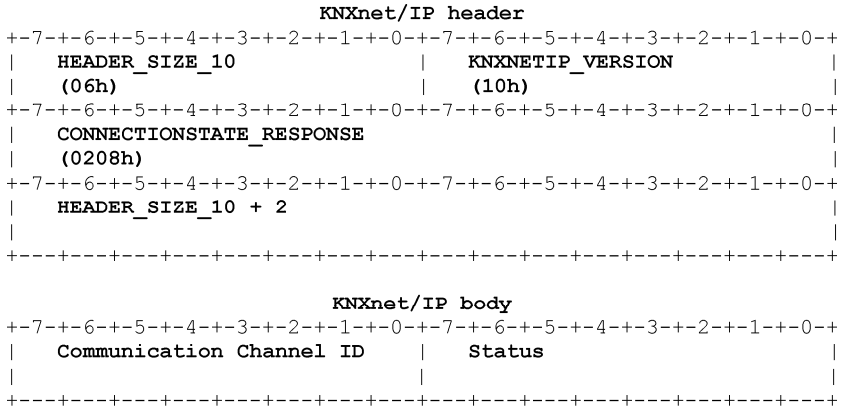


Figure 24 — CONNECTIONSTATE_RESPONSE frame binary format

5.2.7.8.5 DISCONNECT_REQUEST

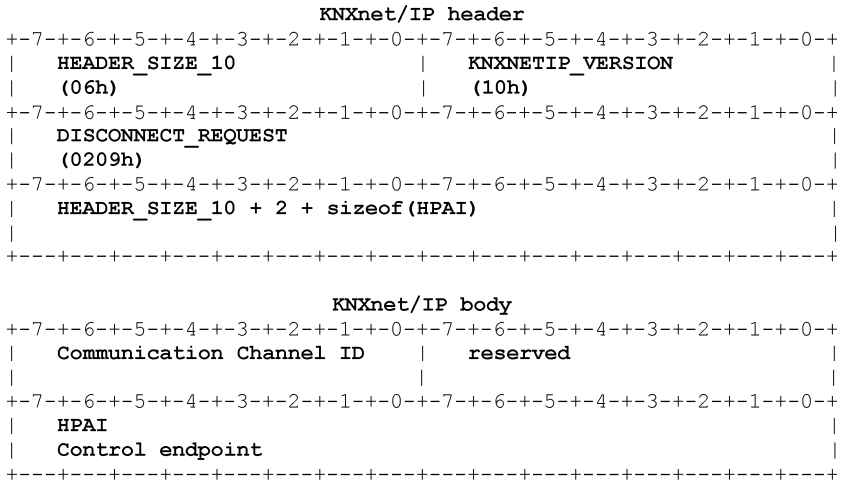


Figure 25 — DISCONNECT_REQUEST frame binary format

5.2.7.8.6 DISCONNECT_RESPONSE

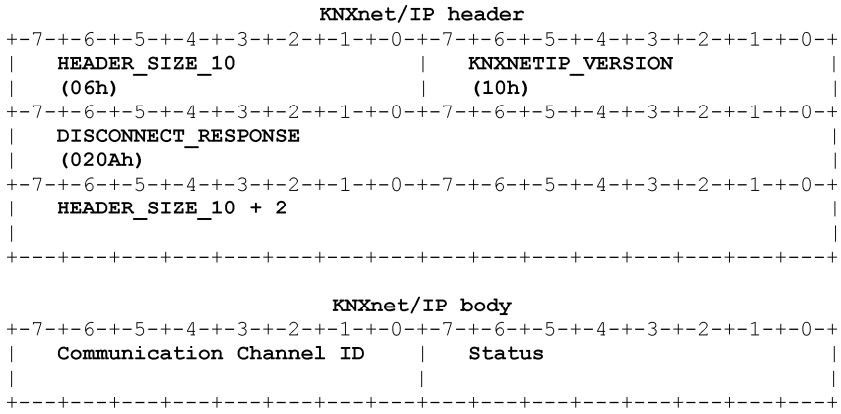


Figure 26 — DISCONNECT_RESPONSE frame binary format

5.2.8 IP Networks

5.2.8.1 Introduction

The KNXnet/IP protocol is used to tunnel or route KNX data over the widely spread Internet Protocol (IP), enabling remote access and maintenance across long distances, as well as usage as high-speed backbone for KNX networks.

This part of the standard defines which IP parameters and features are supported by KNXnet/IP.

It is assumed that the reader is familiar with the Internet Protocols TCP and UDP.

5.2.8.2 Physical vs. logical network

IP networks are not like KNX networks: KNX networks are physical busses by nature. This implies that all devices on the channel will by default receive all packets transmitted on the network. In addition, when a new device is added to the network it is not necessary that other devices on the network become aware of it before they can exchange packets. To transmit a telegram over KNX, it is only necessary that a device be capable of physically transmitting the telegram on the bus, nothing more. If a device is simply physically connected to a KNX network, it is capable of exchanging telegrams with other devices on the channel.

By contrast, an IP network is not physical, but logical in nature. There are a number of different physical media that can support IP communications and any of them should be capable of supporting tunnelling KNXnet/IP frames. Because it is dealt with a logical channel, it is necessary to “construct” the channel by informing each device on the channel of the existence of the other devices on that channel. In other words before a device can transmit a packet to some other device on an IP channel it shall be made aware of how to specifically send a packet to that device, i.e. its IP address.

Another significant difference between physical and logical networks is that in the case of typical physical networks it is possible to calculate fixed upper bounds on the length of time it will take a packet to traverse from one device to another once the packet is transmitted on the channel. This is not always possible for IP networks. The deviation of packet delivery times between KNXnet/IP devices on an IP channel are much higher than those experienced with native KNX devices.

The IP channel is used as an intermediary transport mechanism for the KNX telegrams by a variety of KNXnet/IP devices. When a KNXnet/IP packet is transported on an IP channel, an IP message encapsulating the KNX telegram is sent to other KNXnet/IP devices on that IP channel. The IP channel is specified by the list of unicast IP addresses, exactly one for each KNXnet/IP device. There is no maximum to the number of KNXnet/IP devices on a single IP network.

5.2.8.3 Transport mechanisms

IP is a network level protocol. It is designed to operate over a wide range of physical media and link layer protocols. As such, this document does not specify anything about the link or physical layers of the IP stack.

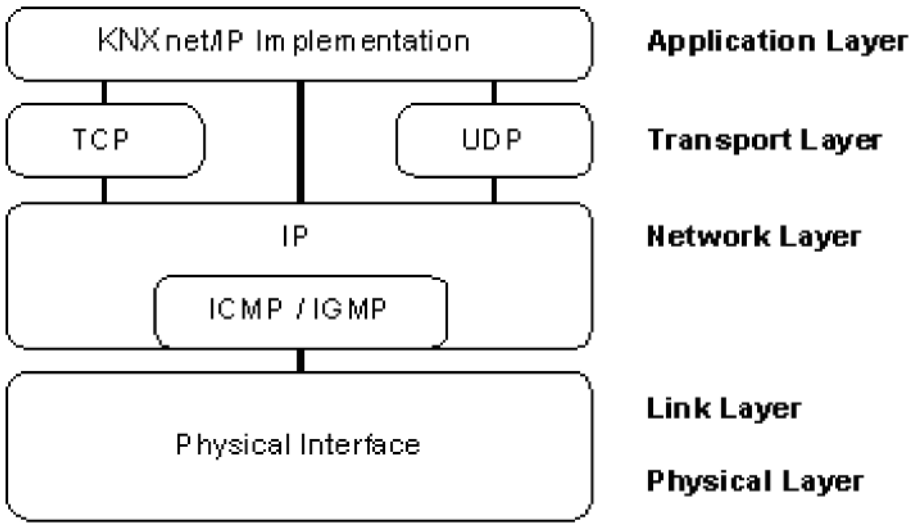


Figure 27 — IP protocol stack

Three most common mechanisms used to transport IP packets are raw IP, TCP and UDP. TCP and UDP are transport protocols built on top of IP.

TCP implements a reliable, connection-oriented end-to-end transport service. It includes provisions to guarantee the correct transmission and to preserve the ordering of the received data stream.

UDP on the other hand implements a best effort datagram service. Raw IP is a mechanism implemented by a number of operating systems to provide host applications with direct access to the IP layer, bypassing any transport service.

Although it is possible to use any of these protocols as a KNXnet/IP host protocol, from an application point of view it is much easier to simply exclude raw IP and restrict the specification to TCP and UDP.

5.2.8.4 UDP and TCP

Since the KNX protocol itself employs end-to-end acknowledgment, it is not necessary to guarantee tunnelling IP packets containing KNX data telegrams are received between the KNXnet/IP devices. Using a reliable transport service introduces unnecessary protocol overhead in this case, which makes TCP less efficient than UDP or raw IP. On the other hand, there will be configuration and status messages exchanged between KNXnet/IP devices that do not contain tunnelled KNX data and yet need to be received reliably.

TCP has the advantages of reliable delivery service and hence will guarantee that the received packet ordering is preserved. On the down side, it does not support multi-cast addressing and is less efficient than UDP. TCP also consumes more resources of the KNXnet/IP device to implement than UDP. UDP is more efficient in carrying tunnelled KNX data messages, but for the lack of a reliable delivery, service will not guarantee that the packet ordering is preserved.

Given the increased efficiencies of UDP regarding the transport of KNX data messages and its support of multi-cast addressing, it will be used as the default to communicate between KNXnet/IP devices. All KNXnet/IP devices shall support UDP. The reliability advantages of TCP may be supported in addition to UDP. TCP support in KNXnet/IP devices is OPTIONAL.

To address the sequencing issue there shall be the sequence counter option added to the connection header to help in sequencing.

Using UDP, datagrams can be sent using either unicast or multi-cast addressing. Unicast is point to point meaning that a datagram is sent from one IP host to a single other IP host. When sending the same datagram to multiple IP hosts as it is necessary for routing of KNX data, it is much more efficient to use multi-cast addressing. It is therefore recommended that KNXnet/IP devices support both unicast and multi-cast IP

addressing although it is not required that a KNXnet/IP device supports multi-casts in order to inter-operate with a KNXnet/IP device that does.

5.2.8.5 IP Address Assignment

5.2.8.5.1 IP unicast address

5.2.8.5.1.1 Fixed IP address

A fixed IP address is assigned to the device through a user interface, via ETS or some other tool. This IP address assignment is fixed.

Any KNXnet/IP device shall support fixed assigned IP addresses.

5.2.8.5.1.2 BootP / DHCP

A BootP or DHCP server is designed to automatically assign an IP address to a device. Configuration of both types of servers is part of network administration and is available on all network server platforms (Windows NT Server, Windows 2000 Server, Windows XP Server, Unix, Linux). Pre-administered DHCP servers like DSL modems or ISDN routers can also be used.

Either a BootP or a DHCP client shall be implemented on a KNXnet/IP device.

5.2.8.5.1.3 AutoIP

A device implementing AutoIP is capable of assigning itself a unicast IP address in the range of 169.254.1.0 to 169.254.254.255.

A KNXnet/IP device may implement AutoIP.

5.2.8.5.1.4 IP address assignment procedure

Table 11 describes the address assignment procedure for KNXnet/IP devices.

Table 11 — Address assignment procedure

1	IF	Fixed IP address assigned to KNXnet/IP device	THEN	→ 2
			ELSE	→ 6
2	IF	BootP or DHCP address assignment is enabled	THEN	→ 3
			ELSE	Use fixed IP address already assignet
3	IF	BootP or DHCP address assignment is successful	THEN	Use newly assigned IP address
			ELSE	→ 4
4	IF	AutoIP address assignment is enabled	THEN	→ 5
			ELSE	Use fixed IP address already assigned
5	IF	AutoIP address assignment is successful	THEN	Use newly assigned IP address
			ELSE	Use fixed IP address already assigned
6	IF	BootP or DHCP address assignment is enabled	THEN	→ 7
			ELSE	→ 8
7	IF	BootP or DHCP address assignment is successful	THEN	Use newly assigned IP address
			ELSE	→ 8
8	IF	AutoIP address assignment is enabled	THEN	→ 9
			ELSE	No IP address is assigned. Enter 0.0.0.0 in IP address field of HPAI
9	IF	AutoIP address assignment is successful	THEN	Use newly assigned IP address.
			ELSE	No IP address is assigned. Enter 0.0.0.0 in IP address field of HPAI

5.2.8.5.2 IP multicast addresses

5.2.8.5.2.1 KNXnet/IP system setup multicast address

To ensure that any KNXnet/IP device can be reached by the KNXnet/IP Core discovery services a "System Setup Multicast Address" is defined. The value of the "System Setup Multicast Address" shall be 224.0.23.12.

5.2.8.5.2.2 KNXnet/IP routing multicast address

Any KNXnet/IP device implementing KNXnet/IP routing shall have a "Routing Multicast Address". This address shall be derived from the "System Setup Multicast Address" by adding an offset. This offset has a default value of zero. If there is more than one installation in a project KNXnet/IP routers in separate installations shall be assigned to different Routing Multicast Addresses.

5.2.8.6 KNXnet/IP host protocol

5.2.8.6.1 Device specification

A KNXnet/IP device shall behave like any standard IP host capable of exchanging IP packets with any other IP host either on the same IP subnet or anywhere else in the IP network cloud. A KNXnet/IP device shall have a single unicast IP address, shall belong to at least one IP multi-cast group, and may be capable of belonging to up to 16 multi-cast groups. A KNXnet/IP device shall support multi-casting.

A KNXnet/IP device shall support these IP protocols: ARP, ICMP, IGMP, BootP/DHCP[4] and UDP. Additionally it may support other IP protocols like RARP, TCP, NTP, FTP, HTTP, SMTP, DNS or SNMP.

In order to initiate communication via TCP/UDP to a KNXnet/IP device it is necessary to define a fixed port number for the discovery procedure in addition to the individual IP address.

To support the routing of IP packets between subnets or through the Internet KNXnet/IP devices shall be compatible with whatever standard mechanisms (IP routers, switches, etc.) are required to perform the IP routing functions.

5.2.8.6.2 Host Protocol Address Information

The Host Protocol Address Information shall contain the information that is necessary to uniquely identify an Internet Protocol transport connection endpoint. This shall include the network layer address and the transport layer identifier called Port number. Both, IP address and port number, are stored binary in network octet order.

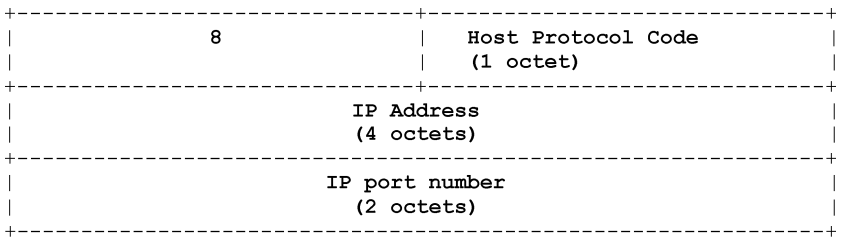


Figure 28 — IP Host Protocol Address Information binary format

Table 12 — Host protocol codes for IP network

Constant name	Value	V.	Description
IPV4_UDP	01h	1	Identifies an Internet Protocol version 4 address and port number for UDP communication
IPV4_TCP	02h	1	Identifies an Internet Protocol version 4 address and port number for TCP communication

5.2.8.6.3 KNXnet/IP Endpoints

5.2.8.6.3.1 General

The use of the KNXnet/IP endpoints as a logical view of the communication between KNXnet/IP devices results in a big flexibility of the actual implementation using the specific host protocol.

As mentioned above KNXnet/IP devices can support UDP or optionally TCP as the transport layer protocol for IP communication. These IP channels can dynamically be negotiated between the KNXnet/IP devices using the HPAI structure of the KNXnet/IP frames. Because of the point-to-multipoint requirement of the discovery communication, UDP at port 3671 is the only allowed transport mechanism.

The KNXnet/IP port number 3671 shall be used for the discovery end point. It may be used for the data and control end points too. This is not mandatory. The KNXnet/IP server port number returned in response to a CONNECT_REQUEST may be any valid port number.

4) BootP/DHCP: It is essential that either one is implemented.

It is therefore possible to implement a KNXnet/IP device that uses only one bi-directional UDP port for all communication or one could implement a KNXnet/IP device that uses two unidirectional UDP ports for the discovery endpoint and for each control endpoint and TCP ports for data connections.

5.2.8.6.3.2 Discovery Endpoint

Only UDP is allowed for discovery endpoint communication. An attempt to request TCP communication will result in a host protocol type error.

The KNXnet/IP client shall send a SEARCH_REQUEST frame using UDP local broadcast from any source port to the fixed discovery endpoint destination at port 3671. Any KNXnet/IP server receiving this request shall ignore the source information at IP level. Only the HPAI of the received KNXnet/IP frame is relevant. The KNXnet/IP servers shall then send their SEARCH_RESPONSE frame(s) using UDP unicast from any source port to the requested destination.

5.2.8.6.3.3 Control Endpoint

UDP and TCP are allowed for control endpoint communication. Every KNXnet/IP device shall support UDP communication. TCP communication is OPTIONAL.

The KNXnet/IP client shall receive the port information about the KNXnet/IP server's control endpoint from the HPAI of the SEARCH_RESPONSE frames. A KNXnet/IPserver shall not change this port once it is announced.

The KNXnet/IP server shall receive the complete address information about the KNXnet/IP client's control endpoint with every new control request. Although possible, it is recommended that the client does not change this port either.

5.2.8.6.3.4 Data endpoints

UDP and TCP are allowed for data endpoint communication. Every KNXnet/IP device shall support UDP communication. TCP communication is OPTIONAL. Data endpoints can be connection oriented for point-to-point communication or connectionless for point-to-multipoint communication as it is necessary for KNXnet/IP routing. All connectionless data endpoints use special Host Protocol Address Information structures for KNXnet/IP multicasts. These IP addresses and port numbers are fixed and defined in the corresponding KNXnet/IP service protocol description.

5.2.8.6.3.5 Network Address Translation (NAT)

If KNXnet/IP communication has to traverse across network routers using Network Address Translation (NAT) the KNXnet/IP client shall set the value of the IP address and/or the port number in the HPAI to zero to indicate NAT traversal to the receiving KNXnet/IP server.

For the IP address and port number in the datagrams sent from the KNXnet/IP server to the KNXnet/IP client, the KNXnet/IP server shall replace the zero value for the IP address and/or the port number in the HPAI by the corresponding IP address and/or port number in the IP package received and use this value as the target IP address or port number for the response to the KNXnet/IP client.

Typically the KNXnet/IP client should set both the IP address and the port number to zero but may choose to set only one (IP address or port number) to zero if required by the application and possible in the given network configuration.

5.2.9 Certification

5.2.9.1 Introduction

This chapter provides information on the test procedures and requirements of the certification process.

5.2.9.2 Supported services

The supported services for KNXnet/IP and the Internet Protocol (IP) are listed in Tables 13 and 14. IPv6 will be specified in the future and is listed for completeness only.

Table 13

Service name	Sent from… to…	Implementation is
SEARCH_REQUEST	Client → Server	M
SEARCH_RESPONSE	Server → Client	M
DESCRIPTION_REQUEST	Client → Server	M
DESCRIPTION_RESPONSE	Server → Client	M
CONNECT_REQUEST	Client → Server	M
CONNECT_RESPONSE	Server → Client	M
CONNECTIONSTATE_REQUEST	Client → Server	M
CONNECTONSTATE_RESPONSE	Server → Client	M
DISCONNECT_REQUEST	Client → Server Server → Client	M
DISCONNECT_RESPONSE	Client → Server Server → Client	M

Table 14

Service name	Client	Server
IPV4_UDP	M	M
IPV4_TCP	O	O
IPV6_UDP	to be defined	to be defined
IPV6_TCP	to be defined	to be defined

5.3 Clause 3: Device Management Specification

5.3.1 Scope

This clause “Device Management” of the KNXnet/IP standard describes point-to-point exchange of IP datagrams over an IP network between a KNXnet/IP device acting as a server and a KNXnet/IP client for remote configuration and management of the KNXnet/IP device.

5.3.2 KNXnet/IP Device Management

5.3.2.1 Introduction

KNX devices connected to an IP network through KNXnet/IP devices are configured by ETS as described in 5.4.

KNXnet/IP Device Management defines management of KNXnet/IP devices.

IP unicast addressing shall be used for direct communication with individual KNXnet/IP devices.

5.3.2.2 General remarks

As a KNXnet/IP device is connected to two different networks, two different ways of configuration are imaginable: via KNX and via IP. Every KNXnet/IP device SHOULD allow configuration by means of the KNXnet/IP device configuration protocol implementation (IP). Implementation of configuration using KNX is mandatory if the KNXnet/IP device physically connects to a KNX Subnetwork. For download, all parameter values shall be presented in “big endian” format (Motorola-like), so the most significant byte is always transferred firstly.

5.3.2.3 Configuration and Management

5.3.2.3.1 Introduction

A KNXnet/IP device shall implement configuration and management using KNX Interface Objects for application and parameter download. KNXnet/IP device configuration and management using IP shall be based on Interface Object Properties and follow the same pattern as using KNX. The KNXnet/IP protocol defined for this purpose has the advantage of supporting large data structures.

Any KNXnet/IP device shall accept configuration through KNX subnetwork data telegrams. A KNXnet/IP router shall accept configuration through KNXnet/IP Routing services if those services carry cEMI data telegrams addressed to the KNX address of the KNXnet/IP router.

5.3.2.3.2 KNXnet/IP endpoints and service containers

A KNXnet/IP server shall implement one service container for each KNX Subnetwork connected to the server. While it shall implement at least one service container, it may implement more than one. If the KNXnet/IP server supports more than one KNX subnetwork connection, it is required that every KNX subnetwork is represented by a different control endpoint.

A KNXnet/IP client shall therefore consider every service container represented by a control endpoint as one independent entity no matter whether they are implemented in only one or two separate KNXnet/IP servers. In addition, multiple service containers in a single KNXnet/IP server shall behave exactly like multiple service containers in multiple KNXnet/IP servers (e.g. regarding IP traffic).

The control endpoints exposed by KNXnet/IP servers can be discovered according to the discovery procedure described in 5.2.

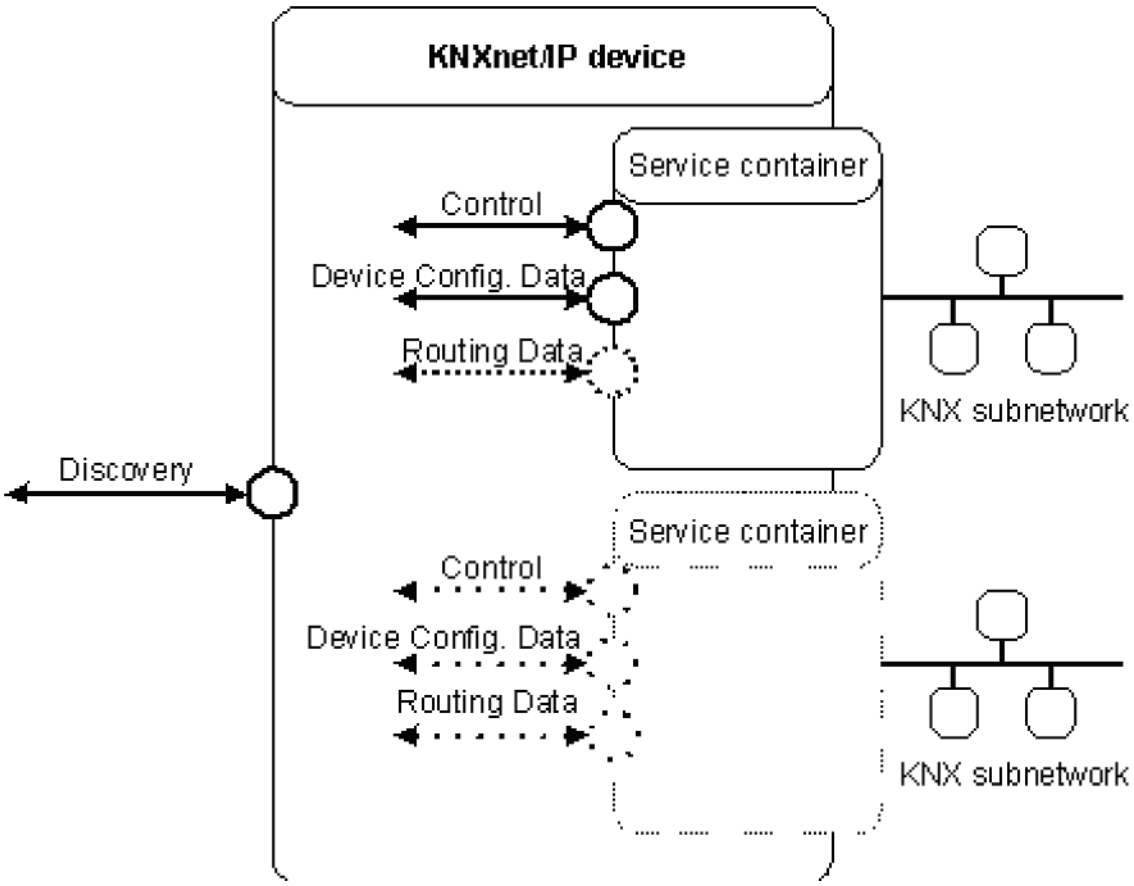


Figure 29 — KNXnet/IP device endpoints

A KNXnet/IP device's service container shall implement a Device Configuration and Management end point. Further service end points like Routing Data may be implemented depending on the device function.

The configuration and management end point shall allow access to device parameters, filter table etc. after a connection is established through the control endpoint. It shall be used by the device to exchange configuration information with ETS. Every KNXnet/IP device shall use the "KNXnet/IP Device HPAI" (unicast IP address and port number) for the Device Configuration and management data endpoint.

To access configuration and management, a KNXnet/IP client shall connect to the respective data endpoint according to 5.2 "Clause 2: Core". All communication traffic following that connection shall be handled by DEVICE_CONFIGURATION_REQUEST and DEVICE_CONFIGURATION_ACK messages.

If the KNXnet/IP client does not receive a DEVICE_CONFIGURATION_ACK within the DEVICE_CONFIGURATION_REQUEST_TIMEOUT (= 10 s) or the status of a received DEVICE_CONFIGURATION_ACK message signalled any kind of error condition, the client shall repeat the DEVICE_CONFIGURATION_REQUEST three times and then terminate the connection by sending a DISCONNECT_REQUEST to the server's control endpoint.

The KNXnet/IP server shall send no DEVICE_CONFIGURATION_ACK and shall discard the message if it receives a message with an unexpected sequence number.

5.3.2.4 Interface Object Types

5.3.2.4.1 General

KNXnet/IP and KNX IP devices shall be managed mainly through Properties of Interface Objects. For the KNXnet/IP Device Management specific functionality, the following Interface Object Types shall mainly be used:

— the Device Object, and

— the KNXnet/IP Parameter Object.

5.3.2.4.2 Device Object

The PID_DEVICE_DESCRIPTOR in the Device Object shall be mandatory for any KNXnet/IP device.

5.3.2.4.3 KNXnet/IP Parameter Object

The KNXnet/IP Parameter Object includes the IP parameters for the KNXnet/IP device's service container.

The following 5.3.2.5 describes the KNXnet/IP Property Identifiers used in KNXnet/IP Parameter Object.

5.3.2.5 KNXnet/IP Parameter Object

5.3.2.5.1 General

The KNXnet/IP Parameter Object shall include the IP parameters for the KNXnet/IP device's service container. Several Properties of the KNXnet/IP Parameter Object are closely related to each other.

EXAMPLE PID_IP_ADDRESS and PID_SUBNET_MASK are related to each other.

Write accesses to the Properties shall therefore always use KNX Transport Layer connections.

Following is a specification of Properties of the KNXnet/IP Parameter Object.

5.3.2.5.2 PID_PROJECT_INSTALLATION_ID (PID = 51)

— Property name: Project Installation Identification;

— Datapoint Type: None.

This property shall be set by ETS only and expresses which project and installation this KNXnet/IP device is part of.

The 16 bit value shall contain the project number in the upper 12 bits and the installation number in the lower 4 bits. The factory default value for this property shall be set to 0000h.

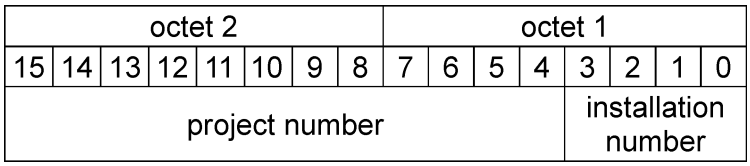


Figure 30 — PID_PROJECT_INSTALLATION_ID

ETS shall assign the proper value based on the ETS project number and the number of installations in a project.

The project number shall always be in the range [1 … 4095].

If a project has only one installation, the installation number of that installation shall be set to zero (0).

If a project has more than one installation then the installation number shall be set to the number of the installation in the range [1 … 15].

5.3.2.5.3 PID_KNX_INDIVIDUAL_ADDRESS (PID = 52)

— Property name: KNX Individual Address;

— Datapoint type: None.

This property shall be implemented by any KNXnet/IP device.

This property shall contain the KNX Individual Address of the KNXnet/IP device. The device shall ensure that the value of this property is coordinated with the combined value of the corresponding Device Object properties.

5.3.2.5.4 PID_ADDITIONAL_INDIVIDUAL_ADDRESSES (PID = 53)

— Property name: Additional Individual Addresses;

— Datapoint Type: None.

This property shall be implemented by devices providing KNXnet/IP Routing and KNXnet/IP Tunnelling.

This property shall contain a sorted list of additional KNX Individual Addresses. The first entry in the list shall be the length of the list. Subsequent entries shall be KNX Individual Addresses sorted in ascending order.

5.3.2.5.5 PID_CURRENT_IP_ASSIGNMENT_METHOD (PID = 54)

— Property name: Current IP Assignment Method;

— Datapoint Type: None.

This Property shall capture the IP address assignment method employed to set the current IP address.

The IP assignment methods shall be identified as follows:

Table 15

Value of PID_CURRENT_IP_ASSIGNMENT_METHOD	Assignment method
1	manually
2	BootP
4	DHCP
8	AutoIP

Only one value shall be set at a time.

5.3.2.5.6 PID_IP_ASSIGNMENT_METHOD (PID = 55)

— Property name: IP Assignment Method;

— Datapoint Type: None.

This property shall show the enabled IP address assignment methods for setting the current IP address. At least one shall be enabled.

The supported assignment methods are defined as follows.

Table 16

Value of PID_CURRENT_IP_ASSIGNMENT_METHOD	Assignment method
1	manually
2	BootP
4	DHCP
8	AutoIP

This property shall determine the behaviour of the IP address assignment procedure as described in 5.2.

5.3.2.5.7 PID_IP_CAPABILITIES (PID = 56)

— Property name: IP Capabilities;

— Datapoint Type: None.

This property shall show the IP capabilities supported by the KNXnet/IP device.

Property bits 0, 1, and 2 shall show the available IP address assignment methods for setting the current IP address. Manual address assignment shall always be available.

Table 17 — Device Capabilities

Bit	Description
0	BootIP
1	DHCP
2	AutoIP
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

This property is optional but its information shall be available in the ETS database.

5.3.2.5.8 PID_CURRENT_IP_ADDRESS (PID = 57)

— Property name: Current IP Address;

— Datapoint Type: None.

This property shall contain the currently used IP v4 address (32 bit). This value shall be read only and shall be set as described in 5.2, sub-clause “IP Address Assignment”.

5.3.2.5.9 PID_CURRENT_SUBNET_MASK (PID = 58)

— Property name: Current Subnet Mask;

— Datapoint Type: None.

This property shall contain the currently used IP subnet mask (32 bit).

5.3.2.5.10 PID_CURRENT_DEFAULT_GATEWAY (PID = 59)

— Property name: Current Default Gateway;

— Datapoint Type: None.

This property shall contain the IP address of the default gateway.

5.3.2.5.11 PID_IP_ADDRESS (PID = 60)

— Property name: IP Address;

— Datapoint Type: None.

This property shall contain the configured fixed IP v4 address (32 bit) value. This property shall contain the IP address as configured by a configuration tool. It shall be used when a manual address assignment is enabled. IP Address Assignment is described in Clause 5.2 "Core", sub-clause "IP Address Assignment".

5.3.2.5.12 PID_SUBNET_MASK (PID = 61)

— Property name: Subnet Mask;

— Datapoint Type: None.

This property shall contain the configured IP subnet mask (32 bit). This property shall contain the IP subnet mask as configured by a configuration tool. It shall be used when a manual address assignment is enabled. IP Address Assignment is described in Clause 5.2 “Core”, sub-clause “IP Address Assignment”.

5.3.2.5.13 PID_DEFAULT_GATEWAY (PID = 62)

— Property name: Default Gateway;

— Datapoint Type: None.

This property shall contain the configured IP address of the default gateway. This property shall contain the IP address of the default gateway as configured by a configuration tool. It shall be used when a manual address assignment is enabled. IP Address Assignment is described in clause 5.2 “Core”, sub-clause “IP Address Assignment”.

5.3.2.5.14 PID_DHCP_BOOTP_SERVER (PID = 63)

— Property name: DHCP/BootP Server;

— Datapoint Type: None.

This property shall contain the IP address of the DHCP/BootP server the KNXnet/IP device last received its IP address from. This property shall be read-only and optional.

5.3.2.5.15 PID_MAC_ADDRESS (PID = 64)

— Property name: MAC Address;

— Datapoint Type: None.

This property shall contain the MAC address (48 bit) of the KNXnet/IP device. The value shall be set by the manufacturer and shall be read only.

5.3.2.5.16 PID_SYSTEM_SETUP_MULTICAST_ADDRESS (PID = 65)

— Property name: System Setup Multicast Address;

— Datapoint Type: None.

This property shall contain the KNXnet/IP System Setup Multicast Address. The value of this property shall be fixed at 224.0.23.12.

5.3.2.5.17 PID_ROUTING_MULTICAST_ADDRESS (PID = 66)

— Property name: Routing Multicast Address;

— Datapoint Type: None.

This property shall contain the KNXnet/IP Routing Multicast Address. The default value of this property shall be equal to the KNXnet/IP System Setup Multicast Address. The KNXnet/IP Routing Multicast Address may be set at run-time. A changed value becomes active after a reset of the KNXnet/IP device.

5.3.2.5.18 PID_TTL (PID = 67)

— Property name: Time To Live;

— Datapoint Type: None.

This property shall hold the Time-To-Live (TTL) value for IP communication across IP network routers. The default value shall be set to 16. This value may be changed.

5.3.2.5.19 PID_KNXNETIP_DEVICE_CAPABILITIES (PID = 68)

— Property name: KNXnet/IP Device Capabilities;

— Datapoint Type: None.

The list of KNXnet/IP device capabilities shall provide information about which features are implemented in the KNXnet/IP device. A value of ‘1’ always means "available", ‘0’ means "not available/not supported".

Table 18 — Device Capabilities

Bit	Description
0	Device Management
1	Tunnelling
2	Routing
3	Remote Logging
4	Remote Configuration and Diagnosis
5	Object Server
6	Reserved
7	Reserved
8	Reserved
9	Reserved
10	Reserved
11	Reserved
12	Reserved
13	Reserved
14	Reserved
15	Reserved

5.3.2.5.20 PID_KNXNETIP_DEVICE_STATE (PID = 69)

— Property name: KNXnet/IP Device State;

— Datapoint Type: None.

This property shall be implemented by any KNXnet/IP server.

This Property shall contain status information of the KNXnet/IP device.

The property shall be evented i.e. if the value of the property changes the current value shall be sent using M_PropInfo.ind.

Table 19 — Device State

Bit	Description
0	KNX Fault: is set if KNX network cannot be accessed
1	IP Fault: is set if IP network cannot be accessed
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

5.3.2.5.21 PID_KNXNETIP_ROUTING_CAPABILITIES (PID = 70)

— Property name: KNXnet/IP Routing Capabilities;

— Datapoint Type: None.

This property shall be implemented by devices providing KNXnet/IP Routing.

This Property shall contain the description of the device's KNXnet/IP Routing capabilities. It shall provide information about which features are implemented in the KNXnet/IP device.

A value of '1' always means "available" or "yes", '0' means "not available/not supported" or "no".

Table 20 — Routing Capabilities

Bit	Description
0	Statistics, Queue overflow error count, implemented
1	Statistics, Transmitted telegram info count, implemented
2	Priority/FIFO implemented
3	Multiple EIB/KNX installations supported
4	Group Address mapping supported
5	Reserved
6	Reserved
7	Reserved

If statistics for queue overflow error count (bit 0) or for transmitted telegram count (bit 1) is implemented then the corresponding bit is set. These are optional features.

If priority FIFO is implemented then the corresponding bit is set. This is an optional feature.

If multiple KNX installations are supported by a KNXnet/IP router the corresponding bit is set. This is an optional feature.

If a KNXnet/IP router supports mapping of Group Addresses between KNX installations the corresponding bit is set. This is an optional feature.

5.3.2.5.22 PID_PRIORITY_FIFO_ENABLED (PID = 71)

— Property name: Priority FIFO Enabled;

— Datapoint Type: None.

This property shall be implemented by devices implementing priority FIFO as described in 5.5, sub-clause "Forwarding rules".

This property shall show if priority FIFO is enabled (1) or disabled (0).

5.3.2.5.23 PID_QUEUE_OVERFLOW_TO_IP (PID =72)

— Property name: Queue Overflow to IP;

— Datapoint Type: None.

This property shall be implemented by devices providing KNXnet/IP Routing.

This property shall contain the number of telegrams lost due to an overflow of the queue to the IP network.

This property shall be an unsigned integer (~ 64 000) and is for informational purposes only. It may be reset by a scheme at the discretion of the manufacturer. The count does not wrap around when the maximum is reached.

5.3.2.5.24 PID_QUEUE_OVERFLOW_TO_KNX (PID = 73)

— Property name: Queue Overflow to KNX;

— Datapoint Type: None.

This property shall be implemented by devices providing KNXnet/IP Routing.

This property shall contain the number of telegrams lost due to an overflow of the queue to the KNX subnetwork.

This property shall be an unsigned integer (~ 64 000) and is for informational purposes only. It may be reset by a scheme at the discretion of the manufacturer. The count does not wrap around when the maximum is reached.

5.3.2.5.25 PID_MSG_TRANSMIT_TO_IP (PID = 74)

— Property name: Telegrams Transmitted to IP;

— Datapoint Type: None.

This property SHOULD be implemented by devices providing KNXnet/IP Routing.

This property shall show the number of telegrams successfully transmitted to the IP network.

This shall include the number of KNXnet/IP telegrams of any kind (KNXnet/IP Core, KNXnet/IP Tunnelling, KNXnet/IP Routing, KNXnet/IP Device Management etc.) associated with any KNX Individual Address that is successfully sent on the IP network. This shall include KNXnet/IP ACK telegrams.

5.3.2.5.26 PID_MSG_TRANSMIT_TO_KNX (PID = 75)

— Property name: Telegrams Transmitted to KNX;

— Datapoint Type: None.

This property SHOULD be implemented by devices providing KNXnet/IP Routing.

This Property shall contain the number of telegrams successfully transmitted by a KNXnet/IP Router to the KNX Subnetwork.

For a KNX IP device, this Property shall contain the number of telegrams received and forwarded to its application.

5.3.2.5.27 PID_FRIENDLY_NAME (PID = 76)

— Property name: Friendly Name;

— Datapoint Type: None.

This property shall be implemented by any KNXnet/IP device.

This property shall contain a string with a human readable (friendly) name for the KNXnet/IP device.

The device friendly name may be any NULL (00h) terminated ISO/IEC 8859-1 [7] character string with a maximum length of 30 octets. Unused octets are filled with the NULL (00h) character.

5.3.2.5.28 PID_ROUTING_BUSY_WAIT_TIME (PID = 78)

— Property name: Routing Busy Wait Time;

— Datapoint Type: None.

The Property PID_ROUTING_BUSY_WAIT_TIME shall be accessible via the KNXnet/IP Parameter Object of a KNXnet/IP Router or KNX IP device.

This Property shall hold the value for the wait time t_w sent with a ROUTING_BUSY frame. The default value shall be 100 ms. The permissible value range is any integer value between 20 ms and 100 ms.

This Property is mandatory for devices implementing KNXnet/IP or KNX IP.

5.3.2.6 Transport Layer interface

5.3.2.6.1 Switching to the cEMI Transport Layer in the cEMI Server by opening a KNXnet/IP Device Management connection

5.3.2.6.1.1 Use

This shall be implemented by devices that use the KNXnet/IP Device Management.

5.3.2.6.1.2 Activation of the cEMI Transport Layer interface

If a KNXnet/IP Device Management connection is established, the Management Server shall implicitly switch its Transport Layer to “cEMI Transport Layer mode”. The cEMI Server Transport Layer shall issue a T_Connect.ind primitive to the remote Application Layer to indicate that a Transport Layer connection is established.

This Transport Layer operation mode shall last for the duration of the KNXnet/IP Device Management connection.

The Transport Layer operation mode shall be reset to its default value if the KNXnet/IP Device Management connection is broken by the KNXnet/IP Device Management Client or - Server device (e.g. when the KNXnet/IP Device Management connection times out).

General exception handling

If a cEMI Server does not support the cEMI Transport Layer communication mode and it receives a cEMI T_Data_Individual.req – or cEMI T_Data_Connected.req frame, then

— this frame shall be ignored by the cEMI Server, and

— on IP, the originating KNXnet/IP DEVICE_CONFIGURATION_REQUEST frame that transports this cEMI-frame shall be confirmed by a KNXnet/IP DEVICE_CONFIGUARATION_ACK frame.

5.3.2.6.1.3 Supervising connection timeouts on the host protocol

The KNXnet/IP or KNX IP device shall deactivate the cEMI Transport Layer Mode and return to the normal Transport Layer mode if the KNXnet/IP Device Management connection is broken.

The KNXnet/IP Device Management may be broken by the KNXnet/IP Device Management Client (ETS), or autonomously by the KNXnet/IP Device Management Server (device) if the KNXnet/IP Device Management connection times out.

5.3.2.6.1.4 T_Data_Individual.con and T_Data_Connected.con

The cEMI Transport Layer instance shall provide the T_Data_Individual.con respectively the T_Data_Connected.con to the Application Layer after reception of the DEVICE_CONFIGURATION_ACK frame.

5.3.2.6.1.5 Controlling sequencing of the cEMI messages

This is guaranteed by the KNXnet/IP Device Management protocol.

5.3.2.6.1.6 Deactivation of the cEMI Transport Layer interface

The cEMI Server shall deactivate the cEMI Transport Layer interface if the KNXnet/IP Device Management connection is closed.

This may be caused by any of the following (not exclusive):

— a request by the KNXnet/IP Device Management Client to close the KNXnet/IP Device Management connection, or

— a time-out of the KNXnet/IP connection, etc.

When the cEMI Server Transport Layer is deactivated, the cEMI Server shall issue a T_Disconnect.ind primitive to the remote Application Layer to indicate that the Transport Layer connection is closed.

5.3.2.7 Object Types

5.3.3 Implementation rules and guidelines

5.3.3.1 Introduction

This clause of the standard describes the implementation details and features of KNXnet/IP Device Management.

5.3.3.2 Discovery and self-description

Every KNXnet/IP device shall support discovery and self-description according to clause 5.2 "Core".

5.3.3.3 Device Management

A KNXnet/IP device SHOULD implement Device Management (see clause 5.1 “Overview”, sub-clause "KNXnet/IP device classes").

Device Management shall use the common KNXnet/IP port number 3671.

5.3.3.4 Security

Device Management frames are not encrypted or otherwise secured.

5.3.3.5 Error handling

5.3.3.5.1 Introduction

Some errors may occur in normal operation, e.g. due to unplugged network connections. These errors are reflected in the device parameter object.

If a property access failure occurs, this is noted in the configuration response.

5.3.3.5.2 KNX net failure

Should the KNXnet/IP device not be able to transmit telegrams over the KNX subnet for five seconds, the corresponding bit in the PID_KNXNETIP_DEVICE_STATE property shall be set to ‘1’. If communication can be resumed, the bit shall be set to ‘0’.

5.3.3.5.3 IP net failure

Should communication to the IP network fail for more than five seconds the corresponding bit in the PID_KNXNETIP_DEVICE_STATE property shall be set to ‘1’. If communication can be resumed, the bit shall be set to ‘0’.

5.3.3.5.4 Queue overflow

Should one of the two routing queues overflow and queue overflow statistics are implemented (see 5.3.2.4.21), the corresponding counter is increased.

5.3.3.6 Device statistics and status information

A KNXnet/IP device shall provide statistics and status information (see also 1.1.1, device parameter object description). All values are unsigned, and all counts do not wrap around when max. is reached (property can then be set to 0).

Table 21 — Device statistics

Name	Type	Size	Description
Queue overflow to IP	Error	2 octets	Number of telegrams lost because queue to IP overflowed
Queue overflow to KNX	Error	2 octets	Number of telegrams lost because queue to KNX overflowed
Telegrams transmitted to IP	Info	4 octets	Number of telegrams successfully transmitted to IP
Telegrams transmitted to KNX	Info	4 octets	Number of telegrams successfully transmitted to KNX

5.3.4 Data packet structures

5.3.4.1 Introduction

All KNXnet/IP data packets, or frames, have a common header, consisting of the protocol version, length information, and the KNXnet/IP service type identifier.

Requests sent to connectionless KNXnet/IP endpoints shall include information about the return address. This HPAI for the reception of the response information shall always be the first data of the KNXnet/IP body of all these requests. Additional KNXnet/IP service related data may follow. Response packets do not contain this kind of return address information.

5.3.4.2 IP Configuration and Management

5.3.4.2.1 Device management services

Two service types shall be defined for Device Management of KNXnet/IP devices: DEVICE_CONFIGURATION_REQUEST and DEVICE_CONFIGURATION_ACK.

Table 22 — KNXnet/IP Device Management service type identifiers

Service name	Code	V.	Description
DEVICE_CONFIGURATION_REQUEST	0310h	1	Reads/Writes KNXnet/IP device configuration data (Interface Object properties)
DEVICE CONFIGURATION ACK	0311h	1	Sent by a KNXnet/IP device to confirm the reception of the DEVICE_CONFIGURATION_REQUEST

5.3.4.2.2 Connection Type

The connection type value for the connection type DEVICE MGMT_CONNECTION shall be 03h.

Refer to 5.2 for more details.

5.3.4.2.3 Connection Request Information

The Connection Request Information (CRI) shall have no options.

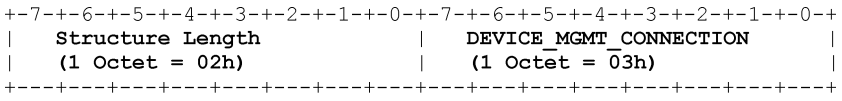


Figure 31 — KNXnet/IP Device Management CRI binary format

5.3.4.2.4 Connection Response Data Block

The Connection Response Data Block (CRD) shall have no options.

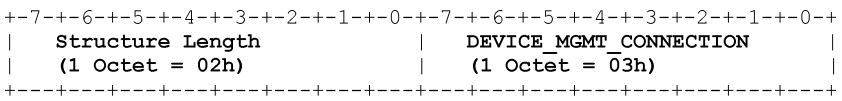


Figure 32 — KNXnet/IP Device Management CRD binary format

5.3.4.2.5 Configuration Message

DEVICE_CONFIGURATION_REQUEST shall carry a cEMI frame with the configuration message. The configuration message shall contain a local device management service as defined in Annex D:

Table 23

Supported services	KNXnet/IP Device Management version	
	1	2
KNXnet/IP Client → KNXnet/IP Server		
— M_PropRead.req	M	M
— M_PropWrite.req	M	M
— M_Reset.req	M	M
— M_FuncPropCommand.req	M	M
— M_FuncPropStateRead.req	M	M
— cEMI T_Data_Individual.req	X	M
— cEMI T_Data_Connected.req	X	M
KNXnet/IP Server → KNXnet/IP Client		
— M_PropRead.con	M	M
— M_PropWrite.con	M	M
— M_PropInfo.ind	M	M
— M_FuncPropStateResponse.con	M	M
— cEMI T_Data_Individual.ind	X	M
— cEMI T_Data_Connected.ind	X	M

These property services shall use Interface Object Type and Interface Object Instance instead of local Interface Object Index as addressing scheme for the device management. M_PropInfo.ind shall be used for the evented DeviceState property.

M_FuncPropCommand.req, M_FuncPropStateRead.req, and M_FuncPropStateResponse.con are described in Annex D.

5.3.4.2.6 DEVICE_CONFIGURATION_REQUEST

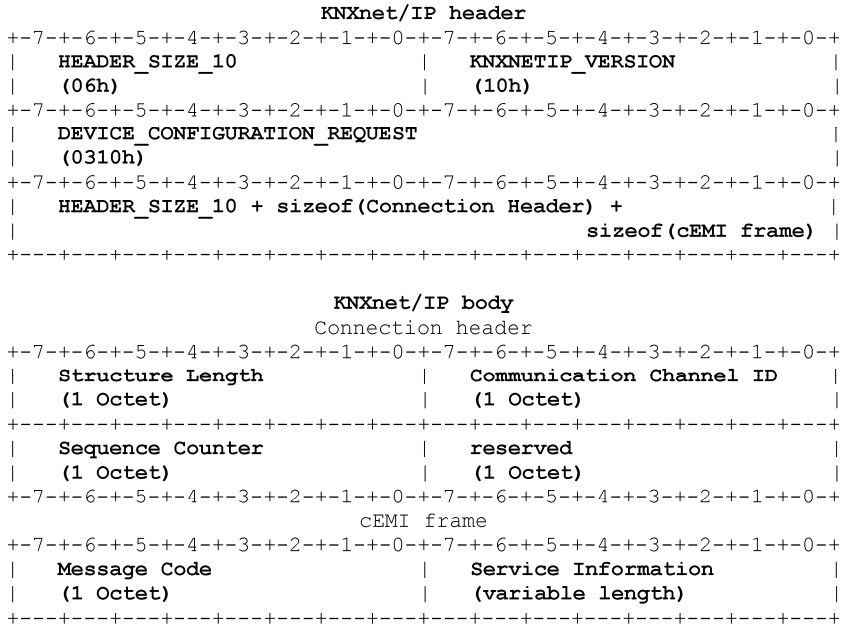


Figure 33 — DEVICE_CONFIGURATION_REQUEST frame binary format

5.3.4.2.7 DEVICE_CONFIGURATION_ACK

The configuration response includes the configuration header as well as a status field. The status field indicates success or failure of the requested operation, providing some error information.

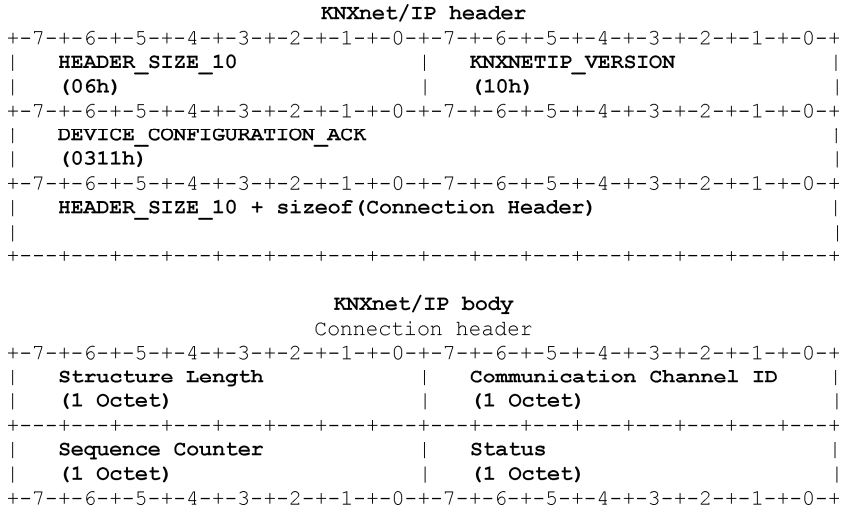


Figure 34 — DEVICE_CONFIGURATION_ACK frame binary format

Table 24 — Configuration status code

Constant Name	Value	Description
E_NO_ERROR	00h	The message was received successfully

5.3.5 Certification

5.3.5.1 Introduction

This clause provides information on the test procedures and requirements of the certification process.

Table 25 — Service support matrix

Service name	Sent from… to…	Implementation is
DEVICE_CONFIGURATION_REQUEST	Client → Server Server → Client	M
DEVICE_CONFIGURATION_ACK	Client → Server Server → Client	M

5.3.5.2 Test cases

5.3.5.2.1 Introduction

Listed here are a number of test cases a KNXnet/IP router implementation should be checked against.

5.3.5.2.2 Normal operation

Normal operation means device configuration.

Table 26 — Normal operation

Nr.	Description	Expected Result	
1.1	Device receives configuration telegram from KNX subnet addressed to the device	If valid configuration telegram	Parameters are set
		If invalid configuration telegram is received	Error message is returned
1.2	Device receives configuration telegram from IP	If valid configuration telegram	Parameters are set
		If invalid configuration telegram is received	Error message is returned

5.3.5.2.3 Parameterisation through IP

This requires a valid connection through the control endpoint of the service container, which in turn has to be searched by the discovery service before.

Table 27

Nr.	Description	Expected Result
2.1	Read device capabilities via a DEVICE_CONFIGURATION_REQUEST	A DEVICE_CONFIGURATION_ACK message with configuration status E_NO_ERROR and 4 bytes of data

5.4 Clause 4: Tunnelling

5.4.1 Scope

This Clause "Tunnelling" of the KNXnet/IP standard describes point-to-point exchange of KNX telegrams over an IP network between a KNXnet/IP device acting as a server and a KNXnet/IP client for configuration and diagnostics. KNX telegrams are encapsulated inside IP datagrams. KNXnet/IP Tunnelling does not address timing issues caused by IP data network latency greater than one second. Refer to chapter 6, Remote Configuration and Diagnosis.

5.4.2 Tunnelling of KNX telegrams

5.4.2.1 Introduction

KNXnet/IP Tunnelling is characterised by the KNXnet/IP client (e.g. Engineering Tool Software) sending a single KNX telegram in an IP frame and waiting until the response arrives or a time-out is reached.

5.4.2.2 Tunnelling

5.4.2.2.1 General

KNX frames shall always be sent within a TUNNELLING_REQUEST frame. This frame shall contain the KNX data packet in cEMI format. cEMI format shall be supported by all KNXnet/IP devices.

After a communication channel has been established, the following KNX services shall be supported by a KNXnet/IP version 1 implementation:

- KNXnet/IP Tunnelling on KNX Data Link Layer
  - Client → Server L_Data.req, M_Reset.req
  - Server → Client L_Data.con, L_Data.ind
- KNXnet/IP Tunnelling in cEMI Raw mode
  - Client → Server L_Raw.req, M_Reset.req
  - Server → Client L_Raw.con, L_Raw.ind
- KNXnet/IP Tunnelling on KNX Busmonitor
  - Client → Server n.a.
  - Server → Client L_Busmon.ind

5.4.2.2.2 Tunnelling on KNX Data Link Layer

Implementation of Tunnelling on KNX Data Link Layer is MANDATORY.

Each KNXnet/IP Tunnelling connection shall correspond with a KNX Individual Address, i.e. when the Tunnelling connection is established the KNXnet/IP server shall assign a KNX Individual Address to it. This KNX Individual Address shall be returned in the CONNECT_RESPONSE frame Connection Response Data Block (CRD). If the KNXnet/IP server assigns its own KNX Individual Address to the Tunnelling connection then management of the KNXnet/IP server shall not be possible via Tunnelling messages or from the KNX Subnetwork. The KNX Individual Address of the KNXnet/IP server itself shall be obtained by assignment through ETS. This address may be used for a Tunnelling connection with the implication stated above; see Figure 35, A.

KNX Individual Addresses for any additional Tunnelling connections shall be assigned by ETS. The additional KNX Individual Addresses shall be stored in property PID_ADDITIONAL_INDIVIDUAL_ADDRESSES.

To prevent other Management Clients from using or assigning any of these additional IAs, the KNXnet/IP Tunnelling device shall defend its additional IA. Such Management Client will carry out the Management Procedure NM_IndividualAddress_Check. If the Management Client in this request tries to establish a Transport Layer connection to any of these IAs by sending a T_Connect-PDU, then the KNXnet/IP Tunnelling device shall act as follows.

— If no Tunnelling connection is open for the additional KNX Individual Address, then the Tunnelling device shall send a T_Disconnect-PDU.

— If the Tunnelling connection for this additional IA is currently open, then the Tunnelling device shall forward the above request as T_Connect-PDU to the connected KNXnet/IP Tunnelling Client. The KNXnet/IP Tunnelling Client then shall respond to the request.

Additional KNX Individual Addresses shall be permanent. The KNXnet/IP server shall generate IACK frames for these additionally assumed KNX Individual Addresses; see Figure 35, B.

Every KNXnet/IP Tunnelling connection shall use its own (unique) Individual Address and hence logically appears as another device on the KNX bus. Hence, it shall acknowledge telegrams even if the telegram is coming from the physically same device.

If a KNXnet/IP server also implements KNXnet/IP Routing, it shall not use its own KNX Individual Address for KNXnet/IP Tunnelling connections but shall assume KNX Individual Addresses as described above; see Figure 35, C.

A KNXnet/IP router shall not activate KNXnet/IP Tunnelling until at least one additional KNX Individual Address has been set via KNXnet/IP Device Management.

These are the bootstrap steps to take for a KNXnet/IP router.

a) KNXnet/IP Discovery of KNXnet/IP router IP address.

b) Initiation of KNXnet/IP Device Management connection to KNXnet/IP router and setting KNX Individual Address for this KNXnet/IP router.

c) Setting KNXnet/IP router additional KNX Individual Address(es).

The KNXnet/IP Tunnelling server shall only pass those KNX point-to-point addressed telegrams to the KNXnet/IP Tunnelling client that contain the KNX Individual Address of the KNXnet/IP Tunnelling connection with this KNXnet/IP Tunnelling client. All KNX telegrams on KNX point-to-multipoint communication (i.e. group addressing) shall be forwarded to the KNXnet/IP Tunnelling client.

If the KNXnet/IP Tunnelling client sends a cEMI frame L_Data.req with a KNX Individual Address (KNX Source Address) set to 0000h then the KNXnet/IP Tunnelling server shall enter the KNX Individual Address assigned to this Tunnelling connection. Otherwise, the KNX telegram shall be sent unchanged (see cEMI in Annex D).

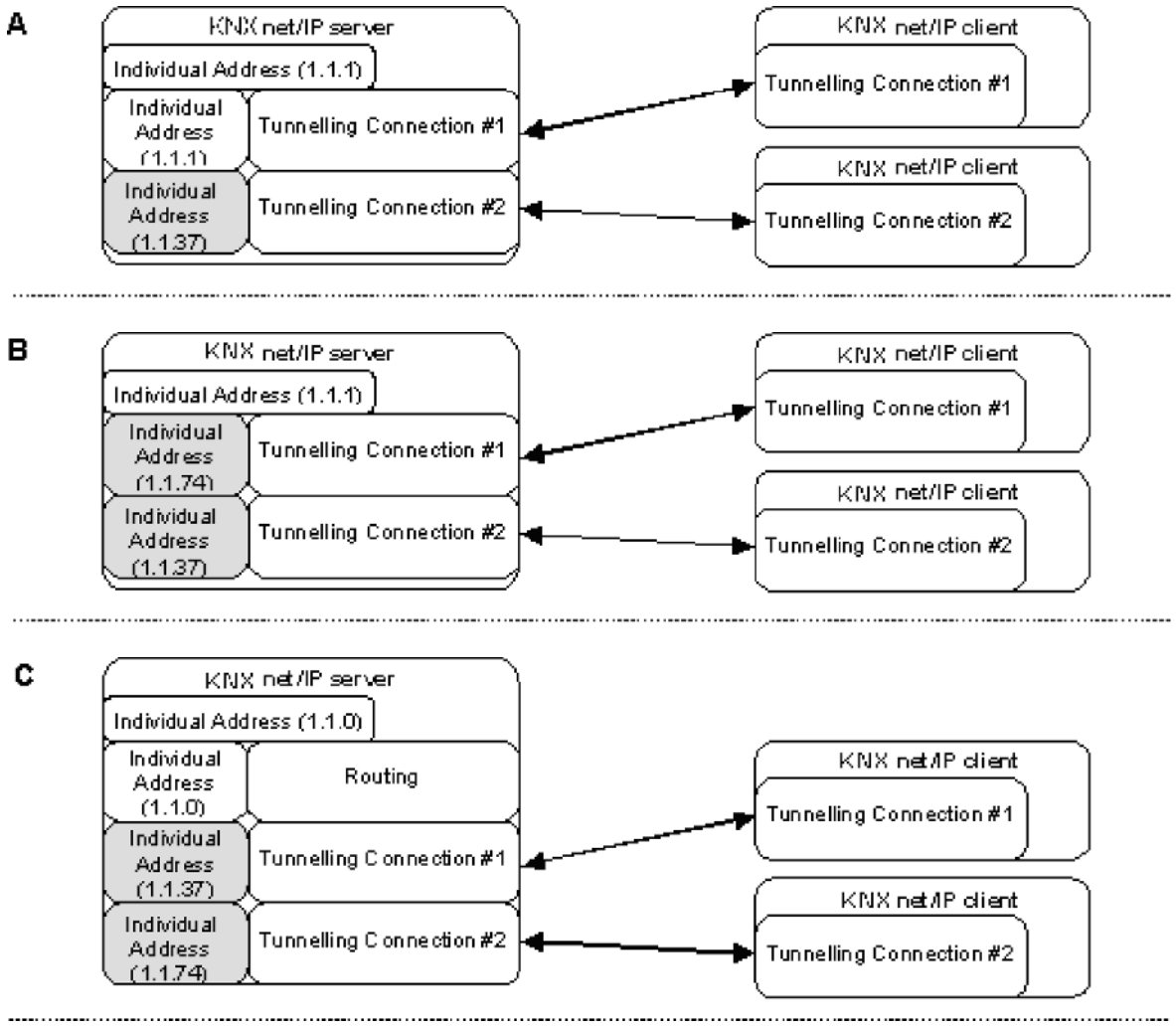


Figure 35 — Tunnelling connections and KNX Individual Addresses in the KNXnet/IP server

5.4.2.2.3 Tunnelling in cEMI Raw mode

Implementation of Tunnelling on KNX cEMI Raw mode is optional.

The KNXnet/IP Tunnelling server shall pass any KNX telegram received to the KNXnet/IP Tunnelling client.

The KNXnet/IP Tunnelling server shall not generate IACK frames in response to KNX telegrams forwarded onto a KNXnet/IP Tunnelling connection in cEMI Raw mode.

Because KNXnet/IP Tunnelling on cEMI Raw mode would therefore disable the KNXnet/IP Routing function in a KNXnet/IP router, KNXnet/IP Tunnelling on cEMI Raw mode shall not be supported in a KNXnet/IP Routing device.

5.4.2.2.4 Tunnelling on KNX Busmonitor

Implementation of Tunnelling on KNX Busmonitor is optional.

If Tunnelling on KNX Busmonitor is implemented, the KNXnet/IP server shall only support one KNXnet/IP Tunnelling connection per KNX Subnetwork. If Tunnelling on KNX Busmonitor is activated, then the KNXnet/IP server may not support any other KNXnet/IP services for the KNX Subnetwork.

Because KNXnet/IP Tunnelling on KNX Busmonitor would therefore disable the KNXnet/IP Routing function in a KNXnet/IP router, KNXnet/IP Tunnelling on KNX Busmonitor shall not be supported in a KNXnet/IP Routing device.

5.4.2.3 Timing

Tunnelling as such does not provide any mechanisms to modify timing requirements on the tunnelled data packets. More precisely, the timing requirements defined by the KNX standard are still valid while tunnelling or routing over a non-KNX network. The transfer time of tunnelled data packets shall therefore allow for operation within these timing requirements.

This implies that $t_{Tunneling\_protocol\_transfer\_time} << t_{KNX\_transfer\_time\text{-}outs}$.

5.4.2.4 Sending KNX telegrams via KNXnet/IP server to local subnet

To send a KNX telegram, the client shall send a TUNNELLING_REQUEST frame to the KNXnet/IP server, with an L_Data.req, according to the KNX standard. The client shall be connected and shall not be in busmonitor mode.

5.4.2.5 Receiving KNX telegrams via KNXnet/IP server from local subnet

When a connection is established, the KNXnet/IP server shall send a TUNNELLING_REQUEST for each telegram it receives from the local KNX Subnetwork. The KNX services of the embedded KNX telegram may be L_Data.con, L_Data.ind, or L_Busmon.ind, according to the KNX standard.

5.4.2.6 Datagram confirmation

TUNNELLING_REQUEST frames shall be confirmed using TUNNELLING_ACK frames.

If a TUNNELLING_REQUEST frame is not confirmed within the TUNNELLING_REQUEST_TIME_OUT time of one (1) second then the telegram shall be repeated once with the same sequence counter value by the sending KNXnet/IP device.

If the KNXnet/IP device does not receive a TUNNELLING_ACK frame within the TUNNELLING_REQUEST_TIMEOUT (= 1 s) or the status of a received TUNNELLING_ACK frame signalled any kind of error condition, the sending device shall repeat the TUNNELLING_REQUEST frame once and then terminate the connection by sending a DISCONNECT_REQUEST frame to the other device's control endpoint.

If a KNXnet/IP Tunnelling server receives a data packet with a sequence number that is the expected sequence number then it shall reply with a TUNNELLING_ACK (Status = E_NO_ERROR) frame and process the received frame.

If a KNXnet/IP Tunnelling server receives a data packet with a sequence number that is one less than the expected sequence number then it shall reply with a TUNNELLING_ACK (Status = E_NO_ERROR) frame and discard the frame.

If a KNXnet/IP Tunnelling server receives a data packet with a sequence number that is not equal to the expected sequence number and not equal to one less than the expected sequence number, the KNXnet/IP Tunnelling server shall not reply and shall discard the received frame.

If a KNXnet/IP Tunnelling client receives a data packet with a sequence number that is the expected sequence number then it shall reply with a TUNNELLING_ACK (Status = E_NO_ERROR) frame and process the received frame.

If a KNXnet/IP Tunnelling client receives a data packet with a sequence number that is one less than the expected sequence number then it shall reply with a TUNNELLING_ACK (Status = E_NO_ERROR) frame and discard the received frame.

If a KNXnet/IP Tunnelling client receives a frame with a sequence number that is not equal to the expected sequence number and not equal to one less than the expected sequence number, the KNXnet/IP Tunnelling client shall not reply and shall discard the received frame.

5.4.3 Configuration and Management

General device management and configuration of KNXnet/IP devices is specified in 5.3.

KNXnet/IP Tunnelling does not require any configuration beyond the general device management.

5.4.4 Frame structures

5.4.4.1 Introduction

All KNXnet/IP frames have a common header, consisting of the protocol version, length information, and the KNXnet/IP service type identifier.

5.4.4.2 Common constants

Refer to clause 5.1 “Overview” for a list of valid KNXnet/IP common constants.

5.4.4.3 Common error codes

Refer to clause 5.1 “Overview” for a list of valid KNXnet/IP common error codes.

5.4.4.4 KNX telegram tunnelling

5.4.4.4.1 General

KNXnet/IP Tunnelling is initiated by establishing a KNXnet/IP Tunnelling connection from the KNXnet/IP client, e.g. ETS, to the KNXnet/IP server. Refer to clause 5.2 “Core” for details.

5.4.4.4.2 KNXnet/IP services

The following table lists all valid KNXnet/IP service types for KNXnet/IP Tunnelling.

Table 28 — Tunnelling KNXnet/IP service type identifiers

Service name	Code	V.	Description
TUNNELLING_REQUEST	0420h	1	Used for sending and receiving single EIB/KNX telegrams between KNXnet/IP client and server
TUNNELLING_ACK	0421h	1	Sent by a KNXnet/IP device to confirm the reception of the TUNNELLING_REQUEST

5.4.4.4.3 Connection Type

The connection type value for the connection type TUNNEL_CONNECTION shall be 04h.

Refer to 5.2 for more details.

5.4.4.4.4 Connection Request Information (CRI)

The Connection Request Information (CRI) contains the requested Tunnelling KNX layer.

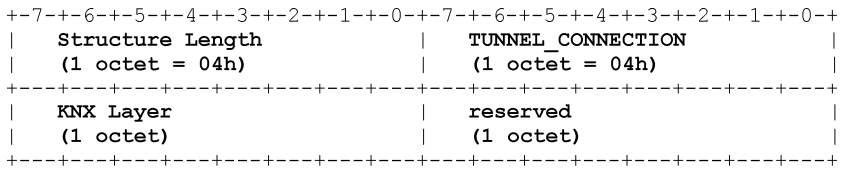


Figure 36 — Tunnelling CRI binary format

Table 29 — Tunnelling KNX layers

Constant name	Value	V.	Description
TUNNEL_LINKLAYER	02h	1	Establish a link layer tunnel to the KNX network
TUNNEL_RAW	04h	1	Establish a raw tunnel to the KNX network
TUNNEL_BUSMONITOR	80h	1	Establish a busmonitor tunnel to the KNX network

Table 30 — Tunnelling CONNECT_ACK error codes

Error constant	Value	V.	Description
E_NO_ERROR	00h	1	The message was received successfully.
E_TUNNELLING_LAYER	29h	1	The requested tunnelling layer is not supported by the KNXnet/IP Server device.

5.4.4.4.5 Connection Response Data Block (CRD)

The Connection Response Data Block (CRD) contains the KNX Individual Address assigned to this Tunnelling connection.

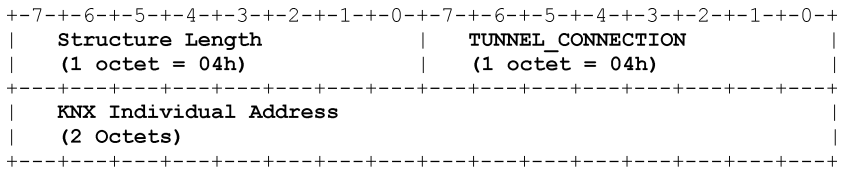


Figure 37 — Tunnelling CRI binary format

5.4.4.4.6 Connection header

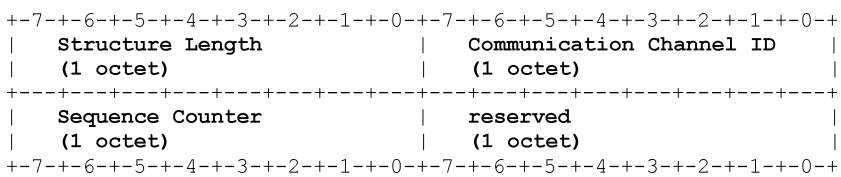


Figure 38 — Tunnelling connection header binary format

5.4.4.4.7 TUNNELLING_REQUEST

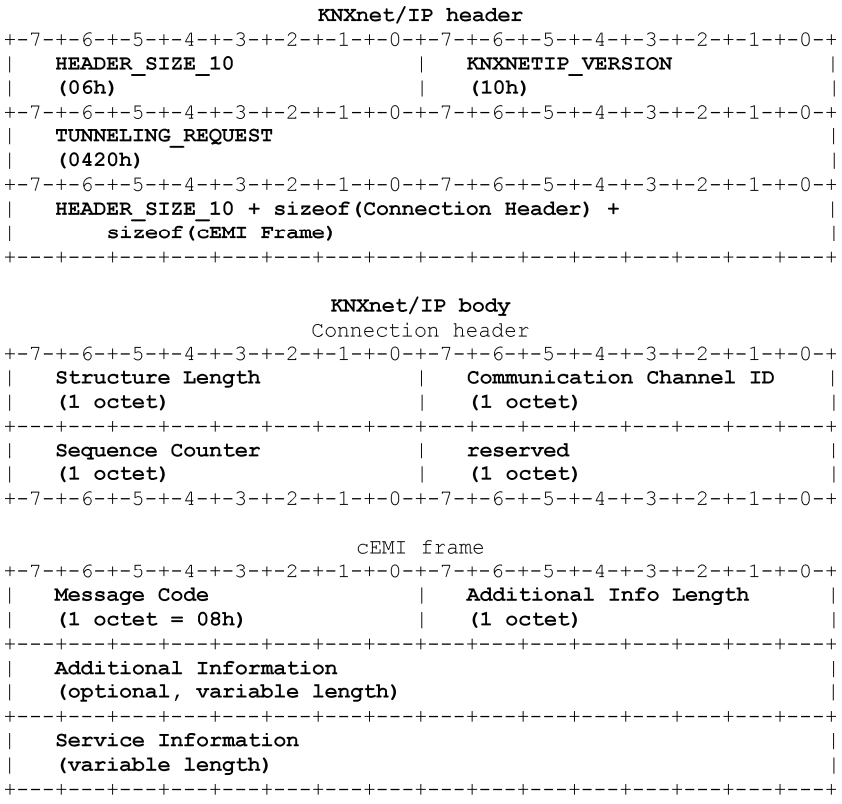


Figure 39 — TUNNELLING_REQUEST frame binary format

5.4.4.4.8 TUNNELLING_ACK

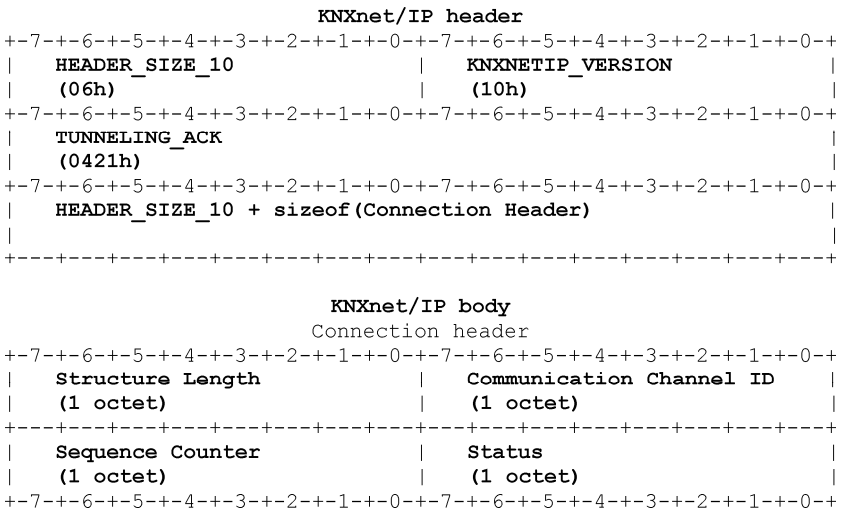


Figure 40 — TUNNELLING_ACK frame binary format

5.4.5 Certification

This clause provides information on the test procedure and requirements of the certification process.

Table 31 — Support matrix

Service name	Sent from… to…	Implementation is
TUNNELLING_REQUEST	Client → Server (L_Data.req, M_Reset.req) Server → Client (L_Data.con, L_Data.ind)	M
TUNNELLING_REQUEST	Client → Server (L_Raw.req) Server → Client (L_Data.con, L_Data.ind)	O
TUNNELLING_ACK	Client → Server Server → Client	M

5.5 Clause 5: Routing

5.5.1 Scope

This clause "Routing" describes a point-to-multipoint standard protocol for routing between KNX devices, called KNXnet/IP Routers, over an IP network.

5.5.2 KNXnet/IP Routing of KNX telegrams

5.5.2.1 Introduction

KNXnet/IP routing is defined as a set of KNXnet/IP Routers communicating over a one-to-many communication relationship (multicast), in which KNX data is transferred from one device to one or more other devices simultaneously over an IP network. A set of KNXnet/IP Routers can replace KNX Line- and Backbone Couplers and connected Main Lines respectively Backbone Lines, allowing usage of existing cabling (e.g. Ethernet) and faster transmission times (and simultaneousness) between KNX subnets. The IP network acts as a fast backbone that connects KNX subnets and is a high-speed replacement for the KNX backbone.

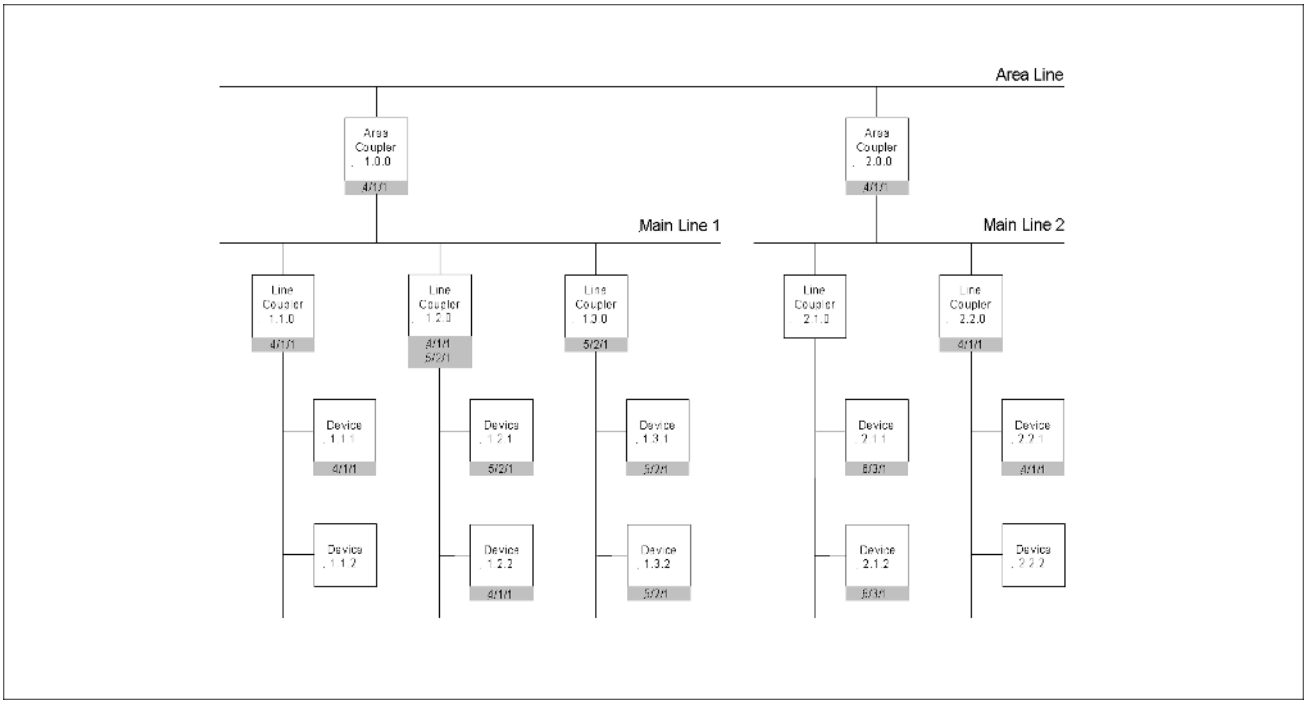


Figure 41 — KNX Group Telegram Routing

In Figure 41, a part of a KNX network is displayed. Every KNX device ("Device", Line Coupler and Backbone Coupler) has its unique KNX Individual Address that reflects the bus topology. Couplers make use of this correlation when filtering telegrams in point-to-point communication mode, using a filter mask defined by their own KNX Individual Address and telegram direction (from hierarchically lower subnetwork to hierarchically higher subnetwork or vice versa).

A group telegram sent by a KNX device is passed through the network from Coupler to Coupler (provided Filter Tables and/or routing rules allow the telegram to pass) until it reaches the target device(s). Depending on bus topology and KNX Group Address usage, it may take multiple Couplers to receive and retransmit a group telegram until it reaches all targets; the same obtains for point-to-point connection-less and connection-oriented telegrams, only there is exactly one target device.

EXAMPLE: A group telegram (on Group Address 4/1/1) transmitted by device 1.1.1 is retransmitted by Coupler 1.1.0 on Main Line 1.0, then by Coupler 1.2.0 on its Line and by Coupler 1.0.0 on the Backbone Line, by Coupler 2.0.0 on Main Line 2.0 and finally by Coupler 2.2.0 on its Line 2.2, which makes a total of five consecutive transmissions (the sixth one is handled at the same time as another transmission and is not counted here).

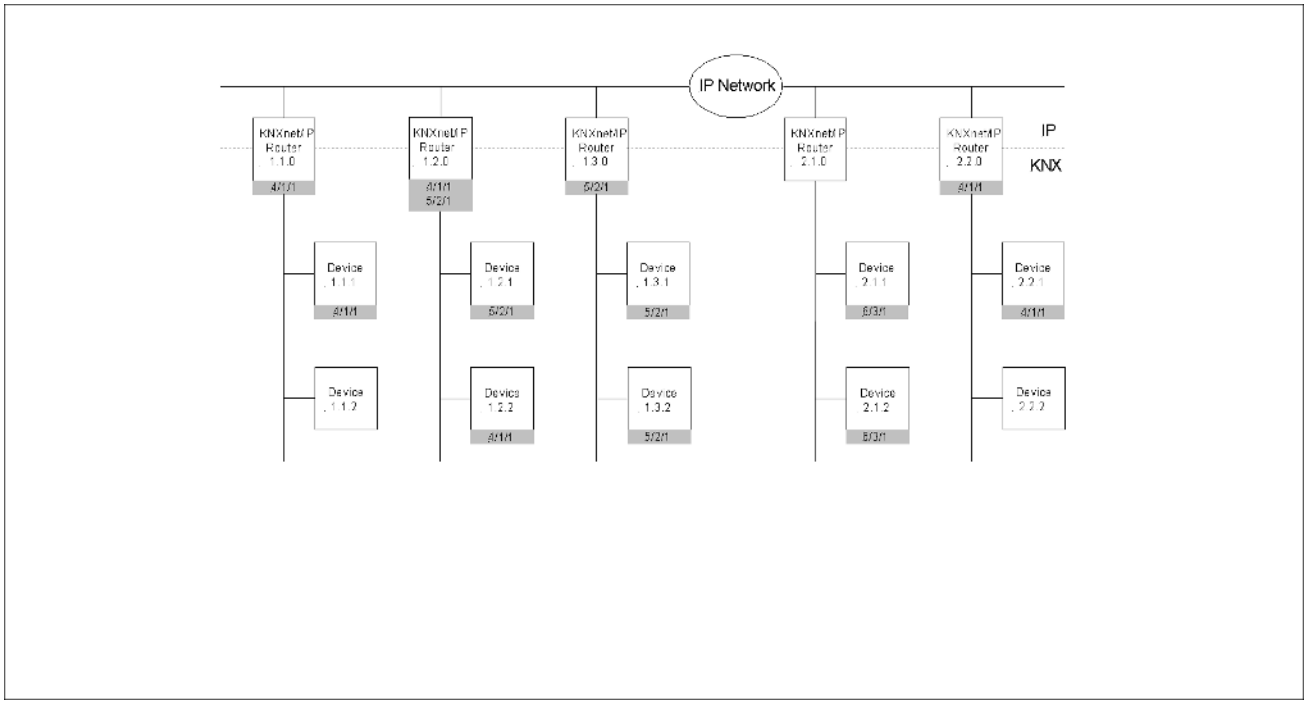


Figure 42 — KNXnet/IP Group Telegram Routing

In Figure 42, the same KNX devices as in Figure 41 are connected to KNXnet/IP Routers, which in turn are all connected to the same IP network. Every telegram from a subnetwork that matches the filter criteria in its KNXnet/IP Router is encapsulated in an IP datagram and transferred to the IP network. All KNXnet/IP Routers connected to the IP network will receive the datagram simultaneously and, when in turn the filter criteria are matched, transmit the KNX telegram to their subnetwork.

Compared to the example above: a group telegram (on Group Address 4/1/1) transmitted by device 1.1.1 is put on the IP network by KNXnet/IP Router 1.1.0 and finally by KNXnet/IP Routers 1.2.0 and 2.2.0 on their respective subnetworks, which makes a total of three consecutive transmissions. One of these transmissions is over the IP backbone and therefore very fast.

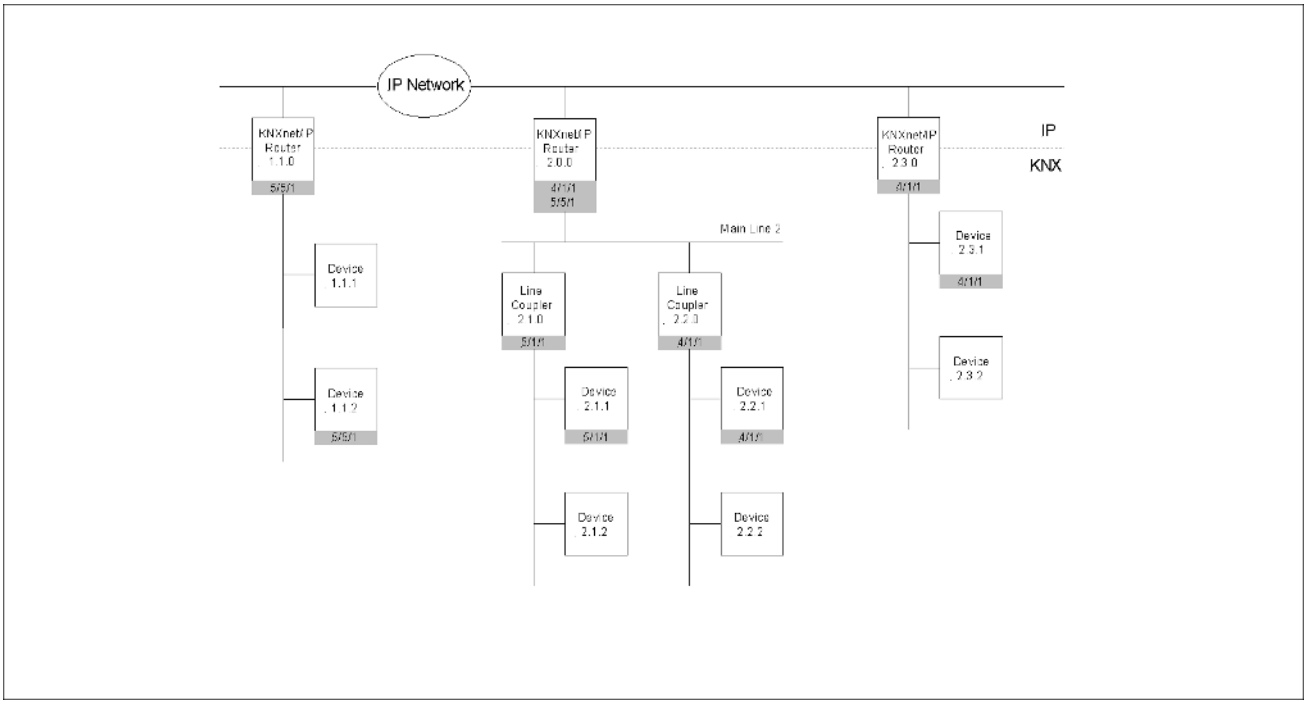


Figure 43 — Mixed topology (undesired subnetwork addressing)

Figure 43 shows a mixed topology of IP and KNX network. One KNXnet/IP Router couples a KNX Main Line to the IP network; others couple KNX Lines. It should be noticed that one of the KNX Lines would normally be located below the KNX Main Line, but it is coupled to the IP network directly.

Considering a telegram in point-to-point (connectionless or connection-oriented) communication mode sent by device 2.2.1 targeting device 2.3.2, which is routed by Line Coupler 2.2.0 to the KNX Main Line 2, Router 2.0.0 would normally not expect to route this telegram over IP. Or, reversed, for a telegram in point-to-point communication mode sent by device 1.1.1 targeting the same device 2.3.2, which is routed by router 1.1.0 to IP, router 2.0.0 would normally expect that target device to be located down its own KNX route and therefore transmit it to KNX Main Line 2, where it is, in this example, not needed.

The mixed topology is a result of the demand to use IP as a fast backbone: to take as much advantage of the fast IP network transmission as possible, most KNX Lines should be attached directly (by use of a KNXnet/IP Router) to the IP network. As KNX end devices can also be connected to KNX Main Lines, it shall be possible to attach the latter to the IP network as well. As a result, to maintain standard routing of telegrams in point-to-point connectionless and connection-oriented communication mode (and routing Filter Table computation) ETS address assignment rules have to be extended.

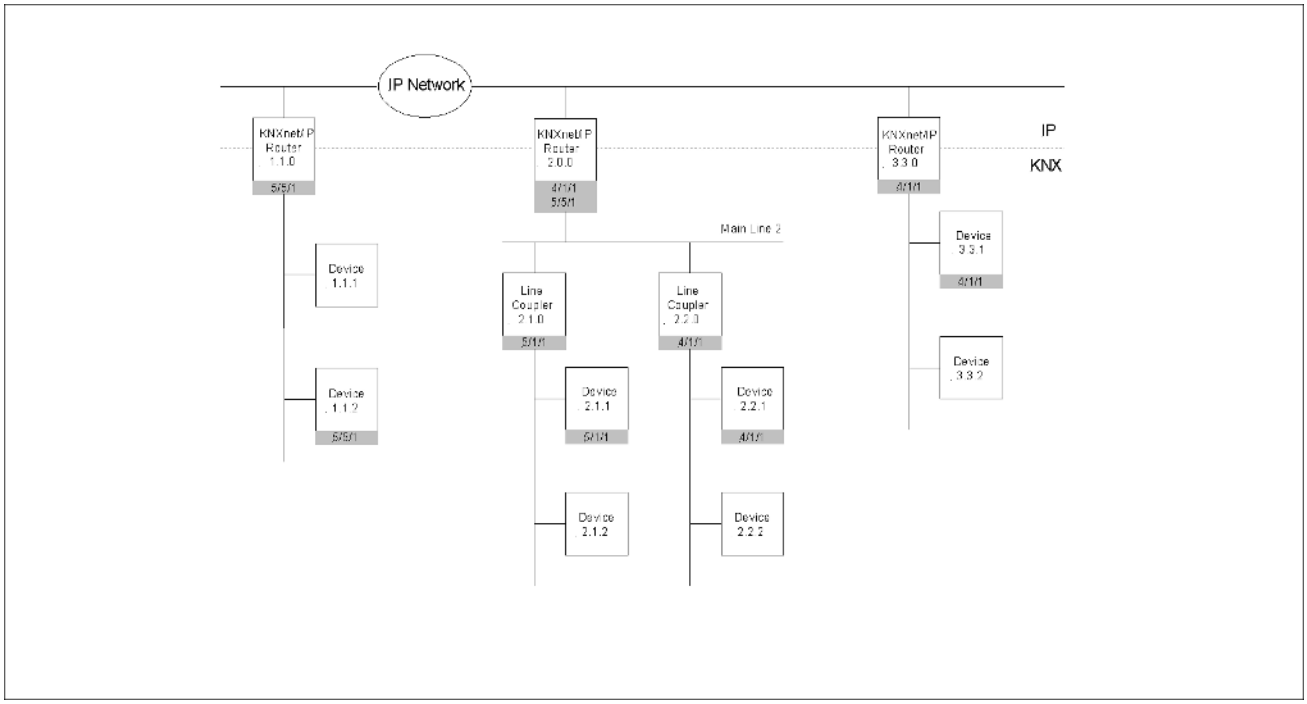


Figure 44 — Mixed topology (desired Subnetwork address assignment)

ETS shall implement the following rules for address assignment to KNXnet/IP devices.

A KNXnet/IP Router shall not be assigned to a KNX Line Coupler address (x.y.0) if the KNX Backbone Coupler address (x.0.0) is already assigned to another KNXnet/IP Router or if one or more KNX Line Couplers are already assigned to a KNX Line Coupler address (x.y.0), where y = [1 … 15]. A KNXnet/IP Router shall not be assigned to a KNX Backbone Coupler address (x.0.0) if one or more KNXnet/IP Routers are already assigned to a KNX Line Coupler address (x.y.0), where y = [1 ... 15].

These rules ensure proper routing of KNX telegrams with KNXnet/IP Routers.

Any KNXnet/IP device may be assigned to the KNX Individual Addresses in the range [(0.0.1)…(0.0.255)]. These KNXnet/IP devices shall implement KNXnet/IP Routing as the standard means of communication even though these KNXnet/IP devices are not KNXnet/IP Routers.

5.5.2.2 IP Multicasts

The one-to-many relationship requires IP multicasts to use a connectionless transport protocol. Every KNXnet/IP Router shall therefore support UDP/IP multicasts. Successful transmission of data from one router to another is not guaranteed, but on IP networks data loss is extremely unlikely. KNXnet/IP Routers shall use IGMP (Internet Group Management Protocol) to inform local IP multicast routers of the new multicast address user (IGMP v2 Membership Report), allowing KNXnet/IP routing datagrams to cross IP Routers into other segments or sub networks of the IP network.

By setting the TTL of outgoing datagrams, a KNXnet/IP Router can control how many “hops” the datagram can make, that is, how many IP routers the datagram may pass until it is discarded. As a result, the KNXnet/IP Router can be parameterised to transmit its multicast datagrams to the local IP network only. On the other hand, there is no way to prohibit multicast datagrams from other parts of the IP network to reach the KNXnet/IP Router, even when this does not announce its multicast group membership to local IP routers, because other devices may do so.

A sequence of telegrams sent by one router may also arrive at another router out of order, if these KNXnet/IP Routers are not situated on the same IP network.

5.5.2.3 Routing

5.5.2.3.1 Basics

Every installation uses the same IP multicast address and port for the transmission of routing information, as stated in the “KNXnet/IP routing HPAI”. The port number 3671 is registered at the Internet Authority for Number Assignment (IANA) for this purpose.

5.5.2.3.2 Multicast group membership

KNXnet/IP Routers do not establish communication channels between each other. A KNXnet/IP Router always transmits its data to all other KNXnet/IP Routers in the same IP multicast (group) address residing on the same network or even, depending on the TTL of the routing datagram, other networks reachable over IP routers.

KNXnet/IP Routers are by default assigned to the KNXnet/IP System Setup Multicast Address. Any telegram from the KNX network is sent to and received from this KNXnet/IP Routing Multicast Address.

Yet, if more than one KNX installation shall use the same IP network, it shall do so without interfering with other KNX installations.

The same is the case with installations exceeding 180 (= 15 * 12) lines i.e. where more than one EIB installation is installed.

In those cases, KNXnet/IP Routers of separate installations shall use different KNXnet/IP Routing Multicast Addresses.

The KNXnet/IP Routing Multicast Address denotes the EIB installation a KNXnet/IP Router is assigned to.

A KNXnet/IP Router may support routing from one EIB installation to another.

5.5.2.3.3 Hardware requirements

As every KNXnet/IP Router receives every routing datagram, a KNXnet/IP Router should be able to handle the theoretical maximum IP media traffic, mostly depending on the used physical layer.

5.5.2.3.4 Lost message handling

Depending on the configuration, a KNXnet/IP Router could receive more messages from the LAN than it can send to the KNX Subnetwork. This can lead to an overflow of the LAN-to-KNX queue and subsequent loss of a KNXnet/IP message because it could not be transferred from the network buffer to the queue.

In this event, the PID_QUEUE_OVERFLOW_TO_KNX property value shall be incremented and a ROUTING_LOST_MESSAGE notification shall be multicast to the KNXnet/IP Routing Multicast Address.

This notification allows a central supervising entity to log the routing traffic and determine potential problems with the system network design.

If the connection of the EIBnet/IP Router to the LAN is broken it cannot forward telegrams from the KNX Subnetwork to the LAN. This can lead to an overflow of the KNX-to-LAN queue and subsequent loss of a KNX message because it could not be transferred from the KNX subnetwork buffer to the queue.

In this event, the PID_QUEUE_OVERFLOW_TO_IP property value shall be incremented.

5.5.2.3.5 Flow control handling

Depending on the configuration, a KNXnet/IP Router could receive more Datagrams from the LAN than it can send to the KNX Subnetwork. This could lead to an overflow of the LAN-to-KNX queue and subsequent loss

of one or more KNXnet/IP telegrams because they could not be transferred from the network buffer to the queue to the underlying KNX Subnetwork.

For KNXnet/IP Routers and KNX IP devices, flow control shall avoid the loss of Datagrams due to overflowing queues in KNXnet/IP Routers and KNX IP devices.

Limiting the data rate of sending devices is not a solution for flow control as it does not guarantee that the incoming queue on a specific device (e.g. a KNXnet/IP Router) does not overflow because it is receiving Datagrams to be sent onto the local Subnetwork from more than one sending device. The solution is for a receiving device to indicate to all other devices that its incoming queue is filling up and it may loose Datagrams if they do not stop sending.

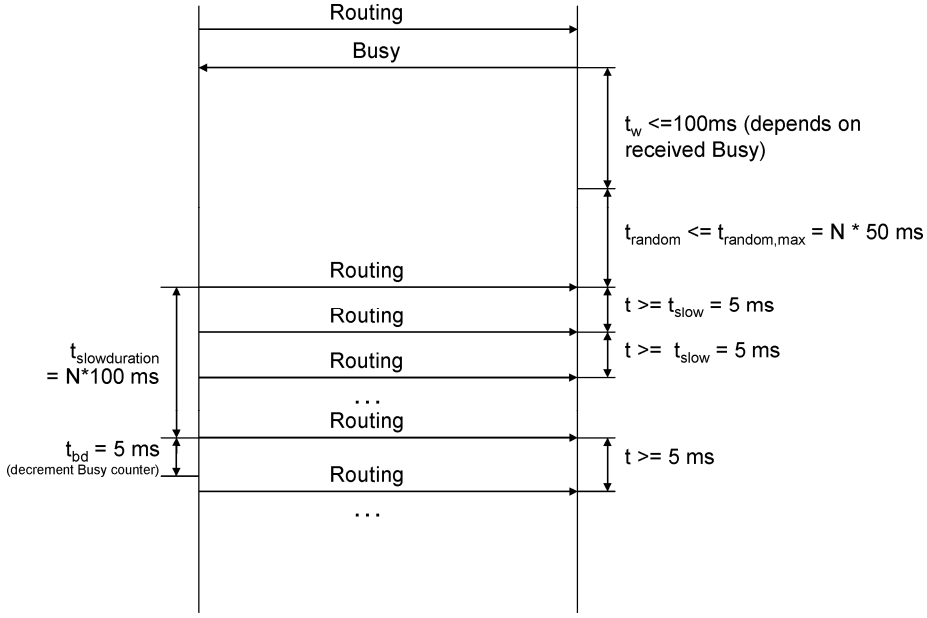


Figure 45 — Flow control with ROUTING_BUSY

Device sending ROUTING_BUSY

If the incoming queue (e.g. KNX IP to KNX TP1) of a KNXnet/IP Router or KNX IP device exceeds the number of Datagrams that can be processed (e.g. sent to the local KNX Subnetwork by a KNXnet/IP Router) within a period of $T_{process}$ then this device shall send a ROUTING_BUSY Frame with a wait time t_w. The default value for $T_{process}$ SHOULD be 100 ms and may be greater than 100 ms. t_w SHOULD resemble the time required to empty the incoming queue.

The value of t_w used by a device shall be stored in Property PID_ROUTING_BUSY_WAIT_TIME. t_w shall be at least 20 ms and shall not exceed 100 ms.

The ROUTING_BUSY Frame shall contain the routing_busy_control field. By default this routing_busy_control field shall be set to 0000h. This default value requires all KNX/IP devices and KNXnet/IP Routers to act upon receiving the ROUTING_BUSY Frame.

The threshold for sending a ROUTING_BUSY Frame with the Individual Address from the last ROUTING_INDICATION Frame should be set at five messages in the incoming queue[5)]

The threshold for sending a ROUTING_BUSY Frame to all KNX/IP devices and KNXnet/IP Routers should be set at ten messages in the incoming queue.

5) The recommended values are based on a system simulation assuming that up to 255 devices send 50 ROUTING_INDICATION datagrams per second.

The incoming queue should be able to hold at least 30 messages.

Device receiving ROUTING_BUSY

Any KNX IP device and KNXnet/IP Router shall stop sending ROUTING_INDICATION Frames as soon as it receives a ROUTING_BUSY Frame for the time t_w with a routing busy control field set to 0000h. This is where the ROUTING_BUSY Frame acts as general flow control.

If another ROUTING_BUSY Frame is received before the time t_w has elapsed the resulting time t_w shall be determined by the higher value of the remaining time of a previous ROUTING_BUSY and the value t_w received with this last ROUTING_BUSY.

A KNX IP device or KNXnet/IP Router may resume sending after the wait time t_w has expired and an additional random wait time t_{random} has passed. For an individual device the total time from receiving the ROUTING_BUSY to resuming sending shall be:

$$t_{w,total} = t_w + t_{random} \quad (1)$$

$$t_{random} = [0...1]_{random} \bullet N \bullet 50\text{ms};\ \{0 \le t_{random} \le N \bullet 50\text{ms}\} \quad (2)$$

The additional random wait time t_{random} shall be derived from a random real number in the range [0…1] multiplied by 50 ms to transmit and process a datagram times N. N is defined as the number of ROUTING_BUSY Frames received in a moving period. N shall be incremented by one with each ROUTING_BUSY Frame received after 10 ms[6)] ave passed since the last ROUTING_BUSY and decremented by one every t_{bd} = 5 ms after $t_{slowduration}$ has elapsed.

$$t_{slowduration} = N \bullet 100\text{ms} \quad (3)$$

The ROUTING_BUSY Frame allows a central supervising entity to log the routing traffic and determine potential problems with the system network design.

If the ROUTING_BUSY Frame contains a routing busy control field value not equal to 0000h then any device that does not interpret this routing busy control field shall stop sending for the time t_w. In this case, the rules for the general flow control also apply.

5.5.3 Implementation rules and guidelines

5.5.3.1 Introduction

This section of the standard describes the implementation details and features of a KNXnet/IP Router. KNX Extended frames are routed in the same way as KNX standard frames.

5.5.3.2 Discovery and self-description

Every KNXnet/IP Router shall support discovery and self-description according to 5.2.

5.5.3.3 Group Address Filtering

All KNX Group Address telegrams received by a KNXnet/IP Router are subject to Group Address filtering, unless filtering is switched off completely.

Details regarding KNX Group Address filtering are described in Annex E.

6) This value assumes that incoming ROUTING_BUSY datagrams are not due to network delays past 10 ms after the first ROUTING_BUSY is received. This avoids incrementing the counter N because more than one device exceeds the buffer trigger for sending ROUTING_BUSY at the same time.

5.5.3.4 Individual Address Filtering

All KNXnet/IP Routers have a KNX Individual Address of type x.y.0 or x.0.0, where x denotes the Area Address and y the Line Address. As with Line Couplers, all individual address telegrams are routed to the Area respectively Line denoted by the Individual Address of the KNXnet/IP Router.

This normal behaviour may be changed by setting Individual Address filters in the KNXnet/IP Router. Details regarding Individual Address filtering settings are described in Annex E.

5.5.3.5 Telegram from KNX network

Every telegram received from the KNX subnet is subject to forwarding rules according to 5.5.3.3 and 5.5.3.4. In addition, the routing counter shall be taken into consideration; see 5.5.3.9.

5.5.3.6 Datagram from IP network

5.5.3.6.1 General

The IP network adapter receives every UDP/IP datagram that targets the KNXnet/IP routing multicast IP address. The router software then has to filter the data by the following criteria:

— port number, and

— multicast IP address, and

— filter table and/or mask.

In addition, the routing counter shall be taken into consideration, see 5.5.3.9.

5.5.3.6.2 Port number filtering

All routing IP datagrams target a UDP/IP port number 3671. Only datagrams on this port are taken into consideration for transmission to the KNX subnet.

5.5.3.6.3 Group and Individual Address Filtering

The Group Address is filtered according to 5.5.3.3; the Individual Address is filtered according to 5.5.3.4. Note the KNX routing counter decrementation rules (see 5.5.3.9).

5.5.3.6.4 KNXnet/IP Device Filter Table Object

The KNXnet/IP device filter table object is defined in Annex E.

5.5.3.7 Telegram queuing and forwarding rules

5.5.3.7.1 Telegram queuing

A KNXnet/IP Router shall provide two telegram queues for at least two telegrams from the KNX subnet and two telegrams from the IP network. Any telegram that cannot be instantly forwarded to the target network is queued in the respective telegram queue.

5.5.3.7.2 Forwarding rules

The KNXnet/IP router may support two forwarding rules that are applied when more than one telegram is waiting in the telegram queue: priority/FIFO and normal FIFO mode.

1) In **priority/FIFO mode**, telegrams with the highest KNX priority (system, urgent, normal or low) shall be routed firstly, even when they were queued later than other telegrams with lower priority. If more

than one telegram has the highest priority, then the one that stayed in the queue for the longest time shall be transmitted first (like in normal FIFO mode).

2) In **normal FIFO mode**, the telegram that stayed in the queue for the longest time shall be transmitted first, regardless of KNX priorities.

Normal FIFO mode shall be implemented; priority/FIFO mode may be supported by a KNXnet/IP device.

The router shall discard any telegrams in the queue when the device detects that the connection to the medium is lost.

5.5.3.7.3 Queue overflow handling

If a telegram is to be queued when the appropriate queue is already full, one telegram has to be discarded. The telegram to be routed last (determined by the forwarding rules), taking into account the last telegram received, is cast off.

5.5.3.8 Routing data format

Telegrams received from the KNX subnet are transported as cEMI (Data Link Layer, L_Data.ind = cEMI message code 29h) messages.

5.5.3.9 KNX Routing Counter

Provided the KNX telegram Routing Counter (RC) is not zero and not seven, it shall be decremented every time a telegram passes through a KNXnet/IP Router from the KNX subnet to the IP network or vice versa.

If the RC is zero on reception of the telegram from the KNX subnet or IP network, it shall not be routed.

If the RC is seven on reception of the telegram from the KNX subnet or IP network, the telegram shall be routed without decrementing the RC.

5.5.3.10 Security

KNXnet/IP Routing frames shall not be encrypted.

This does not prevent routing KNXnet/IP Routing frames through Virtual Private Networks (VPN), which by its very nature encrypts these frames invisible from the KNXnet/IP Routers.

5.5.3.11 Error handling

5.5.3.11.1 Introduction

Some errors may occur in normal operation, e.g. due to unplugged network connections. These errors are reflected in the router parameter object.

5.5.3.11.2 KNX net failure

If the KNXnet/IP Router is not able to transmit telegrams over the KNX subnetwork for five seconds, then the corresponding bit in the property PID_KNXNETIP_DEVICE_STATE (PID = 69; as specified in 5.3) in the KNXnet/IP Parameter Object shall be set to ‘1’. If communication can be resumed, the bit shall be set to ‘0’.

5.5.3.11.3 IP net failure

If communication to the IP network fails for more than five seconds the corresponding section in the property PID_KNXNETIP_DEVICE_STATE (PID = 69; as specified in 5.3) in the KNXnet/IP Parameter Object shall be set to ‘1’. If communication can be resumed, the bit shall be set to ‘0’.

5.5.3.11.4 Queue overflow

Should one of the two routing queues overflow (see 5.5.3.7.3) and queue overflow statistics are implemented and activated (see 5.3 “Device Management”, PID_KNXNETIP_ROUTING_CAPABILITIES), the corresponding counter PID_QUEUE_OVERFLOW_TO_IP or PID_QUEUE_OVERFLOW_TO_KNX shall be incremented.

5.5.3.12 Router statistics and status information

A KNXnet/IP Router shall provide statistics and status information, see 5.3 “Device Management” (sub-clause "KNXnet/IP Parameter Object"). All values shall be unsigned, and all counts shall not wrap around when the maximum is reached (property can then be set to 0). This Property shall not be writable to protect the value from being changed accidentally. It may be reset to zero by removing power from the device or by a device hard reset or by another means provided by and at the discretion of the manufacturer.

5.5.4 Configuration and Management

5.5.4.1 General

General device management and configuration of KNXnet/IP devices is described in 5.3, "Device Management".

5.5.4.2 Property ID Definitions

The KNXnet/IP Parameters Object described in 5.3 contains properties specific to KNXnet/IP Routing.

Additional properties are not defined in this Clause 5.

5.5.4.3 Object Types

5.5.4.3.1 KNXnet/IP Parameter Object

The KNXnet/IP Parameter Object shall include the IP parameters for the KNXnet/IP device’s routing service container.

Refer to 5.3 for a detailed description of this Interface Object Type.

A KNXnet/IP Router shall implement this Interface Object Type.

5.5.4.3.2 Router Object

The Router Interface Object is described in Annex E.

A KNXnet/IP Router shall implement this Interface Object Type.

5.5.5 Data packet structures

5.5.5.1 KNXnet/IP services

Table 32 lists the KNXnet/IP Routing services.

Table 32 — KNXnet/IP Routing service type identifier

Service name	Code	V.	Description
ROUTING_INDICATION	0530h	1	Used for sending KNX telegrams over IP networks. This service is unconfirmed
ROUTING_LOST_MESSAGE	0531h	1	Used for indication of lost KNXnet/IP Routing messages. This service is unconfirmed
ROUTING_BUSY	0532h	1	Used for flow control of KNXnet/IP Routing frames. This service is unconfirmed

5.5.5.2 ROUTING_INDICATION

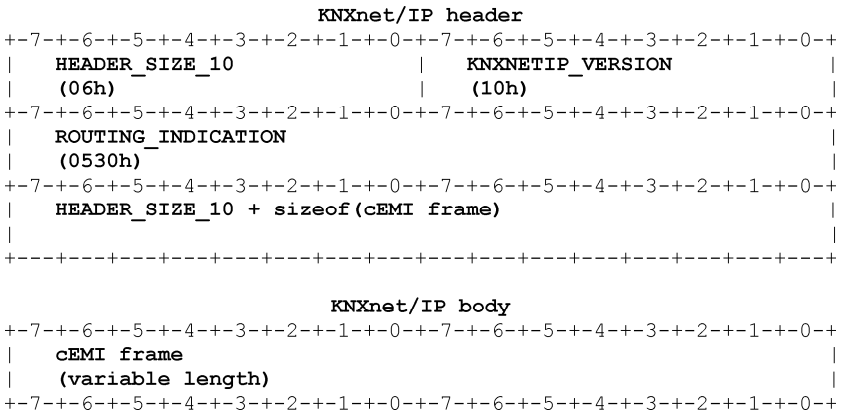


Figure 46 — ROUTING_INDICATION frame binary format

5.5.5.3 ROUTING_LOST_MESSAGE

Figure 47 — ROUTING_LOST_MESSAGE frame binary format

5.5.5.4 ROUTING_BUSY

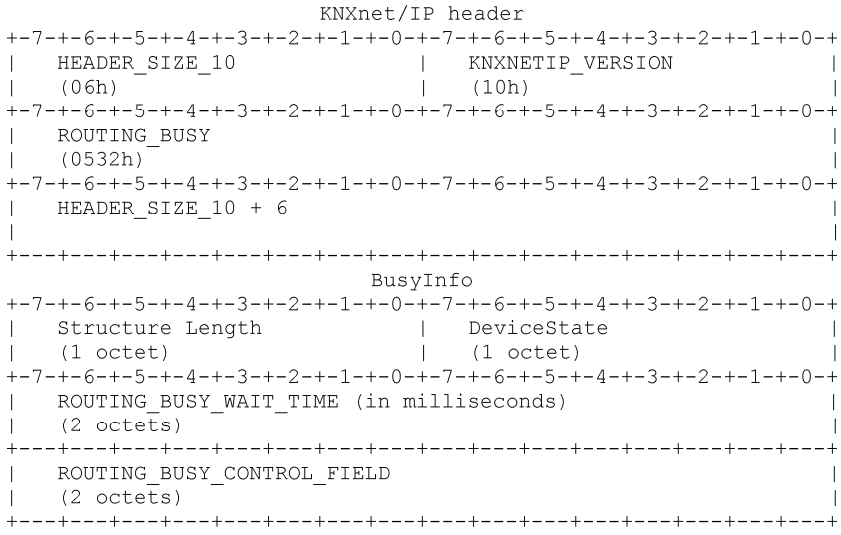


Figure 48 — ROUTING_BUSY Frame binary format

5.5.6 Certification

5.5.6.1 Introduction

This clause provides information on the test procedures and requirements of the certification process.

Table 33 — Service support matrix

Service name	Sent from… to…	Implementation is
ROUTING_INDICATION	Server → Client	M
ROUTING_LOST_MESSAGE	Server → Client	M
ROUTING_BUSY	Server → Client	M

5.5.6.2 Test cases

5.5.6.2.1 Introduction

Listed here are a number of test cases a KNXnet/IP Router implementation should be checked against.

5.5.6.2.2 Normal Operation

Normal operation means single telegram routing.

Table 34 — Normal Operation

Nr.	Description	Expected Result	
1.1	Router receives group telegram from KNX subnet	If filter settings allow routing, telegram is passed to IP (via MC)	
		If filter settings interdict routing	Telegram is discarded
1.2	Router receives individually addressed telegram from KNX subnet	If filter settings allow routing, telegram is passed to KNX	
		If filter settings interdict routing, telegram is discarded Error message is returned	
1.3	Router receives individually addressed telegram from KNX subnet	If filter settings allow routing, telegram is passed to IP (via MC)	
		If filter settings interdict routing	Telegram is discarded
1.4	Router receives individually addressed telegram from IP network	If filter settings allow routing, telegram is passed to KNX	
		If filter settings interdict routing, telegram is discarded	

Table 35 — Error cases

Nr.	Description	Expected Result
3.1	Router receives routable telegrams from KNX subnet without IP network connection	Queue overflow after some telegrams (depends on queue size). If statistics is implemented and enabled, discarded telegrams are counted in PID_QUEUE_OVERFLOW_TO_IP. If queue overflow CO is activated, a telegram shall be sent over the remaining network
3.2	Router receives routable telegrams from IP network without KNX network connection	Queue overflow after some telegrams (depends on queue size). If statistics is implemented and enabled, discarded telegrams are counted in PID_QUEUE_OVERFLOW_TO_KNX. If queue overflow CO is activated, a telegram shall be sent over the remaining network

5.6 Clause 6: Remote Diagnosis and Configuration

5.6.1 Scope

This Clause 6 “Remote Diagnosis and Configuration” of the KNXnet/IP standard provides services for remote configuration and diagnosis of a KNX installation. It addresses:

— the definition of data packets for remote diagnosis via KNXnet/IP communication, and

— the definition of data packets for remote configuration via KNXnet/IP communication.

5.6.2 Remote Diagnosis of KNXnet/IP devices

5.6.2.1 Introduction

KNXnet/IP devices shall support KNXnet/IP Core services including device discovery.

KNXnet/IP devices may receive their IP address via ETS configuration or automatically via DHCP or BootP services. In the latter case or if the network setup is unknown, the KNXnet/IP Core Device Discovery may not work or may not deliver enough information to allow for establishing a Tunnelling or other connection with the KNXnet/IP device.

As a device may have an IP address that is not reachable via unicast datagrams by the Engineering Tool Software, the remote diagnosis and configuration datagrams are used with multicast addressing. Broadcast addressing may be used if multicast addressing does not provide results in a specific network configuration. As the datagrams are transmitted via multicast or optionally via broadcast all KNXnet/IP devices receive the remote diagnosis services in parallel. A Selector is defined to allow for selecting a specific device via MAC address or Programming Mode.

5.6.2.2 REMOTE_DIAGNOSTIC_REQUEST

The REMOTE_DIAGNOSTIC_REQUEST datagram shall be transmitted using multicast or optionally via broadcast. A device that fits the selector shall respond with a REMOTE_DIAGNOSTIC_RESPONSE datagram.

5.6.2.3 REMOTE_DIAGNOSTIC_RESPONSE

The REMOTE_DIAGNOSTIC_RESPONSE datagram shall be the response to a REMOTE_DIAGNOSTIC_REQUEST datagram or to a REMOTE_BASIC_CONFIGURATION_REQUEST datagram. The response shall use the target address of the "discovery endpoint" of the HPAI in the request. The response may contain any number of DIBs. A diagnostic tool analyses only those DIBs that it recognises. All other DIBs are discarded. The device shall send all DIBs that it supports.

5.6.2.4 REMOTE_BASIC_CONFIGURATION_REQUEST

The REMOTE_BASIC_CONFIGURATION_REQUEST datagram shall be transmitted via multicast or optionally via broadcast. A device that fits the selector shall accept the configuration received with a REMOTE_DIAGNOSTIC_RESPONSE datagram. If a Device Information Block contains write-protected data then that data shall not be overwritten with the data in the DIBs of the configuration request. The configuration request shall only contain DIBs that shall be configured. This service shall be acknowledged with a REMOTE_DIAGNOSTIC_RESPONSE datagram.

5.6.2.5 REMOTE_RESET_REQUEST

The REMOTE_RESET_REQUEST datagram shall be transmitted using multicast or optionally via broadcast. A device that fits the selector shall accept the reset command without sending an acknowledgement. It should restart immediately or with a reset to factory default settings before.

5.6.3 Configuration and Management

General device management and configuration of KNXnet/IP devices is described in Clause 3, KNXnet/IP Device Management.

KNXnet/IP Remote Diagnosis and Configuration does not require any configuration beyond the general device management.

5.6.4 Data packet structures

5.6.4.1 Introduction

All KNXnet/IP data packets, or frames, shall have a common header, consisting of the protocol version, length information, and the KNXnet/IP service type identifier.

5.6.4.2 Common constants

Refer to Clause 1, Overview, for a list of valid KNXnet/IP common constants.

5.6.4.3 Common error codes

Refer to Clause 1, Overview, for a list of valid KNXnet/IP common error codes.

5.6.4.4 Remote diagnosis and configuration services

5.6.4.4.1 REMOTE_DIAGNOSTIC_REQUEST

The REMOTE_DIAGNOSTIC_REQUEST datagram shall be transmitted using multicast or optionally via broadcast. A device that fits the selector shall respond with a REMOTE_DIAGNOSTIC_RESPONSE datagram.

Figure 49 — REMOTE_DIAGNOSTIC_REQUEST frame binary format

5.6.4.4.2 REMOTE_DIAGNOSTIC_RESPONSE

The REMOTE_DIAGNOSTIC_RESPONSE datagram shall be the response to a REMOTE_DIAGNOSTIC_REQUEST datagram or a REMOTE_BASIC_CONFIGURATION_REQUEST datagram. The response shall use the target address of the "discovery endpoint" of the HPAI in the request. The response may contain any number of DIBs. A diagnostic tool analyses only those DIBs that it recognises. All other DIBs are discarded. The device shall send all DIBs that it supports.

Figure 50 — REMOTE_DIAGNOSTIC_RESPONSE frame binary format

5.6.4.4.3 REMOTE_BASIC_CONFIGURATION_REQUEST

The REMOTE_BASIC_CONFIGURATION_REQUEST datagram shall be transmitted via multicast or optionally via broadcast. A device that fits the selector shall accept the configuration received with a REMOTE_DIAGNOSTIC_RESPONSE datagram. If a Device Information Block contains write-protected data then that data shall not be overwritten with the data in the DIBs of the configuration request. The configuration request shall only contain DIBs that shall be configured. This service shall be acknowledged with a REMOTE_DIAGNOSTIC_RESPONSE datagram.

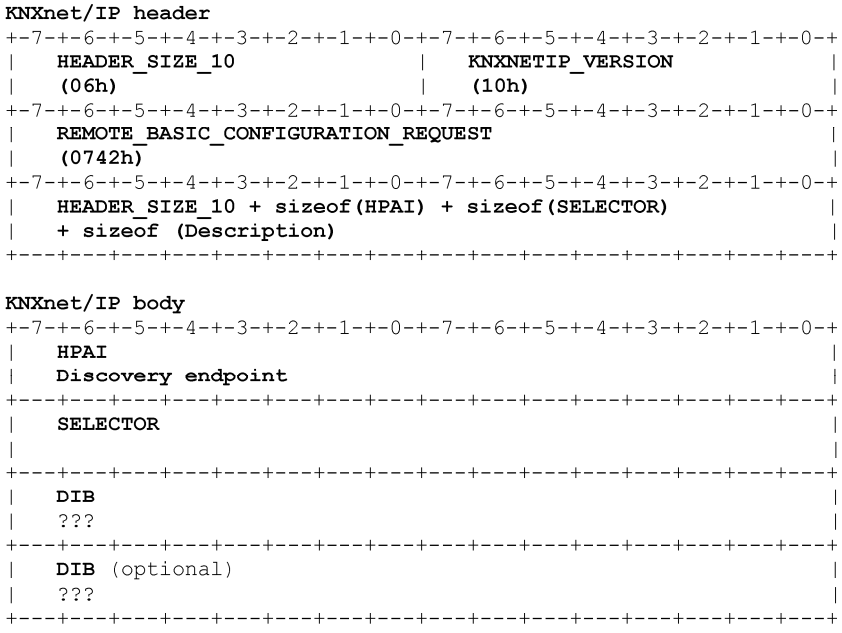


Figure 51 — REMOTE_BASIC_CONFIGURATION_REQUEST frame binary format

5.6.4.4.4 REMOTE_RESET_REQUEST

The REMOTE_RESET_REQUEST datagram shall be transmitted via multicast or optionally via broadcast. A device that fits the selector shall accept the reset command without sending an acknowledgement.

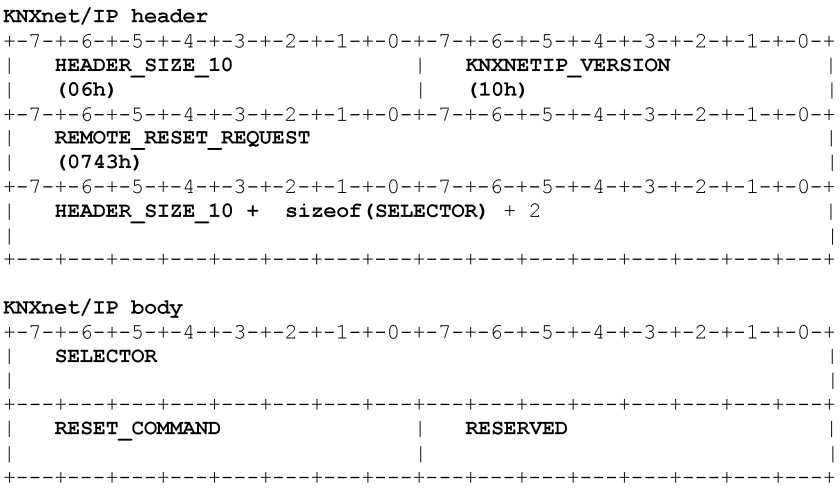


Figure 52 — REMOTE_RESET_REQUEST frame binary format

5.6.4.5 Description Information Block (DIB)

5.6.4.5.1 Introduction

The Description Information Block (DIB) shall be a set of data that is shall be accessed using the remote diagnosis and configuration services. While the Core services for device discovery and device desriptiondescription only allow reading DIBs, the Remote Diagnosis and Configuration DIBs allow reading data from the DIBs and writing data to them.

5.6.4.5.2 DIB description type codes

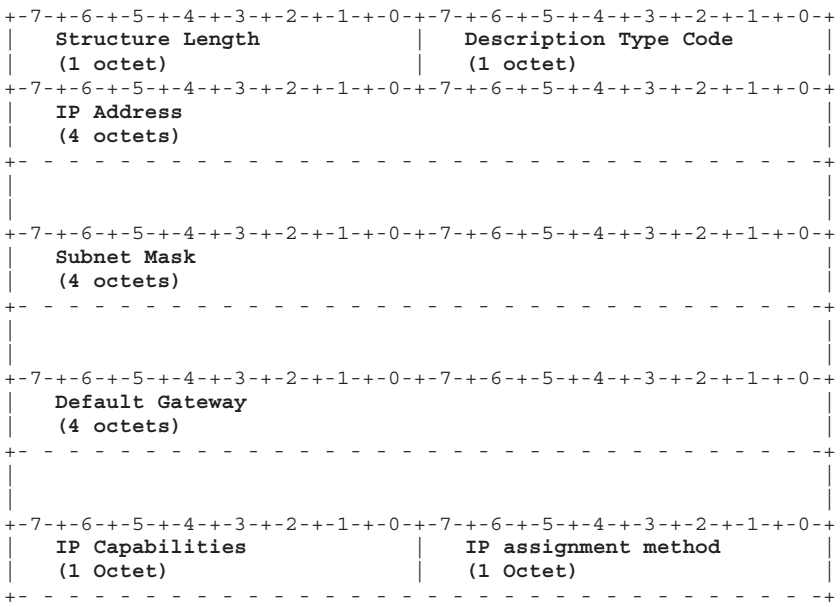
Description Information Blocks (DIB) are defined in Clause 2, KNXnet/IP Core, 5.2.5.4. Table 1 in that clause lists the description type codes. This list is extended with these description type codes used for Remote Diagnosis and Configuration:

Table 36 — Description type codes

Description type	Value	Description
IP_CONFIG	03h	IP configuration
IP_CUR_CONFIG	04h	current configuration
KNX_ADDRESSES	05h	KNX addresses

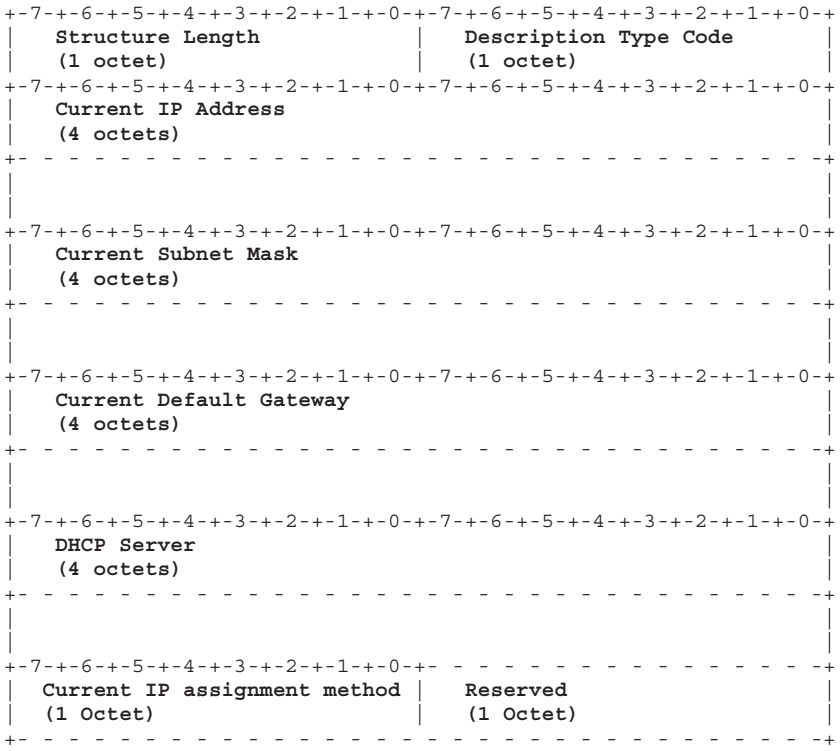
5.6.4.5.3 IP Config DIB

The IP configuration DIB shall have the following structure.



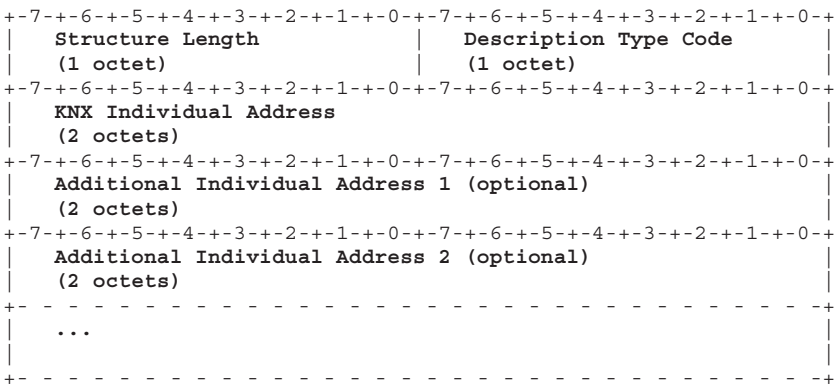
5.6.4.5.4 IP Current Config DIB

The IP current configuration DIB shall have the following structure.



5.6.4.5.5 KNX Addresses DIB

The KNX address DIB shall have the following structure.



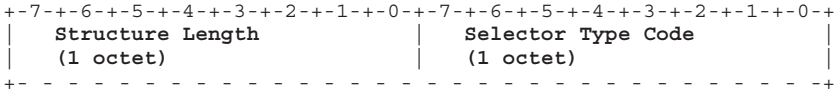
5.6.4.6 SELECTOR

As the datagrams are transmitted via multicast, all KNXnet/IP devices receive the remote diagnosis services in parallel. A Selector is defined to allow for selecting all devices or a specific device via MAC address or Programming Mode.

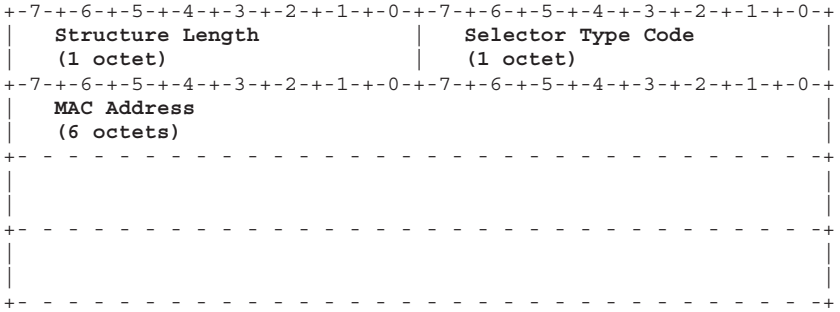
Table 37

Description type	Value	Description
PrgMode Selector	01h	selection of devices in Programming Mode
MAC Selector	02h	selection of a device via MAC address

5.6.4.6.1 PrgMode Selector



5.6.4.6.2 MAC Selector



5.6.4.7 RESET COMMAND

The reset command shall be picked from an enumeration.

The reset shall be executed immediately after receiveing the command.

Table 38

Description type	Value	Description
Restart	01h	The device is restarted.
Master Reset	02h	The device is reset to factory default settings and then restarted.

5.6.5 Certification

5.6.5.1 Introduction

This clause provides information on the test procedures and requirements of the certification process.

5.6.5.2 Support matrix

Table 39

Service name	sent from … to …	Implementation is
REMOTE_DIAGNOSTIC_REQUEST	Client → Server	M
REMOTE_DIAGNOSTIC_RESPONSE	Server → Client	M
REMOTE_BASIC_CONFIGURATION_REQUEST	Client → Server	M
REMOTE_RESET_REQUEST	Client → Server	M

Annex A
(normative)

List of codes

A.1 Introduction

This annex provides a complete listing of all codes used by KNXnet/IP.

This list shall be updated when codes are added, revised, or deleted from subsequent KNXnet/IP documents.

A.2 Common constants

Table A.1 — Common KNXnet/IP constants

Constant name	Value	V. [a]	Description
KNXNETIP_VERSION_10	10h	1	Identifier for KNXnet/IP protocol version 1.0
HEADER_SIZE_10	06h	1	Constant size of KNXnet/IP header as defined in protocol version 1.0

[a] Protocol version since which the constant, error code or service type is defined and may therefore be supported. As this is version 1 of the standard, this value is 1 for all services defined.

A.3 KNXnet/IP services

A.3.1 Service type number ranges

0200h … 020Fh KNXnet/IP Core

0310h … 031Fh KNXnet/IP Device Management

0420h … 042Fh KNXnet/IP Tunnelling

0530h … 053Fh KNXnet/IP Routing

0600h … 06FFh KNXnet/IP Remote Logging

0740h … 07FFh KNXnet/IP Remote Configuration and Diagnosis

0800h … 08FFh KNXnet/IP Object Server

A.3.2 Core KNXnet/IP services

Table A.2 — KNXnet/IP Core service type identifiers

Service name	Code	V.	Description
SEARCH_REQUEST	0201h	1	Sent by KNXnet/IP client to search available KNXnet/IP servers
SEARCH_RESPONSE	0202h	1	Sent by KNXnet/IP server when receiving a KNXnet/IP SEARCH_REQUEST
DESCRIPTION_REQUEST	0203h	1	Sent by KNXnet/IP client to a KNXnet/IP server to retrieve information about capabilities and supported services
DESCRIPTION_RESPONSE	0204h	1	Sent by KNXnet/IP server in response to a DESCRIPTION_REQUEST to provide information about the server implementation
CONNECT_REQUEST	0205h	1	Sent by KNXnet/IP client for establishing a communication channel to a KNXnet/IP server
CONNECT_RESPONSE	0206h	1	Sent by KNXnet/IP server as answer to CONNECT_REQUEST telegram
CONNECTIONSTATE_REQUEST	0207h	1	Sent by KNXnet/IP client for requesting the connection state of an established connection to a KNXnet/IP server
CONNECTIONSTATE_RESPONSE	0208h	1	Sent by KNXnet/IP server when receiving a CONNECTIONSTATE_REQUEST for an established connection
DISCONNECT_REQUEST	0209h	1	Sent by KNXnet/IP device, typically the client, to terminate an established connection
DISCONNECT_RESPONSE	020Ah	1	Sent by KNXnet/IP device, typically the server, in response to a DISCONNECT_REQUEST

A.3.3 Device Management services

Table A.3 — KNXnet/IP Device Management service type identifiers

Service name	Code	V.	Description
DEVICE_CONFIGURATION_REQUEST	0310h	1	Reads/Writes KNXnet/IP device configuration data (interface object properties)
DEVICE_CONFIGURATION_ACK	0311h	1	Sent by a KNXnet/IP device to confirm the reception of the DEVICE_CONFIGURATION_REQUEST

A.3.4 Tunnelling services

Table A.4 — Tunnelling KNXnet/IP service type identifiers

Service name	Code	V.	Description
TUNNELLING_REQUEST	0420h	1	Used for sending and receiving single EIB / KNX telegrams between KNXnet/IP client and server
TUNNELLING_ACK	0421h	1	Sent by a KNXnet/IP device to confirm the reception of the TUNNELINGTUNNELLING_REQUEST

A.3.5 Routing services

Table A.5 — KNXnet/IP Routing service type identifier

Service name	Code	V.	Description
ROUTING_INDICATION	0530h	1	Used for sending KNX telegrams over IP networks. This service is unconfirmed
ROUTING_LOST_MESSAGE	0531h	1	Used for indication of lost KNXnet/IP Routing messages. This service is unconfirmed
ROUTING_BUSY	0532h	1	Used for indication of a temporary overload of a KNXnet/IP Router or KNX/IP device. This service is unconfirmed

A.3.6 Remote Logging services

TBD

A.3.7 Remote Diagnosis and Configuration

Table A.6 — KNXnet/IP Remote Diagnosis and Configuration service type identifier

Service name	Code	V.	Description
REMOTE_DIAGNOSTIC_REQUEST	0740h	1	Used for enquiring the configuration of KNXnet/IP routers and KNX/IP devices over IP networks. This service is unconfirmed
REMOTE_DIAGNOSTIC_RESPONSE	0741h	1	Used for returning the configuration of a KNXnet/IP router or KNX/IP device over IP networks.to an enquiring KNXnet/IP client. This service is unconfirmed
REMOTE_BASIC_CONFIGURATION_REQUEST	0742h	1	Used for setting the basic configuration of a KNXnet/IP router or KNX/IP device over IP networks. This service is unconfirmed
REMOTE_RESET_REQUEST	0743h	1	Used for requiring a reset of a KNXnet/IP Router or KNX/IP device. This service is unconfirmed

A.3.8 Object Server services

TBD

A.4 Connection types

Table A.7 — Connection types

Connection type	Value	V.	Description
DEVICE_MGMT_CONNECTION	03h	1	Data connection used to configure a KNXnet/IP device
TUNNEL_CONNECTION	04h	1	Data connection used to forward EIB telegrams between two KNXnet/IP devices
REMLOG_CONNECTION	06h	1	Data connection used for configuration and data transfer with a remote logging server
REMCONF_CONNECTION	07h	1	Data connection used for data transfer with a remote configuration server
OBJSVR_CONNECTION	08h	1	Data connection used for configuration and data transfer with an Object Server in a KNXnet/IP device

A.5 Error codes

A.5.1 Common error codes

Table A.8 — Common KNXnet/IP error codes

Error constant	Value	V.	Description
E_NO_ERROR	00h	1	Operation successful
E_HOST_PROTOCOL_TYPE	01h	1	The requested host protocol is not supported by the KNXnet/IP device
E_VERSION_NOT_SUPPORTED	02h	1	The requested protocol version is not supported by the KNXnet/IP device
E_SEQUENCE_NUMBER	04h	1	The received sequence number is out of order

A.5.2 CONNECT RESPONSE status codes

Table A.9 — Common CONNECT_RESPONSE status codes

Error constant	Value	V.	Description
E_NO_ERROR	00h	1	The connection was established successfully
E_CONNECTION_TYPE	22h	1	The KNXnet/IP server device does not support the requested connection type
E_CONNECTION_OPTION	23h	1	The KNXnet/IP server device does not support one or more requested connection options
E_NO_MORE_CONNECTIONS	24h	1	The KNXnet/IP server device could not accept the new data connection because its maximum amount of concurrent connections is already busy

A.5.3 CONNECTIONSTATE_RESPONSE status codes

Table A.10 — CONNECTIONSTATE_RESPONSE status codes

Error constant	Value	V.	Description
E_NO_ERROR	00h	1	The connection state is normal
E_CONNECTION_ID	21h	1	The KNXnet/IP server device could not find an active data connection with the specified ID
E_DATA_CONNECTION	26h	1	The KNXnet/IP server device detected an error concerning the data connection with the specified ID
E_KNX_CONNECTION	27h	1	The KNXnet/IP server device detected an error concerning the EIB bus / KNX subsystem connection with the specified ID

A.5.4 Tunnelling CONNECT_ACK error codes

Table A.11 — Tunnelling CONNECT_ACK error codes

Error constant	Value	V.	Description
E_NO_ERROR	00h	1	The message was received successfully
E_TUNNELLING_LAYER	29h	1	The KNXnet/IP server device does not support the requested tunnelling layer

A.5.5 Device Management DEVICE_CONFIGURATION_ACK status codes

Table A.12 — Device Management DEVICE_CONFIGURATION_ACK status codes

Error constant	Value	V.	Description
E_NO_ERROR	00h	1	The message was received successfully

A.6 Description Information Block (DIB)

Table A.13 lists the valid description type codes.

Table A.14 lists the valid KNX medium codes.

Table A.13 — Description type codes

Description type	Value	V.	Description
DEVICE__INFO	01h	1	Device information e.g. KNX medium.
SUPP_SVC_FAMILIES	02h	1	Service families supported by the device.
IP_CONFIG	03h	1	IP configuration of the device.
IP_CUR_CONFIG	04h	1	Current IP configuration of the device.
IP_CONFIG	05h	1	KNX addresses used by / assigned to the device.
Reserved	05h – FDh	1	Reserved for future use.
MFR_DATA	FEh	1	DIB structure for further data defined by device manufacturer.
Not used	FFh	1	Not used.

Table A.14 — KNX medium codes

KNX medium	Value	V.	Description
reserved	01h	1	reserved
TP1 (KNX TP)	02h	1	KNX TP1 = KNX TP
PL110	04h	1	KNX PL110
reserved	08h	1	reserved
RF	10h	1	KNX RF
KNX IP	20h	1	KNX IP

Exactly one single bit shall be set.

A.7 Host protocol codes

Table A.15 lists the valid host protocol codes.

Table A.15 — Host protocol codes for IP network

Constant name	Value	V.	Description
IPV4_UDP	01h	1	Identifies an Internet Protocol version 4 address and port number for UDP communication
IPV4_TCP	02h	1	Identifies an Internet Protocol version 4 address and port number for TCP communication

A.8 Timeout constants

Table A.16 lists the valid timeout constants.

Table A.16 — Timeout constants

Constant name	Value	V.	Description
CONNECT_REQUEST_TIMEOUT	10 s	1	KNXnet/IP client shall wait for 10 seconds for a CONNECT_RESPONSE frame from KNXnet/IP server
CONNECTIONSTATE_REQUEST_TIMEOUT	10 s	1	KNXnet/IP client shall wait for 10 seconds for a CONNECTIONSTATE_RESPONSE frame from KNXnet/IP server
DEVICE_CONFIGURATION_REQUEST_TIMEOUT	10 s	1	KNXnet/IP client shall wait for 10 seconds for a DEVICE_CONFIGURATION_RESPONSE frame from KNXnet/IP server
TUNNELLING_REQUEST_TIMEOUT	1 s	1	KNXnet/IP Client shall wait for 1 second for a TUNNELLING_ACK response on a TUNNELLING_REQUEST frame from KNXnet/IP Server
CONNECTION_ALIVE_TIME	120 s	1	If the KNXnet/IP Server does not receive a heartbeat request within 120 seconds of the last correctly received message frame, the server shall terminate the connection by sending a DISCONNECT_REQUEST to the client's control endpoint

A.9 Internet Protocol constants

Table A.17 lists the Internet Protocol relevant values.

Table A.17 — KNXnet/IP Internet Protocol constants

Description	Value	V.
KNXnet/IP Port Number	3671	1
KNXnet/IP System Setup Multicast Address	224.0.23.12	1

Annex B
(informative)

Binary examples of KNXnet/IP IP frames

B.1 SEARCH_REQUEST

1	**06h**	header size	
2	**10h**	protocol version	
3	**02h**	\	
4	**01h**	> service type identifier 0201h	
5	**00h**	\	
6	**0Eh**	> total length, 14 octets	
7	**08h**	structure length	
8	**01h**	host protocol code, e.g. 01h, for UDP over IPv4	
9	**192**	\	
10	**168**	|	
11	**200**	> IP address of control endpoint, e.g. 192.168.200.12	
12	**12**	/	
13	**0Eh**	\	
14	**57h**	> port number of control endpoint, 3671	

Figure B.1 — SEARCH_REQUEST frame binary format: IP example

B.2 SEARCH_RESPONSE

```
•                +-------------------------------+
•          1     |             06h               |    header size
•                +-------------------------------+
•          2     |             10h               |    protocol version
•                +-------------------------------+
•          3     |             02h               | \
•                +- - - - - - - - - - - - - - - -+  > service type identifier 0202h
•          4     |             02h               | /
•                +-------------------------------+
•          5     |             00h               | \
•                +- - - - - - - - - - - - - - - -+  > total length, 78 octets
•          6     |             4Eh               | /
•                +-------------------------------+
•          7     |             08h               |    structure length (HPAI)
•                +-------------------------------+
•          8     |             01h               |    host protocol code, e.g. 01h, for UDP
over IPv4
•                +-------------------------------+
•          9     |             192               | \
•                +- - - - - - - - - - - - - - - -+ |
•         10     |             168               | |
•                +- - - - - - - - - - - - - - - -+  > IP address of control endpoint,
•         11     |             200               | |   e.g. 192.168.200.12
•                +- - - - - - - - - - - - - - - -+ |
•         12     |              12               | /
•                +-------------------------------+
•         13     |             C3h               | \
•                +- - - - - - - - - - - - - - - -+  > port number of control endpoint,
•         14     |             B4h               | /   e.g. 50100
•                +-------------------------------+
•         15     |             36h               |    structure length (DIB hardware)
•                +-------------------------------+
•         16     |             01h               |    description type code, 01h = DEVICE_INFO
•                +-------------------------------+
•         17     |             02h               |    KNX medium, 02h = TP1 (EIB TP)
•                +-------------------------------+
•         18     |             01h               |    Device Status, 01h = programming mode
•                +-------------------------------+
•         19     |             11h               | \
•                +- - - - - - - - - - - - - - - -+  > EIB physical address, e.g. 1.1.0
•         20     |             00h               | /
•                +-------------------------------+
•         21     |             00h               | \
•                +- - - - - - - - - - - - - - - -+  > Project-Installation ID, e.g. 0011h
•         22     |             11h               | /
•                +-------------------------------+
•         23     |             00h               | \
•                +- - - - - - - - - - - - - - - -+ |
•         24     |             01h               | |
•                +- - - - - - - - - - - - - - - -+ |
•         25     |             11h               | |
•                +- - - - - - - - - - - - - - - -+  > KNX device serial number,
•         26     |             11h               | |            includes manufacturer ID (2
octets)
•                +- - - - - - - - - - - - - - - -+ |
•         27     |             11h               | |
•                +- - - - - - - - - - - - - - - -+ |
•         28     |             11h               | /
•                +-------------------------------+
•         29     |             224               | \
•                +- - - - - - - - - - - - - - - -+ |
•         30     |               0               | |
•                +- - - - - - - - - - - - - - - -+  > device routing multicast address
•         31     |              23               | |   e.g. 224.0.23.12
•                +- - - - - - - - - - - - - - - -+ |
```

Figure B.2 *(1 of 2)*

```
•          32     |              12              | /
•                 +------------------------------+
•          33     |             45h              | \
•                 +- - - - - - - - - - - - - - -+ |
•          34     |             49h              | |
•                 +- - - - - - - - - - - - - - -+ |
•          35     |             42h              | |
•                 +- - - - - - - - - - - - - - -+  > KNXnet/IP MAC address
•          36     |             6Eh              | |
•                 +- - - - - - - - - - - - - - -+ |
•          37     |             65h              | |
•                 +- - - - - - - - - - - - - - -+ |
•          38     |             74h              | /
•                 +------------------------------+
•          39     |             'M'              | \
•                 +- - - - - - - - - - - - - - -+ |
•          40     |             'Y'              | |
•                 +- - - - - - - - - - - - - - -+ |
•          41     |             'H'              | |
•                 +- - - - - - - - - - - - - - -+  > Device Friendly Name, e.g. "MYHOME"
•          42     |             'O'              | |
•                 +- - - - - - - - - - - - - - -+ |
•          43     |             'M'              | |
•                 +- - - - - - - - - - - - - - -+ |
•          44     |             'E'              | |
•                 +- - - - - - - - - - - - - - -+ |
•          45     |             00h              | |
•                 +- - - - - - - - - - - - - - -+ |
•          ..     |             ...              | |
•                 +- - - - - - - - - - - - - - -+ |
•          68     |             00h              | /
•                 +------------------------------+
•          69     |             0Ah              |    structure length (DIB supported service
family)
•                 +------------------------------+
•          70     |             02h              |    description type code, 02h =
SUPP_SVC_FAMILIES
•                 +------------------------------+
•          71     |             02h              |    service family, e.g. 02h = KNXnet/IP
Core
•                 +------------------------------+
•          72     |             01h              |    service family version, e.g. 01h
•                 +------------------------------+
•          73     |             03h              |    service family, e.g. 03h = EIBnet/Device
Mgmt
•                 +------------------------------+
•          74     |             01h              |    service family version, e.g. 01h
•                 +------------------------------+
•          75     |             04h              |    service family, e.g. 04h = KNXnet/IP
Tunneling
•                 +------------------------------+
•          76     |             01h              |    service family version, e.g. 01h
•                 +------------------------------+
•          77     |             05h              |    service family, e.g. 05h = KNXnet/IP
Routing
•                 +------------------------------+
•          78     |             01h              |    service family version, e.g. 01h
•                 +------------------------------+
```

Figure B.2 — SEARCH_RESPONSE frame binary format: IP example *(2 of 2)*

B.3 DESCRIPTION_REQUEST

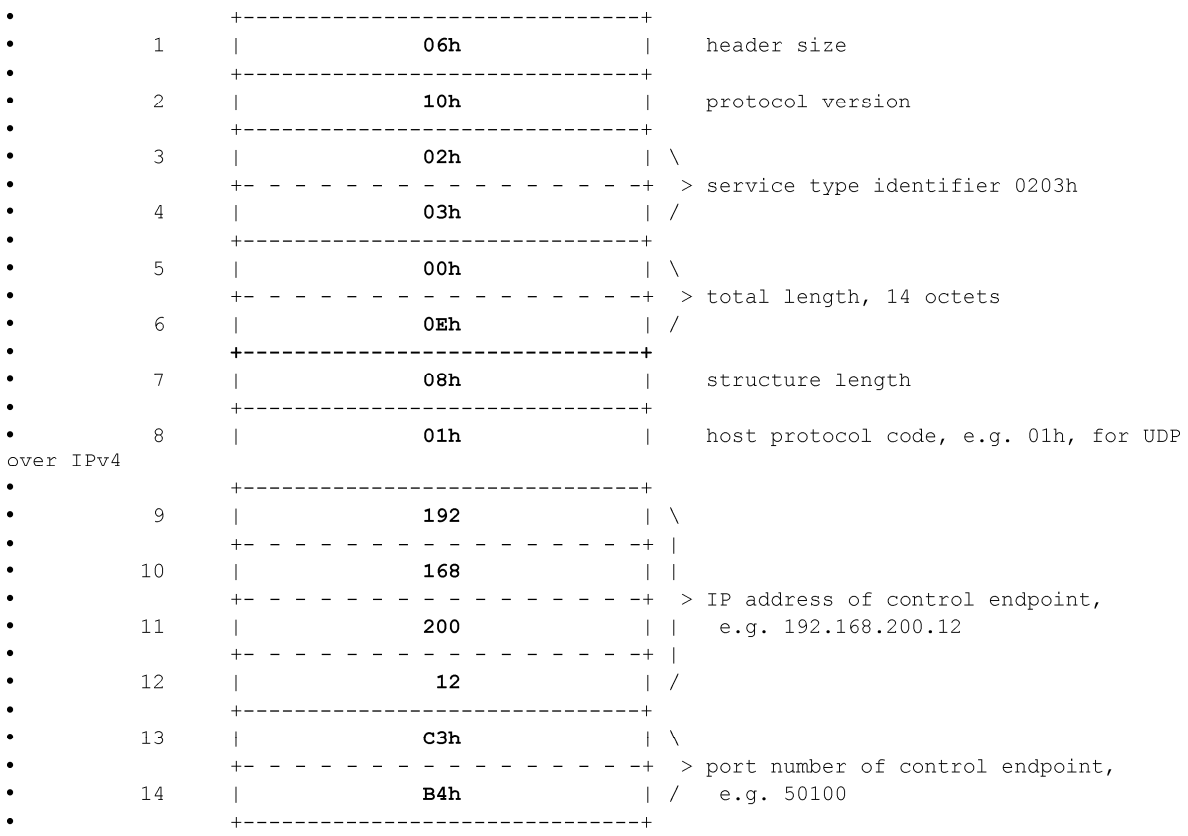


Figure B.3 — KNXnet/IP DESCRIPTION_REQUEST frame binary format example

B.4 DESCRIPTION_RESPONSE

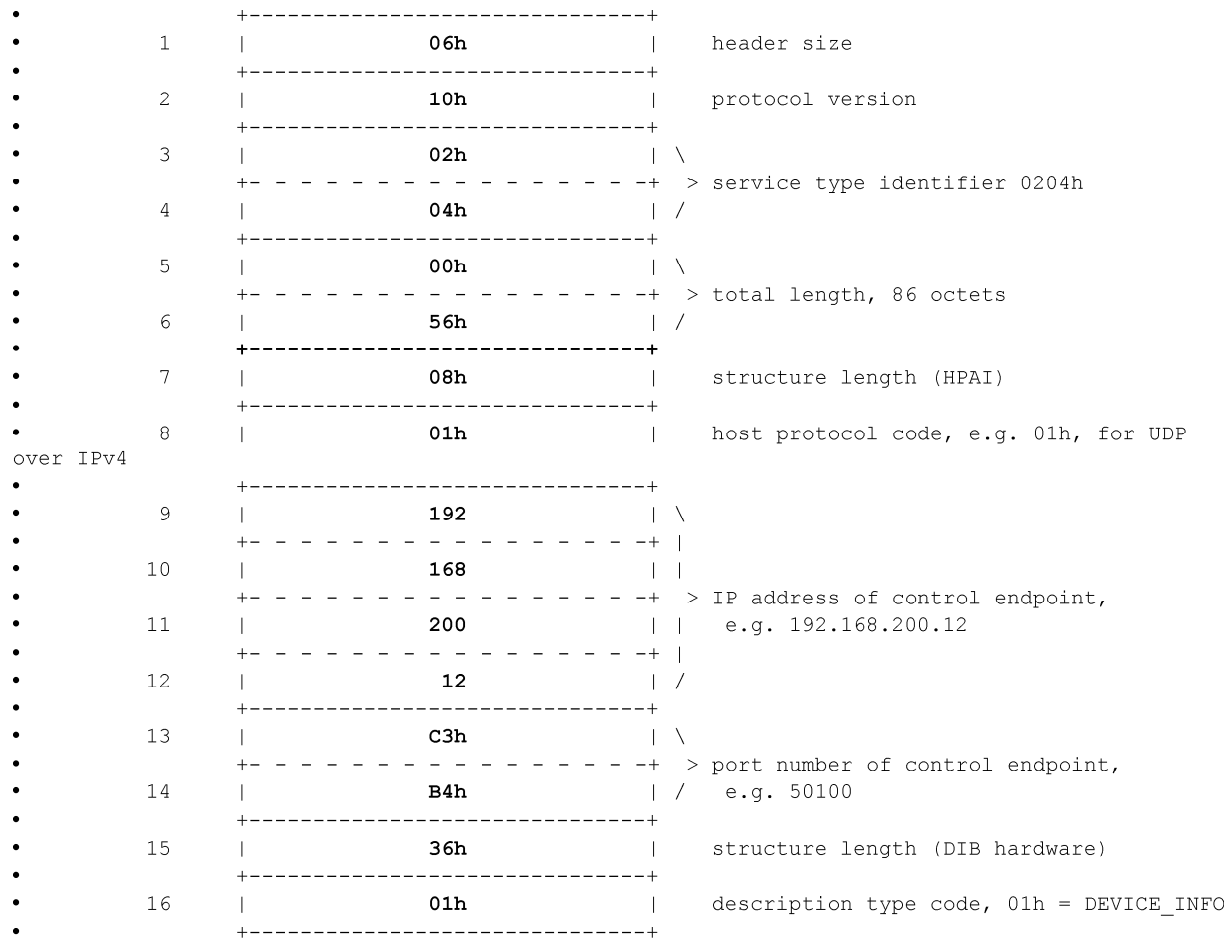


Figure B.4 *(1 of 3)*

```
•          17     |              02h              |    KNX medium, 02h = TP1 (EIB TP)
•                 +-------------------------------+
•          18     |              01h              |    Device Status, 01h = programming mode
•                 +-------------------------------+
•          19     |              11h              | \
•                 +- - - - - - - - - - - - - - - -+  > EIB physical address, e.g. 1.1.0
•          20     |              00h              | /
•                 +-------------------------------+
•          21     |              00h              | \
•                 +- - - - - - - - - - - - - - - -+  > Project-Installation ID, e.g. 0011h
•          22     |              11h              | /
•                 +-------------------------------+
•          23     |              00h              | \
•                 +- - - - - - - - - - - - - - - -+ |
•          24     |              01h              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          25     |              11h              | |
•                 +- - - - - - - - - - - - - - - -+  > KNX device serial number,
•          26     |              11h              | |            includes manufacturer ID (2
octets)
•                 +- - - - - - - - - - - - - - - -+ |
•          27     |              11h              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          28     |              11h              | /
•                 +-------------------------------+
•          29     |              224              | \
•                 +- - - - - - - - - - - - - - - -+ |
•          30     |                0              | |
•                 +- - - - - - - - - - - - - - - -+  > device routing multicast address
•          31     |               23              | |   e.g. 224.0.23.12
•                 +- - - - - - - - - - - - - - - -+ |
•          32     |               12              | /
•                 +-------------------------------+
•          33     |              45h              | \
•                 +- - - - - - - - - - - - - - - -+ |
•          34     |              49h              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          35     |              42h              | |
•                 +- - - - - - - - - - - - - - - -+  > KNXnet/IP MAC address
•          36     |              6Eh              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          37     |              65h              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          38     |              74h              | /
•                 +-------------------------------+
•          39     |              'M'              | \
•                 +- - - - - - - - - - - - - - - -+ |
•          40     |              'Y'              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          41     |              'H'              | |
•                 +- - - - - - - - - - - - - - - -+  > Device Friendly Name, e.g. "MYHOME"
•          42     |              'O'              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          43     |              'M'              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          44     |              'E'              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          45     |              00h              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          ..     |              ...              | |
•                 +- - - - - - - - - - - - - - - -+ |
•          68     |              00h              | /
•                 +-------------------------------+
•          69     |              0Ah              |    structure length (DIB supported service
family)
•                 +-------------------------------+
•          70     |              02h              |    description type code, 02h =
SUPP_SVC_FAMILIES
```

Figure B.4 *(2 of 3)*

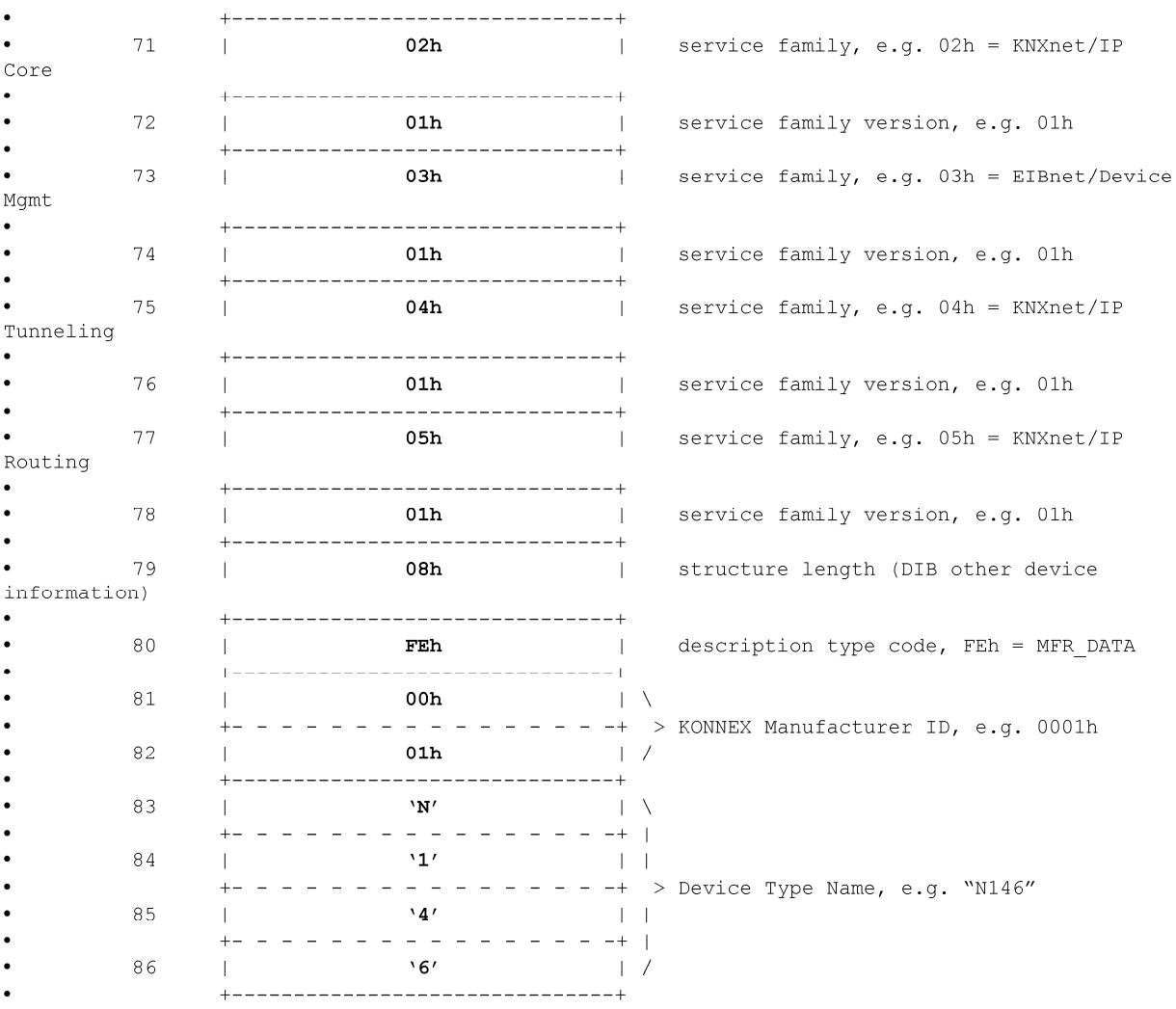


Figure B.4 — DESCRIPTION_RESPONSE frame binary format: IP example *(3 of 3)*

B.5 CONNECT_REQUEST

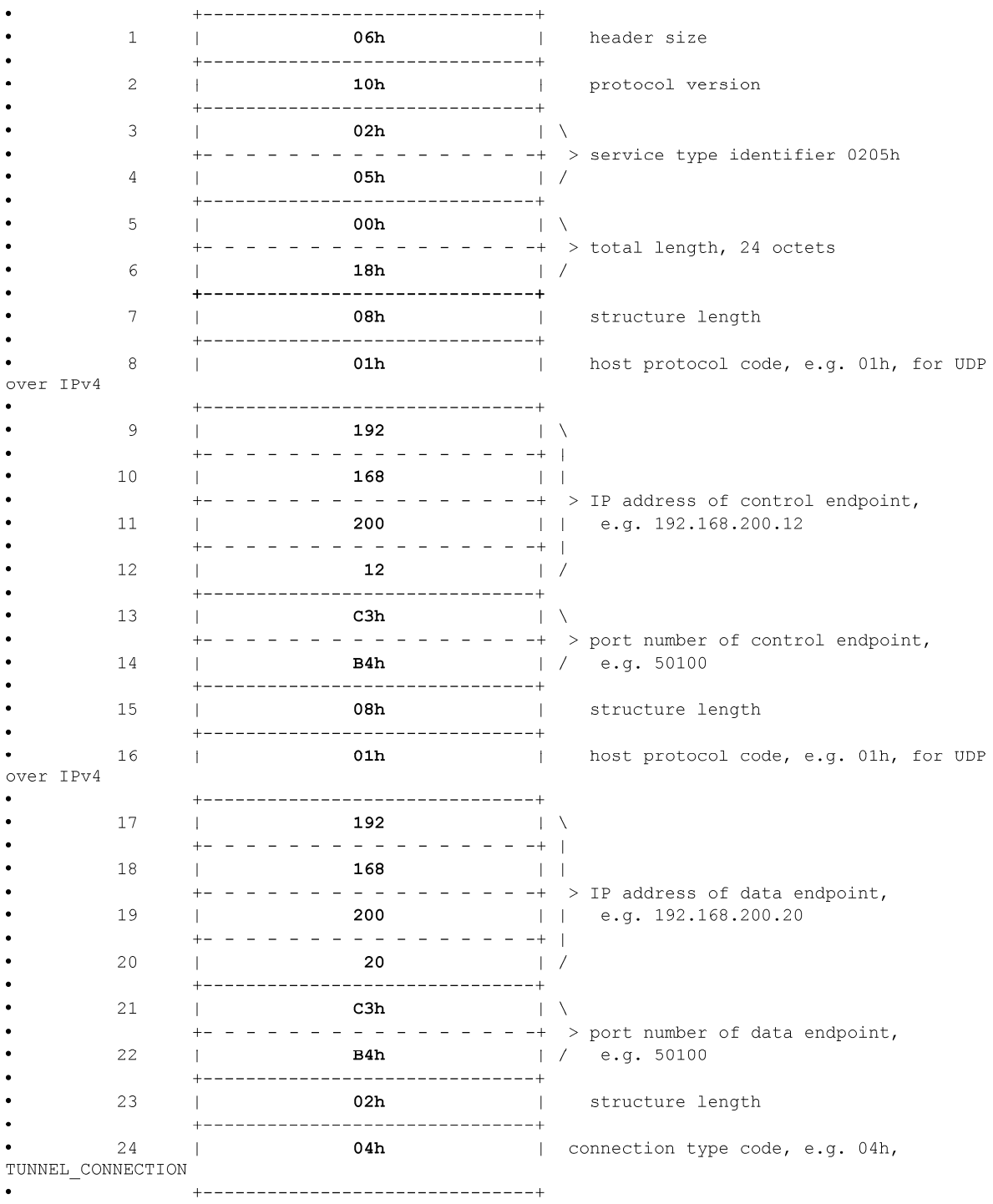


Figure B.5 — KNXnet/IP CONNECT_REQUEST frame binary format example

B.6 CONNECT_RESPONSE

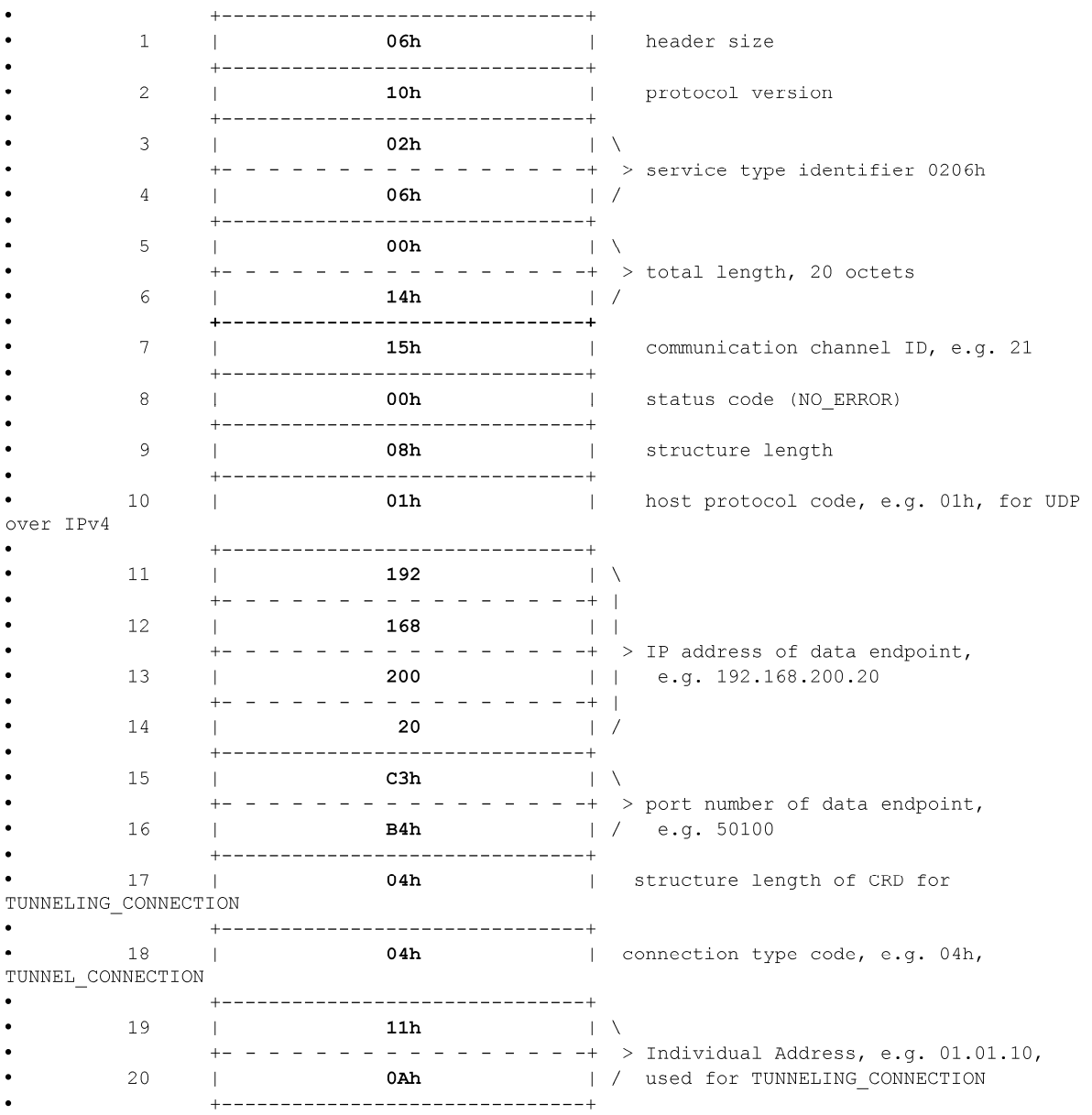


Figure B.6 — CONNECT_RESPONSE frame binary format: IP example

B.7 CONNECTIONSTATE_REQUEST

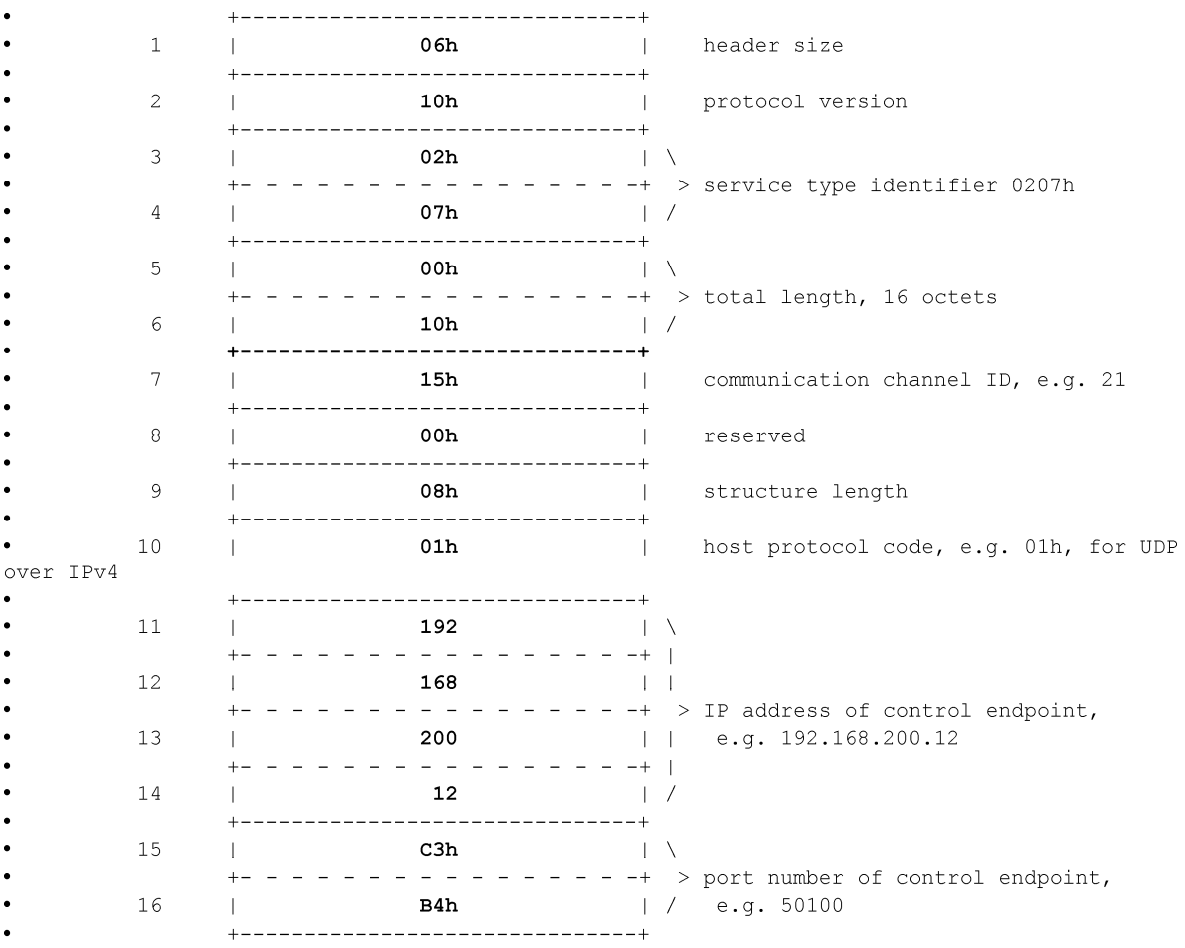


Figure B.7 — CONNECTIONSTATE_REQUEST frame binary format: IP example

B.8 CONNECTIONSTATE_RESPONSE

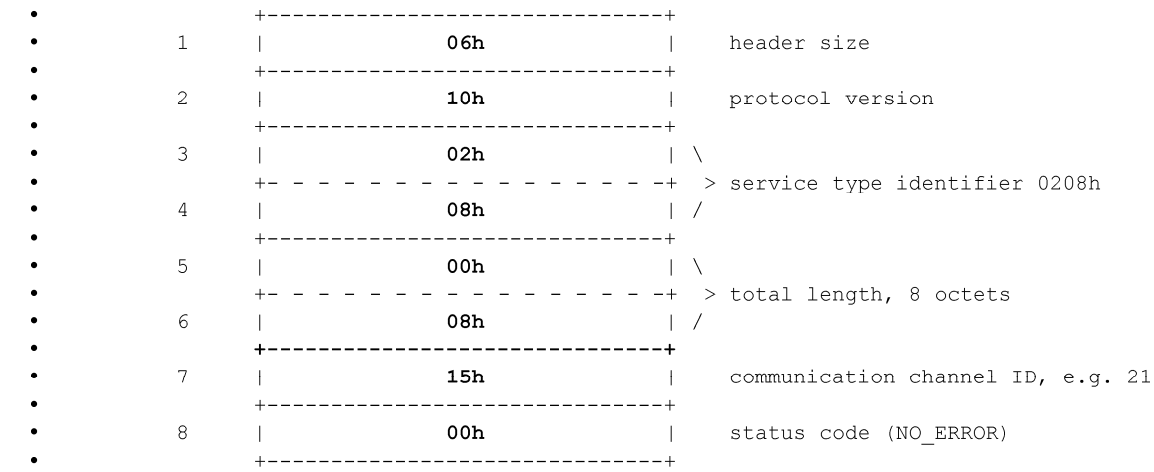


Figure B.8 — CONNECTIONSTATE_RESPONSE frame binary format: IP example

B.9 DISCONNECT_REQUEST

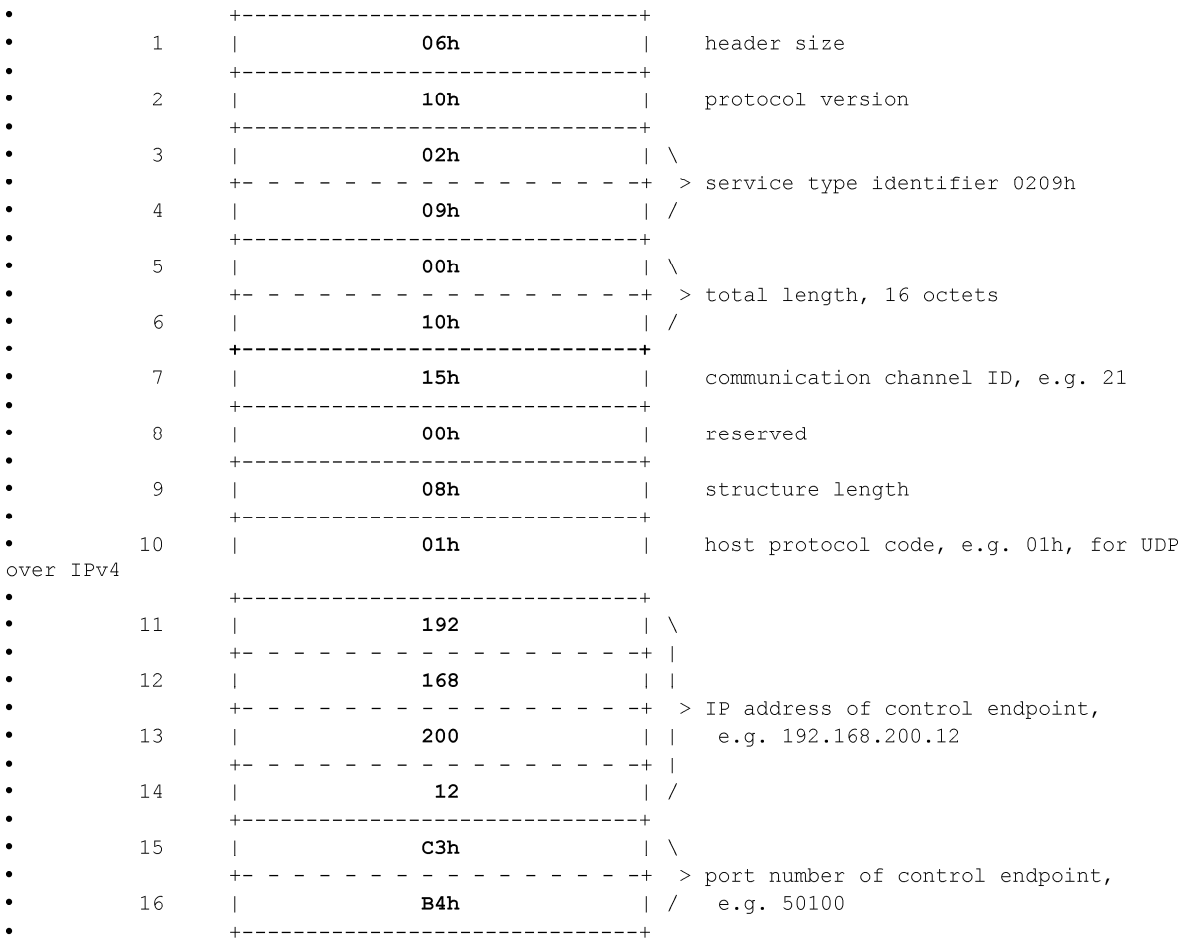


Figure B.9 — DISCONNECT_REQUEST frame binary format: IP example

B.10 DISCONNECT_RESPONSE

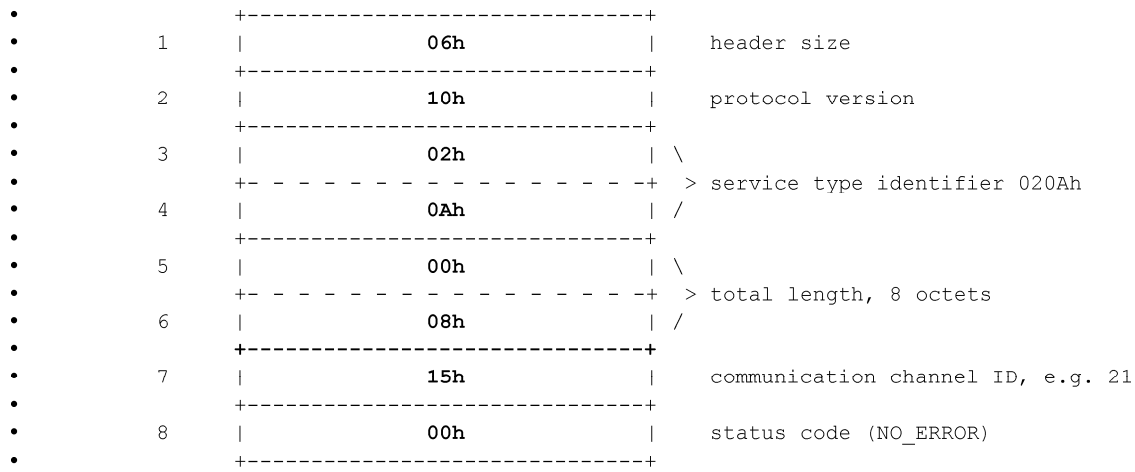


Figure B.10 — DISCONNECT_RESPONSE frame binary format: IP example

B.11 DEVICE_CONFIGURATION_REQUEST

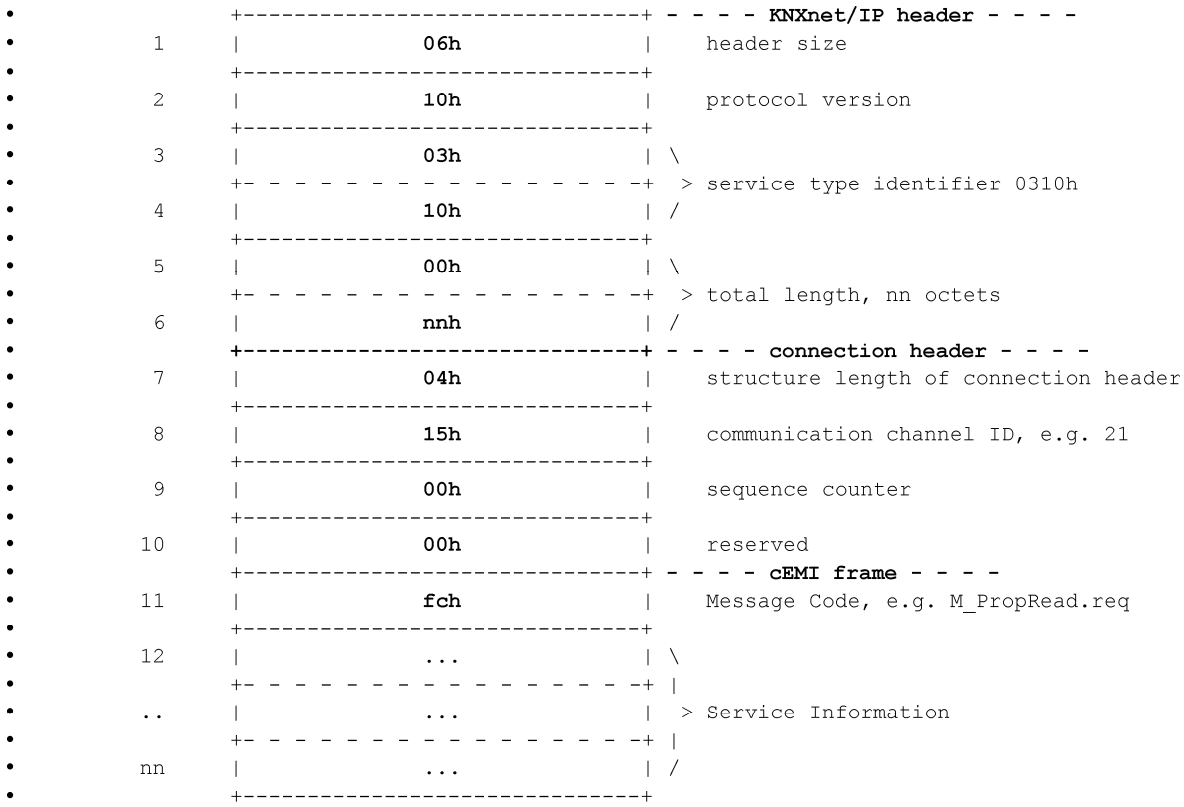


Figure B.11 — DEVICE_CONFIGURATION_REQUEST frame binary format: example

B.12 DEVICE_CONFIGURATION_ACK

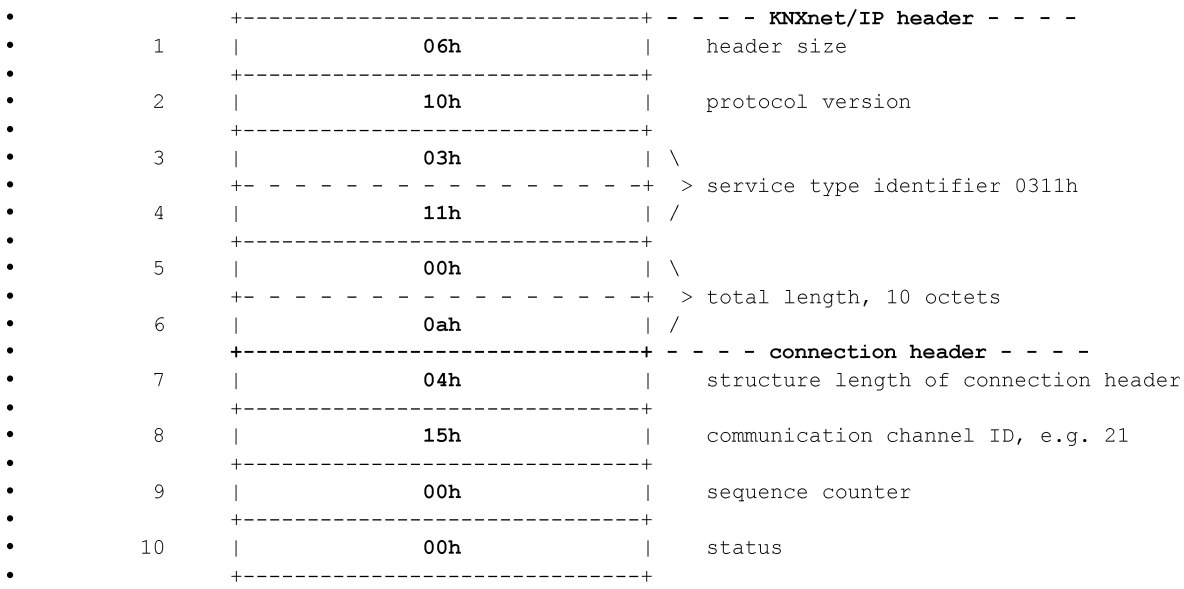


Figure B.12 — DEVICE_CONFIGURATION_ACK frame binary format: example

B.13 TUNNELLING_REQUEST

	06h		**KNXnet/IP header**
1	06h		header size
2	10h		protocol version
3	04h	\	
4	20h	/	service type identifier 0420h
5	00h	\	
6	L+0Ch	/	total length, L+12 octets
			connection header
7	06h		structure length of connection header
8	15h		communication channel ID, e.g. 21
9	00h		sequence counter
10	00h		reserved
			cEMI frame
11	11h		message code (e.g. L_Data.req message)
12	00h		additional information (none)
13	...	\	
14	...	|	Service Information (L bytes)
L+12	...	/	

Figure B.13 — TUNNELLING_REQUEST frame binary format: example

B.14 TUNNELLING_ACK

	06h		**KNXnet/IP header**
1	06h		header size
2	10h		protocol version
3	04h	\	
4	21h	/	service type identifier 0421h
5	00h	\	
6	0Ah	/	total length, 10 octets
			connection header
7	04h		structure length of connection header
8	15h		communication channel ID, e.g. 21
9	00h		sequence counter
10	00h		status, e.g. 00h (NO_ERROR)

Figure B.14 — TUNNELLING_ACK frame binary format: example

B.15 ROUTING_INDICATION

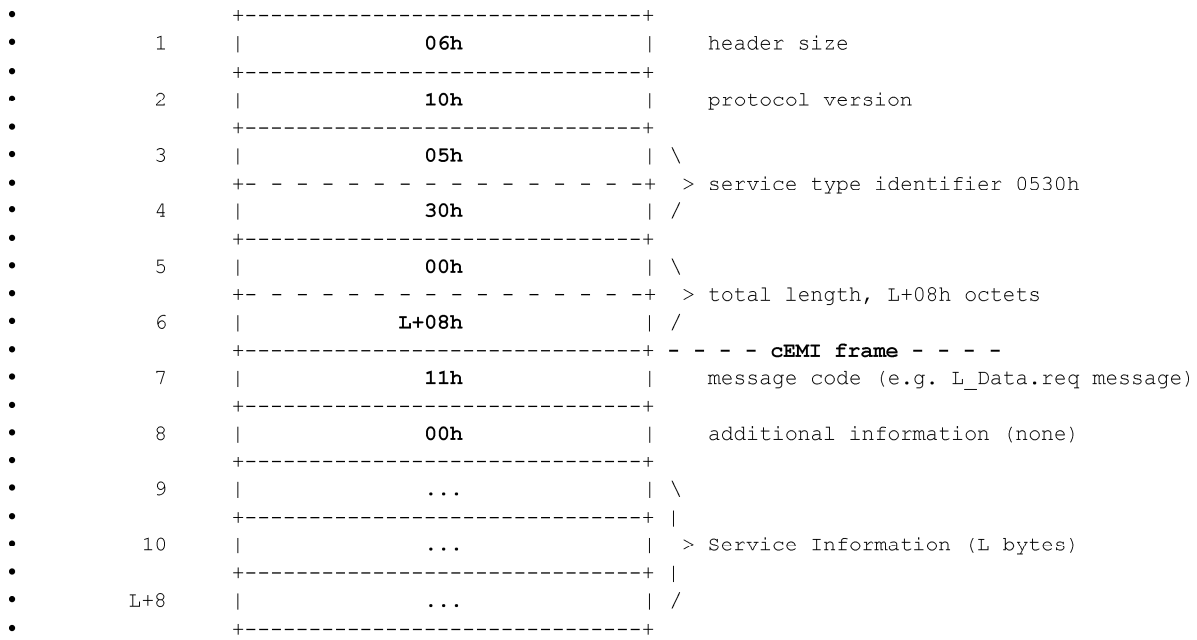


Figure B.15 — ROUTING_INDICATION frame binary format: example

B.16 ROUTING_LOST_MESSAGE

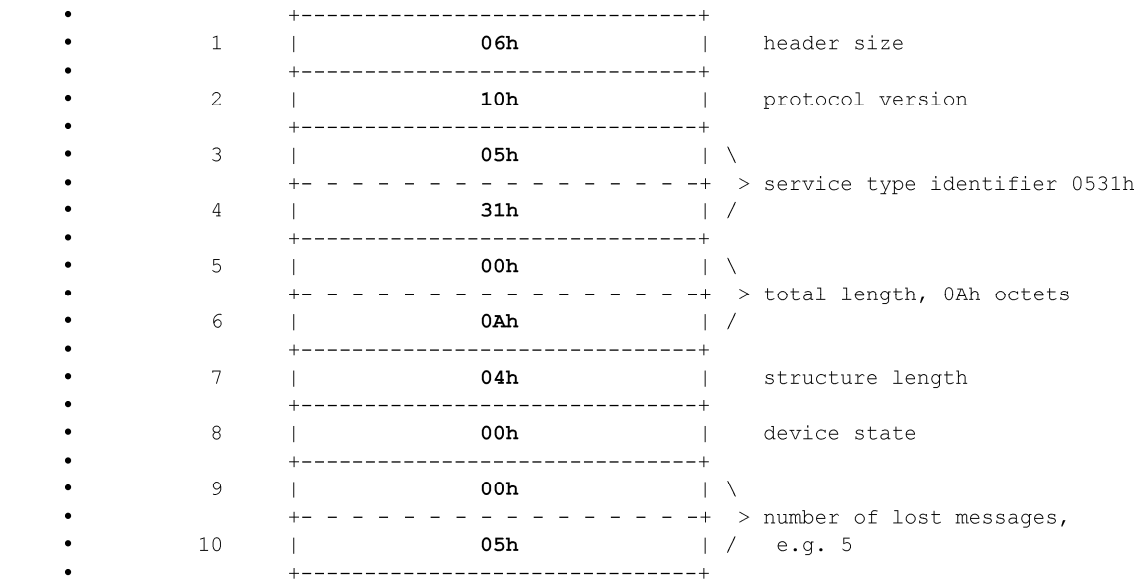


Figure B.16 — ROUTING_LOST_MESSAGE frame binary format: example

B.17 ROUTING_BUSY

```
          +-------------------------------+
 1        |             06h               |    header size
          +-------------------------------+
 2        |             10h               |    protocol version
          +-------------------------------+
 3        |             05h               | \
          +- - - - - - - - - - - - - - - -+  > service type identifier 0532h
 4        |             32h               | /
          +-------------------------------+
 5        |             00h               | \
          +- - - - - - - - - - - - - - - -+  > total length, 0Ch octets
 6        |             0Ch               | /
          +-------------------------------+
 7        |             04h               |    structure length
          +-------------------------------+
 8        |             00h               |    device state
          +-------------------------------+
 9        |             00h               | \
          +- - - - - - - - - - - - - - - -+  > number of milliseconds to wait,
10        |             64h               | /   e.g. 100 ms
          +-------------------------------+
11        |             00h               | \
          +- - - - - - - - - - - - - - - -+  > routing busy control value
12        |             00h               | /  (default value: 0000h)
          +-------------------------------+
```

Figure B.17 — ROUTING_BUSY frame binary format: example

B.18 REMOTE_DIAGNOSTIC_REQUEST

```
•                +-------------------------------+ - - - - KNXnet/IP header - - - -
•          1     |             06h               |    header size
•                +-------------------------------+
•          2     |             10h               |    protocol version
•                +-------------------------------+
•          3     |             07h               | \
•                +- - - - - - - - - - - - - - - -+  > service type identifier 0740h
•          4     |             40h               | /
•                +-------------------------------+
•          5     |             00h               | \
•                +- - - - - - - - - - - - - - - -+  > total length, 16 octets
•          6     |             10h               | /
•                +-------------------------------+ - - - - HPAI - - - -
•          7     |             08h               |    structure length of HPAI
•                +-------------------------------+
•          8     |             01h               |    host protocol code, e.g. 01h, for UDP
over IPv4
•                +-------------------------------+
•          9     |             E0h               | \
•                +- - - - - - - - - - - - - - - -+ |
•         10     |             00h               | |
•                +- - - - - - - - - - - - - - - -+  > IP multicast address
•         11     |             17h               | |   e.g. 224.0.23.12
•                +- - - - - - - - - - - - - - - -+ |   (System Routing Multicast Address)
•         12     |             0Ch               | /
•                +-------------------------------+
•         13     |             0Eh               | \
•                +- - - - - - - - - - - - - - - -+  > port number of control endpoint, 3671
•         14     |             57h               | /
•                +-------------------------------+ - - - - SELECTOR - - - -
•         15     |             02h               |    structure length of SELECTOR
•                +-------------------------------+
•         16     |             01h               |    Programming Mode Selector
•                +-------------------------------+
•
```

Figure B.18 — REMOTE_DIAGNOSTIC_REQUEST frame binary format: example

B.19 REMOTE_DIAGNOSTIC_RESPONSE

		- - - - KNXnet/IP header - - - -	
1	**06h**	header size	
2	**10h**	protocol version	
3	**07h**		\
4	**41h**		/ > service type identifier 0741h
5	**00h**		\
6	**3Ah**		/ > total length, 58 octets
		- - - - SELECTOR - - - -	
7	**02h**	structure length of SELECTOR	
8	**01h**	ProgrammProgramming Mode Selector	
		- - - - DIB IP Config - - - -	
9	**10h**	structure length of DIB IP Config	
10	**03h**	Description Type Code	
11	**C0h**		\
12	**A8h**		|
13	**02h**		| > IP address e.g. 192.168.2.12
14	**0Ch**		/
15	**FFh**		\
16	**FFh**		|
17	**FFh**		| > subnet mask e.g. 255.255.255.0
18	**00h**		/
19	**C0h**		\
20	**A8h**		|
21	**02h**		| > default gateway IP address e.g. 192.168.2.1
22	**01h**		/
23	**02h**	IP capabilities (e.g. DHCP)	
24	**01h**	IP assignment method (e.g. manually)	
		- - - - DIB IP Current Config - - - -	
25	**14h**	structure length of DIB IP Current Config	
26	**04h**	Description Type Code	
27	**C0h**		\
28	**A8h**		|
29	**02h**		| > IP address e.g. 192.168.2.12
30	**0Ch**		/
31	**FFh**		\
32	**FFh**		|
33	**FFh**		| > subnet mask e.g. 255.255.255.0

Figure B.19 *(1 of 2)*

```
•          34     |              00h              | /
•                 +-------------------------------+
•          35     |              C0h              | \
•                 +- - - - - - - - - - - - - - - -+ |
•          36     |              A8h              | |
•                 +- - - - - - - - - - - - - - - -+  > default gateway IP address
•          37     |              02h              | |   e.g. 192.168.2.1
•                 +- - - - - - - - - - - - - - - -+ |
•          38     |              01h              | /
•                 +-------------------------------+
•          39     |              C0h              | \
•                 +- - - - - - - - - - - - - - - -+ |
•          40     |              A8h              | |
•                 +- - - - - - - - - - - - - - - -+  > DHCP server IP address
•          41     |              02h              | |   e.g. 192.168.2.1
•                 +- - - - - - - - - - - - - - - -+ |
•          42     |              01h              | /
•                 +-------------------------------+
•          43     |              04h              |    Current IP assignment method (e.g. DHCP)
•                 +-------------------------------+
•          44     |              00h              |    reserved
•                 +-------------------------------+ - - - - DIB KNX Addresses - - - -
•          45     |              0Eh              |    structure length of DIB KNX Addresses
•                 +-------------------------------+
•          46     |              05h              |    Description Type Code
•                 +-------------------------------+
•          47     |              11h              | \
•                 +- - - - - - - - - - - - - - - -+  > KNX individual address (e.g. 1.1.0)
•          48     |              00h              | /
•                 +-------------------------------+
•          49     |              11h              | \
•                 +- - - - - - - - - - - - - - - -+  > Additional individual address (e.g.
1.1.255)
•          50     |              FFh              | /
•                 +-------------------------------+
•          51     |              11h              | \
•                 +- - - - - - - - - - - - - - - -+  > Additional individual address (e.g.
1.1.254)
•          52     |              FEh              | /
•                 +-------------------------------+
•          53     |              11h              | \
•                 +- - - - - - - - - - - - - - - -+  > Additional individual address (e.g.
1.1.200)
•          54     |              C8h              | /
•                 +-------------------------------+
•          55     |              11h              | \
•                 +- - - - - - - - - - - - - - - -+  > Additional individual address (e.g.
1.1.199)
•          56     |              C7h              | /
•                 +-------------------------------+
•          57     |              11h              | \
•                 +- - - - - - - - - - - - - - - -+  > Additional individual address (e.g.
1.1.150)
•          58     |              96h              | /
•                 +-------------------------------+
```

Figure B.19 — REMOTE_DIAGNOSTIC_RESPONSE frame binary format: example *(2 of 2)*

B.20 REMOTE_BASIC_CONFIGURATION_REQUEST

```
•                +-------------------------------+ - - - - KNXnet/IP header - - - -
•          1     |              06h              |    header size
•                +-------------------------------+
•          2     |              10h              |    protocol version
•                +-------------------------------+
•          3     |              07h              | \
•                +- - - - - - - - - - - - - - - -+  > service type identifier 0742h
•          4     |              42h              | /
•                +-------------------------------+
•          5     |              00h              | \
•                +- - - - - - - - - - - - - - - -+  > total length, 32 octets
•          6     |              20h              | /
•                +-------------------------------+ - - - - HPAI - - - -
•          7     |              08h              |    structure length of HPAI
•                +-------------------------------+
•          8     |              01h              |    host protocol code, e.g. 01h, for UDP
over IPv4
•                +-------------------------------+
•          9     |              E0h              | \
•                +- - - - - - - - - - - - - - - -+ |
•         10     |              00h              | |
•                +- - - - - - - - - - - - - - - -+  > IP multicast address
•         11     |              17h              | |   e.g. 224.0.23.12
•                +- - - - - - - - - - - - - - - -+ |   (System Routing Multicast Address)
•         12     |              0Ch              | /
•                +-------------------------------+
•         13     |              0Eh              | \
•                +- - - - - - - - - - - - - - - -+  > port number of control endpoint, 3671
•         14     |              57h              | /
•                +-------------------------------+ - - - - SELECTOR - - - -
•         15     |              02h              |    structure length of SELECTOR
•                +-------------------------------+
•         16     |              01h              |    ProgrammProgramming Mode Selector
•                +-------------------------------+ - - - - DIB IP Config - - - -
•         17     |              10h              |    structure length of DIB IP Config
•                +-------------------------------+
•         18     |              03h              |    Description Type Code
•                +-------------------------------+
•         19     |              C0h              | \
•                +- - - - - - - - - - - - - - - -+ |
•         20     |              A8h              | |
•                +- - - - - - - - - - - - - - - -+  > IP address
•         21     |              03h              | |   e.g. 192.168.3.12
•                +- - - - - - - - - - - - - - - -+ |
•         22     |              0Ch              | /
•                +-------------------------------+
•         23     |              FFh              | \
•                +- - - - - - - - - - - - - - - -+ |
•         24     |              FFh              | |
•                +- - - - - - - - - - - - - - - -+  > subnet mask
•         25     |              FFh              | |   e.g. 255.255.255.0
•                +- - - - - - - - - - - - - - - -+ |
•         26     |              00h              | /
•                +-------------------------------+
•         27     |              C0h              | \
•                +- - - - - - - - - - - - - - - -+ |
•         28     |              A8h              | |
•                +- - - - - - - - - - - - - - - -+  > default gateway IP address
•         29     |              03h              | |   e.g. 192.168.3.1
•                +- - - - - - - - - - - - - - - -+ |
•         30     |              01h              | /
•                +-------------------------------+
•         31     |              00h              |    IP capabilities (not writable ↛ 00h)
•                +-------------------------------+
•         32     |              01h              |    IP assignment method (e.g. manually)
•                +-------------------------------+
```

Figure B.20 — REMOTE_BASIC_CONFIGURATION_REQUEST frame binary format: example

B.21 REMOTE_RESET_REQUEST

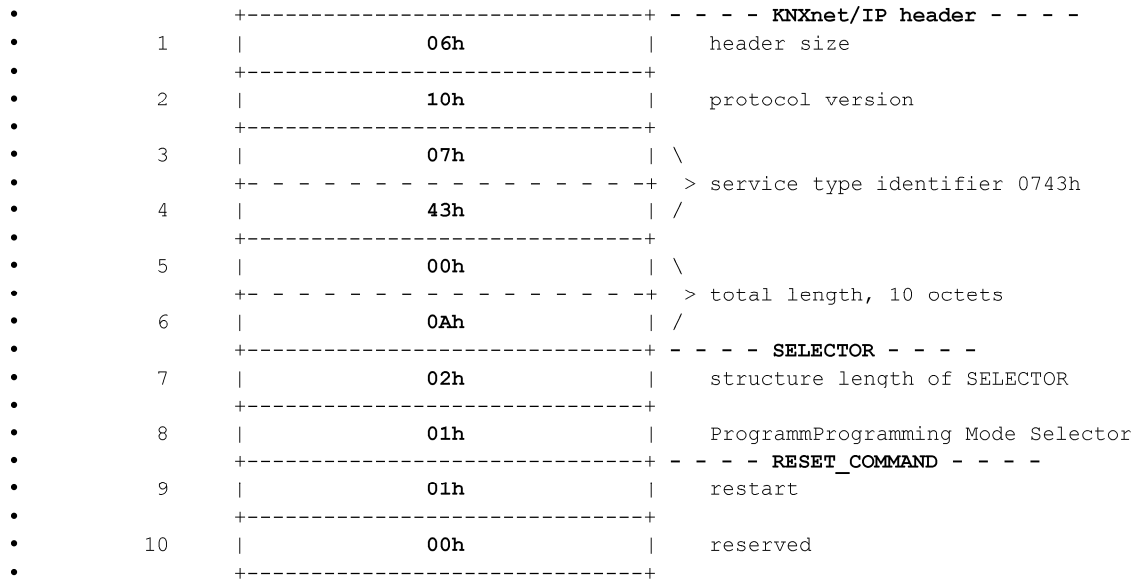


Figure B.21 — REMOTE_RESET_REQUEST frame binary format: example

Annex C
(normative)

KNXnet/IP Parameter Object

Table C.1 — Property Identifiers

PID	Property_name	Write/ Read
51	PID_PROJECT_INSTALLATION_ID	w/r
52	PID_KNX_INDIVIDUAL_ADDRESS	w/r
53	PID_ADDITIONAL_INDIVIDUAL_ADDRESSES	w/r
54	PID_CURRENT_IP_ASSIGNMENT_METHOD	-/r
55	PID_IP_ASSIGNMENT_METHOD	w/r
56	PID_IP_CAPABILITIES	-/r
57	PID_CURRENT_IP_ADDRESS	-/r
58	PID_CURRENT_SUBNET_MASK	-/r
59	PID_CURRENT_DEFAULT_GATEWAY	-/r
60	PID_IP_ADDRESS	w/r
61	PID_SUBNET_MASK	w/r
62	PID_DEFAULT_GATEWAY	w/r
63	PID_DHCP_BOOTP_SERVER	-/r
64	PID_MAC_ADDRESS	-/r
65	PID_SYSTEM_SETUP_MULTICAST_ADDRESS	-/r
66	PID_ROUTING_MULTICAST_ADDRESS	w/r
67	PID_TTL	w/r
68	PID_KNXNETIP_DEVICE_CAPABILITIES	-/r
69	PID_KNXNETIP_DEVICE_STATE	-/r
70	PID_KNXNETIP_ROUTING_CAPABILITIES	-/r
71	PID_PRIORITY_FIFO_ENABLED	w/r
72	PID_QUEUE_OVERFLOW_TO_IP	-/r
73	PID_QUEUE_OVERFLOW_TO_KNX	-/r
74	PID_MSG_TRANSMIT_TO_IP	-/r
75	PID_MSG_TRANSMIT_TO_KNX	-/r
76	PID_FRIENDLY_NAME	w/r
78	PID_ROUTING_BUSY_WAIT_TIME	-/r

Table C.2 — KNXnet/IP Parameter Object Properties

Property Name	Property ID	M/O	Comment
Interface Object Type	PID_OBJECT_TYPE	M	IP Parameter Object 11
Interface Object Name	PID_OBJECT_NAME	O	Name of the parameter setting
Project Installation Identification	PID_PROJECT_INSTALLATION_ID	M	Set by ETS only!
KNX Individual Address	PID_KNX_INDIVIDUAL_ADDRESS	M	Mandatory for devices implementing KNXnet/IP
Additional Individual Addresses	PID_ADDITIONAL_INDIVIDUAL_ADDRESSES	M	Mandatory for devices implementing KNXnet/IP Tunnelling and Routing
Current IP Assignment Method	PID_CURRENT_IP_ASSIGNMENT_METHOD	O	1: manually, 2: BootP, 4:DHCP, 8:AutoIP
IP Assignment Method	PID_IP_ASSIGNMENT_METHOD	M	1: manually, 2: BootP, 4:DHCP, 8:AutoIP
IP Capabilities	PID_IP_CAPABILITIES	O	shows the IP capabilities of the device
Current IP Address	PID_CURRENT_IP_ADDRESS	M	this is the currently used IP address
Current Subnet Mask	PID_CURRENT_SUBNET_MASK	M	this is the currently used subnet mask
Current Default Gateway	PID_CURRENT_DEFAULT_GATEWAY	M	this is the currently used default gateway
IP Address	PID_IP_ADDRESS	M	used for manual address assignment
Subnet Mask	PID_SUBNET_MASK	M	Subnet mask
Default Gateway	PID_DEFAULT_GATEWAY	M	Default gateway
DHCP/BootP Server	PID_DHCP_BOOTP_SERVER	O	IP address of last DHCP/BootP server
MAC Address	PID_MAC_ADDRESS	M	MAC address
System Setup Multicast Address	PID_SYSTEM_SETUP_MULTICAST_ADDRESS	M	default KNXnet/IP system multicast address used for KNXnet/IP Core services
Routing Multicast Address	PID_ROUTING_MULTICAST_ADDRESS	M	second multicast address used for KNXnet/IP routing
Time To Live	PID_TTL	M	TTL for IP routing

Property Name	Property ID	M/O	Comment
(TTL)			
KNXnet/IP Device Capabilities	PID_KNXNETIP_DEVICE_CAPABILITIES	M	description of KNXnet/IP device capabilities
KNXnet/IP Device State	PID_KNXNETIP_DEVICE_STATE	M	KNXnet/IP device status information
KNXnet/IP Routing Capabilities	PID_KNXNETIP_ROUTING_CAPABILITIES	O[a]	description of KNXnet/IP routing capabilities
Priority FIFO Enabled	PID_PRIORITY_FIFO_ENABLED	O	Read, set or reset if priority FIFO is enabled
Queue Overflow to IP	PID_QUEUE_OVERFLOW_TO_IP	O[a]	Number of telegrams lost because queue to IP overflowed
Queue Overflow to KNX	PID_QUEUE_OVERFLOW_TO_KNX	O[a]	Number of telegrams lost because queue to KNX overflowed
Telegrams Transmitted to IP	PID_MSG_TRANSMIT_TO_IP	O[a]	Number of telegrams successfully transmitted to IP
Telegrams Transmitted to KNX	PID_MSG_TRANSMIT_TO_KNX	O[a]	Number of telegrams successfully transmitted to KNX
Friendly Name	PID_FRIENDLY_NAME	M	Mandatory for devices implementing KNXnet/IP

[a] This property shall be implemented by KNXnet/IP devices providing KNXnet/IP Routing.

Annex D
(normative)

Common External Messaging Interface (cEMI)

D.1 cEMI

D.1.1 cEMI: message format and services

D.1.1.1 Introduction

The cEMI message format is a generic structure for medium independent KNX messages, which can be added with information like a timestamp or other. The cEMI message format claims to be independent from the frame structures of the different KNX media. Respectively, it claims to make possible transportation of all information of all the different KNX (medium dependent) frame formats.

D.1.1.2 Message flow – overview

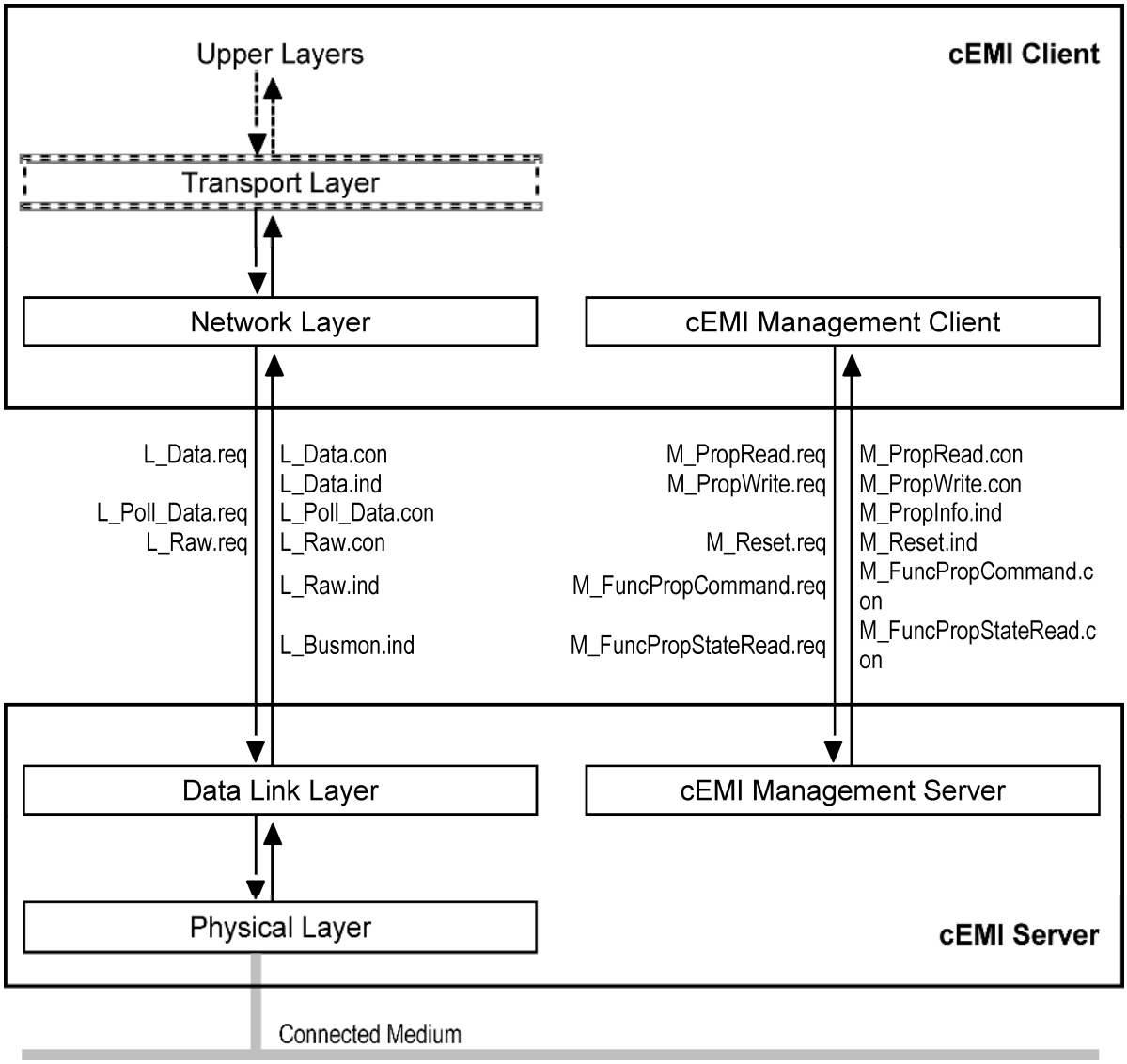


Figure D.1 — Message flow between cEMI client and cEMI Server

D.1.1.3 Message Code and Message Code Set

D.1.1.3.1 Definitions

Each cEMI message shall start with the Message Code octet.

The cEMI Message Code Set covers all different message types as for Busmonitor communication mode, link & Transport Layer communication and for local device management communication.

D.1.1.3.2 Exception handling: unknown messages

General behaviour after reception of an unknown or unsupported message (i.e., the cEMI Server does not know the received cEMI message code): the received message shall be ignored, i.e, the cEMI Server shall generate no confirmation message.

D.1.1.4 Basic message structure

D.1.1.4.1 Generic message structure

A cEMI message shall have the following generic structure:

Table D.1

Message Code	Additional Info Length	Additional Information	Service Information
MC	**AddIL**	**…**	**…**
1 octet	1 octet	var. length	var. length

Services for management of the local device do not need any additional information. For this reason, the Additional Information field and the Additional Info Length (AddIL) field are not present in local device management services.

A cEMI management message shall have the following generic structure:

Table D.2

Message Code	Service Information
MC	**…**
1 octet	var. length

D.1.1.4.2 cEMI length information

A cEMI message shall be encapsulated in a message or frame structure of a host protocol. Currently, known cEMI host protocols (KNX on USB, KNXnet/IP) represent the number of octets of the cEMI frame by a length information field within a header data structure (of the host protocol). The cEMI frame itself is typically the data field (or “body”), or a part of the data field within the host protocols frame structure.

D.1.1.4.3 Additional information

D.1.1.4.3.1 Overview

The Additional Information field is intended for:

— medium dependent information, and

— other information: e.g. (relative) timestamp & error flags Busmonitor function.

The Additional Information field shall contain tagged information, identified by the Type ID:

Table D.3

Additional Information Type	Type ID	Length of Information	Information	Data Direction
	00h		reserved	
PL medium information	01h	2 octets	Domain Address used by PL medium	Client ↔ Server
RF medium information	02h	8 octets	RF-Info byte (formerly named RF-Ctrl) and KNX Serial Number/DoA and Data Link Layer Frame Number (LFN)	Client ↔ Server
Busmonitor – Status Info	03h	1 octet	Busmonitor Error Flags	Client ← Server
Timestamp relative	04h	2 octets	Relative timestamp; e.g. for L_Raw.ind	Client ← Server
Time delay until sending	05h	4 octets	Time delay (L_Raw.req)	Client → Server
Extended relative timestamp	06h	4 octets	Device independent time stamp, e.g. for L_Raw.ind or L_Busmon.ind	Client ← Server
BiBat information	07h	2 octets	Contains b7-b4 of the RF KNX-Ctrl field and BiBat Block-number	Client ↔ Server
…	08h	…	not used	
	…	…		
…	FEh	…	not used	
reserved	FFh	…	for future system extension (ESC Code)	

The structure of the Additional Information field is based on the approach to be open for future extensions; e.g. new types of Additional Information.

Combinations (concatenation) of Additional Information Types can be used in the additional information field.

Each Additional Information Type shall be accompanied by a length information giving the data length (number of octets) of the information type itself.

The Additional Information Length field shall be the number of octets of all additional information including the Type IDs, which shall be 1 octet long, and the type data length field(s), which shall also be each 1 octet long.

If no Additional Information field is included in a cEMI message, then the value representing the Additional Information Length shall be set to zero. Value 255 is reserved for future extension.

Table D.4 — Message structure including Additional Information

Message Code	Additional Info Length	Additional Information								Service Information
MC	**AddIL**	**Type ID**	**Len**	**information**	**Type ID**	**Len**	**information**	**…**	**…**	**….**
1 octet	1 octet	1 octet	1 octet	dep. on inf. type	1 octet	1 octet	dep. on inf. type	…	…	var. length

AddIL (= number of octets of Additional Information)

Concatenated Additional Information fields shall be sorted in ascending order of their Type IDs.

A receiver of a cEMI message including additional information but not interested on all or part of the information can ignore the whole or part of the additional information.

The different length information fields are intended to simplify evaluating the frame, e.g. to easily find the start of the Service Information field or to faster find the beginning of next Additional Information field.

D.1.1.4.3.2 AddInfo-Type 06h: Extended relative timestamp

A cEMI Server can use the "Timestamp relative" (Additional Information Type ID: 04h) to deliver a timestamp in cEMI messages, typically used for bus monitoring (L_Busmon.ind / L_Raw.ind). However, this implementation of timestamp information has some drawbacks:

a) The timestamp information is given in "ticks" within 2 octets, providing a range of values from 1 to 65 535. Typically, the internal clock rate of modern cEMI devices is about 1 MHz to 10 MHz, and therefore a "tick" is usually about a few µs, sometimes even lower. This causes a counter overflow after a very short time (in the ms range).

   On the tool side (e.g. Busmonitor software on PC), this counter overflow has to be taken care of, resulting in complicated timing-procedures to synchronise the device-counter and the tool's clock, especially under non-real-time systems (e.g. WindowsTM).

EXAMPLE 1 cEMI Server with a clock rate of 1 MHz

1 tick = 1 µs

Every 65 ms the internal counter has an overflow, causing the need for a fast and accurate timing on the tool side.

b) The timestamp information only delivers "ticks", which means the tool has to know the internal clock rate of the device to be able to calculate the relative time. This dependency makes it nearly impossible to develop tools being compatible with different cEMI device-types, as this information (clock rate) cannot be read out of the device.

For these reasons, the Additional Information Type called "Extended relative timestamp" is defined:

Table D.5

Type ID (1 octet)	**Len (1 octet)**	**Timestamp (4 octets)**			
06h	04h				

The timestamp shall contain the 4 octet value of the free running counter of the cEMI Server at the time of frame reception. The value shall be measured always at the same position in the frame (e.g. beginning of first start bit) in order to allow the client the precise calculation of the time difference between successive frames.

A cEMI Server implementing the Extended relative time stamp shall provide the Property PID_TIME_BASE (PID = 55, PDT_UNSIGNED_INT) in the cEMI Server Interface Object containing the used time base. The time base shall be measured in nanoseconds per tick of the free running counter. The cEMI Client can read out this time base and display the relative time between frames in a device independent way (e.g. in µs).

EXAMPLE 2 cEMI Server with a clock rate of 1 MHz

1 tick = 1 µs → time base = 1000 (dec)

Overflow after 2^{32} µs = 71,582 min, uncritical

EXAMPLE 3 cEMI Server with a clock rate of 8 MHz

1 tick = 1/8 µs → time base = 125 (dec)

Overflow after 8,94 min, uncritical

D.1.1.4.3.3 Frame examples with indication of additional information

Table D.6 — L_Data.req message without any additional information

Message Code	Additional Info Length	Service Information
MC	**AddIL**	**….**
11h	0	…

Table D.7 — L_Data.con message, from a Powerline medium frame, with Domain Address

Message Code	Additional Info Length	Additional Information				Service Information
MC	**AddIL**	**Type ID**	**Len**	**information**		**….**
2Eh	4	01h	2	Domain	Address	…

Table D.8 — L_Data.ind message, RF frame with RF Control Information and KNX Serial Number (SN; SN6 = MSB)

Message Code	Additional Info Length	Additional Information									Service Information
MC	**AddIL**	**Type ID**	**Len**	**information**							**….**
29h	9	02h	7	RF-Ctrl	S N_6	S N_5	S N_4	S N_3	S N_2	SN_1	…

Table D.9 — L_Data.ind message, RF frame with RF Control Information and RF Domain Address (DoA; DoA6 = MSB)

Message Code	Additional Info Length	Additional Information									Service Information
MC	**AddIL**	**Type ID**	**Len**	**information**							**…**
29h	9	02h	7	RF-Ctrl	DoA_6	…	…	…	…	DoA_1	…

D.1.1.4.4 Service Information

Use of the Service Information is shown in the following clauses of this document.

D.1.1.5 Data Link Layer messages

D.1.1.5.1 Flow Control

cEMI Client

To keep the flow control for Data Link Layer services as simple as possible (this allows a simple flow control state machine in the cEMI client), it is recommended that:

— a cEMI client sends a new Data Link Layer request only when the confirmation of the preceding request is received, or

— a request-to-confirmation timeout is recognised; the recommended time out for the cEMI client is 3 s.

A cEMI client shall at any time be able to accept an indication message from the cEMI Server.

Behaviour of the cEMI Server

A cEMI Server device shall have a receive buffer for one or more cEMI frames. The cEMI Server shall accept new frames from the cEMI client only if the receive buffer is not full. The request frames in the receive buffer shall be treated sequentially after the FIFO rule (first in, first out).

NOTE For cEMI Servers with a receive buffer for more than one frame, the order of received frames can be any concerning the type of received messages (Data Link Layer or Management).

During treatment of a request that is not yet confirmed to the cEMI client, the cEMI Server shall accept a new request from the cEMI client. This is used e.g. for management requests during an L_Data.req/L_Data.con cycle.

D.1.1.5.2 General exception handling

D.1.1.5.2.1 Collisions

If during sending on the bus (on media with CSMA/CA method) the cEMI Server device detects a collision, it shall stop instantly its transmission.

The cEMI client shall not be informed about the detected collisions.

In case of collision, the cEMI Server device shall (instead of complete transmission of its own send request) receive the message from the "winning" device from the bus. This shall lead to a corresponding Data Link Layer indication message from the cEMI Server to the cEMI client.

D.1.1.5.2.2 Physical Medium Error

In case of a physical medium error (transmission medium permanently disturbed, transmission not possible), the cEMI Server device is not able to send the request message on the bus. On level of the Data Link Layer message flow, the cEMI client shall recognise such an error by a confirmation timeout, i.e. no Data Link Layer confirmation message shall be sent from the cEMI Server to the cEMI client.

D.1.1.5.2.3 Use of Frame Type flag (FT) and Extended Frame Format (EFF) field

Any receiver shall be tolerant towards the use of the Frame Format. It shall accept the use of an Extended Frame even if the size of the payload would allow a Standard Frame. Therefore, if a cEMI Server receives a cEMI Frame with FT = 0 and EFF = 0, thus denoting a Standard Frame for an APDU > 15 octets, yet with a payload shorter then 15 octets, then the cEMI Server may:

— either correct the situation and transmit an L_Data_Standard Frame, or

— transmit an L_Data_Extended Frame with payload smaller than 15 octets.

D.1.1.5.3 L_Data services

D.1.1.5.3.1 Implementation and usage

The L_Data services within the Data Link Layer Interface are mandatory for a cEMI Server.

D.1.1.5.3.2 Basic frame structure for L_Data messages

Table D.10

Message Code	Additional Info Length	Additional Information	Control field 1	Control field 2	Src. High	Src. Low	Dest. High	Dest. Low	NPDU	
MC	**AddIL**	**…**	**Ctrl1**	**Ctrl2**	**SAH**	**SAL**	**DAH**	**DAL**	**L**	TPCI/APCI & data
1 octet	1 octet	var. length	1 octet	1 octet	2 octets		2 octets		1 octet	var. length

MC: message code

AddIL: length of additional information

Ctrl1: Control field 1

1 octet							
Ctrl1							
7	6	5	4	3	2	1	0
FT	0	R	SB	P		A	C

— Frame Type flag (FT) (msb):

description: This shall specify the Frame Type that shall be used for transmission or reception of the frame

encoding: 0: extended frame

1: standard frame

— Repeat flag (R) (bit 5):

description: Repeat, not valid for all media.

encoding: 0: repeat frame on medium if error

1: do not repeat

— System Broadcast flag (SB) (bit 4):

description: This shall specify whether the frame is transmitted using system broadcast communication mode or broadcast communication mode (applicable only on open media).

encoding: 0: system broadcast

1: broadcast

Priority (P) (bit 3 and bit 2):

description: This shall specify that Priority that shall be used for transmission or reception of the frame.

encoding: Please refer to “Usage of priority”.

— Acknowledge request flag (A) (bit 1):

description: This shall specify whether an L2-acknowledge shall be requested for the L_Data.req frame or not. This is not valid for all media.

encoding: 0: no acknowledge is requested

1: acknowledge requested

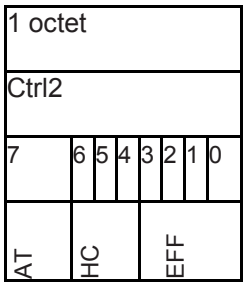
— Confirm flag (C) (lsb):

description: In L_Data.con this shall indicate whether there has been any error in the transmitted frame.

encoding: 0: no error

1: error

— Ctrl2: Control field 2



— Destination Address Type (AT) (msb):

encoding: 0: individual

1: group

— Hop Count (HC) (bit 6 to bit 4):

encoding: value binary encoded

— Extended Frame Format (EFF) (bit 3 to bit 0 (lsb)):

encoding: 0000b: for standard frame (long frames, APDU > 15 octet)

— SAH: Source Address High (Source Subnetwork Address)

— SAL: Source Address Low (Source Device Address)

— DAH: Destination Address High (Destination Subnetwork Address)

— DAL: Destination Address Low (Destination Device Address)

— L: Information-Length (max. value is 255); number of NPDU octets, TPCI octet not included!
→ L = number of octets (without FCS), counting starts with the octet after the TPCI octet
(0 = no octet after the TPCI)

D.1.1.5.3.3 L_Data.req

Table D.11

Message Code	Additional Information Length	Additional Information	Control field 1	Control field 2	Src. High	Src. Low	Dest. High	Dest. Low	NPDU	
MC	**AddIL**	**…**	**Ctrl1**	**Ctrl2**	**SAH**	**SAL**	**DAH**	**DAL**	**L**	TPCI/APCI & data
1 octet	1 octet	var. length	1 octet	1 octet	2 octets		2 octets		1 octet	var. length
11h	…	…	x0rxppa0	…	…		…		…	…

If a cEMI Server receives the L_Data.req message with Source Address set to 0000h, then the cEMI Server shall fill in the Source Address field before sending the message onto the KNX network. Typically, this is the cEMI Server device's own Individual Address.

If the field Source Address is not set to 0000h, the cEMI Server shall send the frame onto the KNX network with the Source Address received from the cEMI client with L_Data.ind message.

On KNX media with Data Link Layer acknowledge (like TP1), this feature is only possible if the cEMI Server device supports more than one Individual Address for sending L2 acknowledges onto the bus in case of received frames (confirmed services, responses in point to point connectionless or connection-oriented communication mode to requests in point to point connectionless or connection oriented communication mode). If the cEMI Server device does not support multiple IAs, the cEMI Server shall always insert its own Individual Address in the Source Address field. This standard is limited to cEMI Server implementations in devices supporting only a single Individual Address.

A cEMI Server in full transparent mode, this is a cEMI Server without an own Individual Address, shall send back a negative confirmation (Confirm Flag set to 1 in L_Data.con) if a L_Data.req message is received from cEMI client with Source Address set to 0000h.

Use of flags in Ctrl1-field

— Frame Type (FT)

description: This field shall specify whether the frame is a standard frame or an extended frame.

encoding:

Table D.12

	Meaning/behaviour on bus media		
Value of FT-Flag	**TP1**	**PL110**	**RF[a]**
0	Extended	Extended	Extended
1	Standard	Standard	Extended

[a] Bit value is “Don’t care” if cEMI Server is interface to RF, since RF uses only extended frames.

— Repetition (r)

description: This shall specify whether repetitions shall be sent on the medium. This flag is relevant only on media with possibility of Data Link Layer controlled frame repetitions (TP1, PL110);

NOTE: "Don’t care" means that (LL-) repetitions shall be sent (if error; e.g. on TP1: if no ACK-, NACK- or BUSY-frame) according the default media’s repetition behaviour, i.e. according the value of the Property PID_MAX_RETRY_COUNT.

encoding: 0: do not repeat if error

1: Don’t care

Table D.13

	Meaning/behaviour on bus media		
Value of r-Flag	**TP1 [a]**	**PL110**	**RF**
0: no repetitions	No repetitions	No repetitions	No repetitions
1: Don’t care	Repetitions are allowed	Repetitions are allowed	No repetitions

— System Broadcast (SB)

description: This flag shall only be applicable on open media.

It shall be "don’t care" on “closed" media (e.g. TP1), i.e. a cEMI Server to a closed medum shall ignore the SB-flag.

encoding: 0: system broadcast

1: broadcast

— Acknowledge request (A)

description: This shall specify whether an L2-acknowledge shall be requested for the L_Data.req frame or not. This is not valid for all media.

"Don't care" means that no explicit L2-acknowledge is requested by the upper layer(s); this means the default behaviour of the Data Link Layer concerning L2-acknowledge requesting applies, as laid down in the standards of the communication medium.

encoding: 0: no acknowledge is requested

1: acknowledge requested

Table D.14

	Meaning / behaviour on bus media		
Value of a-Flag	**TP1**	**PL110**	**RF**
0: Don't care	Requested	Requested	Not applicable
1: Ack requested	Requested	Requested	Not applicable

— C

description: The C-flag shall not be used in the L_data.req. It shall be "don't care", i.e. a cEMI Server shall ignore the C-Flag in L_Data.req.

D.1.1.5.3.4 L_Data.con

Table D.15

Message Code	Additional Information Length	Additional Information	Control field 1	Control field 2	Src. High	Src. Low	Dest. High	Dest. Low	NPDU	
MC	**AddIL**	**…**	**Ctrl1**	**Ctrl2**	**SAH**	**SAL**	**DAH**	**DAL**	**L**	TPCI/APCI & data
1 octet	1 octet	var. length	1 octet	1 octet	2 octets		2 octets		1 octet	var. length
2Eh	…	…	x0rxpp xC	…	…		…		…	…
NOTE C: Confirm Flag: 1 = Error; 0 = No Error.										

The L_Data.con shall be a "local" primitive generated by the cEMI Server's Data Link Layer for its own cEMI client to indicate that it is satisfied with the transmission (error flag C cleared) or not (error flag C set).

If a message is sent onto a medium with immediate acknowledgement, (L2 acknowledge) the confirmation message is normally generated after receiving this immediate acknowledgement.

The Source Address field shall be used to indicate the Source Address of the requested message. The Destination Address field shall be used to indicate the Destination Address of the requested message; the Destination Address Type (AT-bit, Individual or Group) shall be the Destination Address Type of the requested message.

Use of flags in Ctrl1-field

— Frame Type (FT)

description: The cEMI Server shall set the flag to the value according the frame type that is sent onto the bus.

encoding: FT = 0: extended frame

FT = 1: standard frame

— Repeat flag (R) (bit 5):

description: "Don't care": the flag can have the same value as in the original L_Data.req, but it shall not be interpreted by the cEMI Client.

— SB

description: "Don't care": the flag can have the same value as in the original L_Data.req, but it shall not be interpreted by the cEMI Client.

— a

description: "Don't care": the flag can have the same value as in the original L_Data.req, but it shall not be interpreted by the cEMI Client.

encoding:

— Confirm (C)

encoding: 0 = No Error

1 = Error

D.1.1.5.3.5 L_Data.ind

Table D.16

Message Code	Additional Information Length	Additional Information	Control field 1	Control field 2	Src. High	Src. Low	Dest. High	Dest. Low	NPDU	
MC	**AddIL**	**…**	**Ctrl1**	**Ctrl2**	**SAH**	**SAL**	**DAH**	**DAL**	**L**	TPCI/APCI & data
1 octet	1 octet	var. length	1 octet	1 octet	2 octets		2 octets		1 octet	var. length
29h	…	…	x0rxppxx	…	…		…		…	…

The Source Address field shall contain the Source Address that is received by the cEMI Server from the bus in the originating medium specific frame.

Use of flags in Ctrl1-field

— Frame Type (FT):

description:	The cEMI Server shall set the flag to the value according the frame type that is received from the bus.
encoding:	FT = 0: extended frame FT = 1: standard frame

— Repeat flag (R) (bit 5):

description:	If the cEMI Server receives a repeated frame from the bus and if it also receives the originating frame (and acknowledged it with the LL-IAck) then the cEMI Server shall not indicate the repeated frame to the cEMI client.
encoding:	0: repeated L_Data frame on media 1: not repeated frame on media

— System Broadcast (SB):

description:	This field shall be applicable only on open media; it shall be "don't care" on ("closed") media (e.g. TP1), i.e. a cEMI client shall ignore the SB-flag if received from a cEMI Server as interface from a closed media.
encoding:	SB = 0: system broadcast SB = 1: broadcast

— a:

description:	TP1, PL, RF: The a-flag shall not be used; it shall be "don't care"; this is, the cEMI client shall ignore the a-flag.
encoding:	

— Confirm (C):

description:	The C-flag shall in the L_Data.ind be "don't care"; the C-flag does not exist; i.e. a cEMI client shall ignore the C-flag in L_Data.ind
encoding:	

D.1.1.5.4 L_Poll_Data service

D.1.1.5.4.1 Implementation and usage

L_Poll_Data are applicable only with TP physical medium devices. L_Poll_Data is optional for a cEMI Server.

D.1.1.5.4.2 L_Poll_Data.req

Table D.17

Message Code	Additional Info Length	Additional Information	Control field 1	Control field 2	Src. High	Src. Low	Dest. High	Dest. Low	number of slots
MC	**AddIL**	**…**	**Ctrl1**	**Ctrl2**	**SAH**	**SAL**	**DAH**	**DAL**	**NoS**
13h	…	…	x0r0ppa0	…	…	…	Polling	Group	0000ssss

D.1.1.5.4.3 L_Poll_Data.con

Table D.18

Message Code	Additional Info Length	Additional Information	Control field 1	Control field 2	Src. High	Src. Low	Dest. High	Dest. Low	number of slots	Poll Data 0 … (ssss – 1)
MC	**AddIL**	**…**	**Ctrl1**	**Ctrl2**	**SAH**	**SAL**	**DAH**	**DAL**	**NoS**	**Poll Data**
25h	…	…	x0r0ppxC	…	XX	XX	Polling	Group	0000ssss	…
NOTE C: Confirm Flag: 1 = ok; 0 = not ok.										

D.1.1.5 L_Raw service

D.1.1.5.5.1 Implementation and usage

The L_Raw service is optional for a cEMI Server. It is intended for special use, i.e. for use in testing or diagnostic tools. It shall not be used for "normal operation".

D.1.1.5.5.2 Basic message structure for L_Raw messages

Table D.19

Message Code	Additional Info Length	Additional Information	Raw Frame on Medium[7)]
MC	**AddIL**	**…**	**Data**
1 octet	1 octet	var. length	

The L_Raw service, as an important feature, shall also contain the frame checksum information as part of the field Data (Raw Frame on medium).

7) Raw Frame on media: starting with the Ctrl octet & ending with the CRC octet(s).

D.1.1.5.5.3 L_Raw.req[8) 9)]

Table D.20

Message Code	Additional Info Length	Additional Information	Raw Frame on Medium
MC	**AddIL**	**…**	**Data**
10h	…	…	…

The L_Raw.req service primitive shall be used for sending a frame in raw format onto any of the given KNX media This message can be used e.g. in a test environment (by a test tool like the EITT) to generate faulty frames and to send them onto the KNX network.

A time delay until sending can be used. It shall be available as a field within Additional Information.

If no time delay is present as additional information field, the frame shall be sent onto the KNX medium by the cEMI Server device with the regular interframe time.

If the Delay Time is present as additional information with Type ID = 05h, then the Delay Time shall be a 4 octet long (unsigned) counter value.

If Delay Time = 00000000h the frame shall be sent immediately. Otherwise, the frame shall be sent when the free running system counter of the sending device is equal to the value given in Delay Time.

D.1.1.5.5.4 L_Raw.con

Table D.21

Message Code	Additional Info Length	Additional Information	Raw Frame on Medium
MC	**AddIL**	**…**	**Data**
2Fh	…	…	…

The L_Raw.con shall be a local primitive generated by the cEMI Server's Data Link Layer for its own cEMI client to indicate whether it is satisfied with the transmission or not.

A positive L_Raw.con message shall contain the same data in the message as it is received with L_Raw.req. A positive L_Raw.con message shall be generated as soon as the raw frame is sent completely.

Negative L_Raw.con message: to be defined.[10)]

8) EMI 2: the equivalent service is specified as L_Plain_Data.req.

9) If the cEMI Server supports L_Raw service and multiple media types, it's not allowed to indicate more than one medium as "supported" in PID_Medium_Type in the cEMI Server object until it's fully specified (e.g. by a channel concept) how the cEMI client knows the frame format to be used in L_Raw.req.

10) Currently (October 2011), no implementation of L_Raw.req or L_raw.con on cEMI is planned or known; to be defined if needed for an implementation.

D.1.1.5.5.5 L_Raw.ind

Table D.22

Message Code	Additional Info Length	Additional Information	Raw Frame on Medium
MC	**AddIL**	**…**	**Data**
2Dh	…	…	…

The L_Raw.ind service shall be used for receiving a frame in raw format from any of the given KNX media. This message can be used e.g. in a test environment (by a test tool like the EITT) to receive faulty frames from the KNX network and to display them for diagnostic purposes.

L_Raw.ind shall be the Data Link Layer service primitive that transfers received frames from the local Layer-2 completely (including FCS) to the local Layer-2 user, including all received octets in raw format.

As a difference to the L_Busmon.ind service, Data Link Layer acknowledges shall not be transferred with the L_Raw.ind service primitive.

D.1.1.5.5.6 L_Busmon.ind

Table D.23

Message Code	Additional Info Length	Additional Information	Raw Frame on Medium
MC	**AddIL**	**…**	**Data**
2Bh	…	…	…

The L_Busmon.ind message is a Data Link Layer service primitive that shall be used to transfer every received message from the local Layer-2 completely (inclusive the FCS) to the local Layer-2 user, including all received octets in raw format. Data Link Layer acknowledges shall also be transferred.

The value of the last octet within the L_Busmon.ind message is usually the FCS octet that is received from the bus by the cEMI Server device. It is the task of the external Busmonitor application (cEMI Client device/tool) to check its correctness.

An L_Busmon.ind service primitive using the Status information field (Busmonitor Error Flags) is not obliged to support all the flags and the sequence number within the status octet. Unused bits shall not be used for other purpose(s); they shall be fixed to 0.

The field Timestamp Relative shall be a 16 bit value and shall refer to the relative time taken exactly at the time when the frame's control field is completely received at the Data Link Layer. The time shall be the value of the free running counter of the BAU. The time unit ("tick") depends on the clock rate of the BAU micro controller. The field Timestamp Relative shall be mapped to an own information type within the Additional Information.

D.1.1.6 Transport Layer messages

D.1.1.6.1 Basic frame structure for cEMI Transport Layer messages

The cEMI Transport Layer interface shall provide two services.

1) T_Data_Individual

2) T_Data_Connected

The basic Frame Structure for both services shall be as specified below.

Message Code	Additional Info Length	Additional Information	Reserved	TPDU			
MC	**AddIL**	**…**		**L**	**reserved**	**APCI /Data**	**Data**
1 octet	1 octet	var. length	6 octets	1 octet	6 bits	10 bits	var. length
			000000000000h		000000b		

Figure D.2 — Basic cEMI Transport Layer frame structure

— MC: message code

— AddIL: length of additional information

— Additional Information: This shall contain one or more Additional Information fields according the value of the field Additional Information Length.

— Reserved: This field is reserved and shall be filled with 000000000000h.

NOTE The goal of this empty field is to align the position of the fields L and further in the message buffer with the position of these fields in the cEMI L_Data-frame.

— L: This field shall contain the Information Length. The maximal value shall be 255.The value of this field shall be the size in octets of the field TPDU, not including the field L, not including the field reserved and starting with 1 with the field APCI/Data. The minimal value of this field shall thus be 1.

— reserved: This field is reserved and shall be filled with 000000b.

NOTE 2 The goal of this empty field is to align the position of the fields APCI and further in the message buffer with the position of these fields in the cEMI L_Data-frame.

— APCI/Data

name (abbreviation): Application Layer Protocol Control Information (APCI) or Data

description: This field shall contain the APCI of the transported AL service, or the data.

encoding: See KNX application layer definitions

D.1.1.6.2 Flow Control

Unlike the cEMI Data Link Layer, the cEMI Transport Layer does not foresee confirmations of the cEMI frames. The cEMI Transport Layer has no provisions for flow control.

D.1.1.6.3 General exception handling

There is no general exception handling for cEMI Transport Layer frames.

D.1.1.6.4 T_Data_Individual service

D.1.1.6.4.1 Implementation and usage

The implementation of the T_Data_Individual service depends on the cEMI Profile.

D.1.1.6.4.2 Basic frame structure for T_Data_Individual messages

The basic frames structure shall be as specified in D.1.1.6.1 (Basic frame structure for cEMI Transport Layer messages).

D.1.1.6.4.3 Message flow

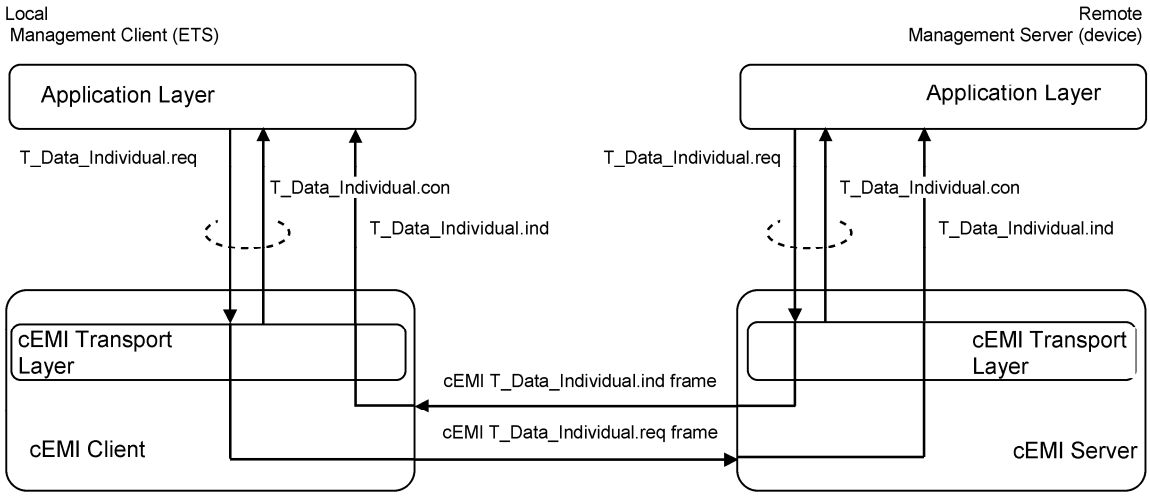


Figure D.3 — Message flow for T_Data_Individual

D.1.1.6.4.4 cEMI T_Data_Individual.req frame

If the local user of the Transport Layer in the cEMI Client device applies the T_Data_Individual.req service primitive then the cEMI Client shall send the cEMI T_Data_Individual.req frame to the cEMI Server.

If the cEMI Client is satisfied with the transmission, it shall apply the T_Data_Individual.con service primitive to the local cEMI Transport Layer user.

If the cEMI Server receives a cEMI T_Data_Individual.req frame, it shall pass the contained APDU to the remote Application Layer by a T_Data_Individual.ind service primitive.

Message Code	Additional Info Length	Additional Information	Reserved	TPDU			
MC	**AddIL**	**…**		**L**	**reserved**	**APCI /Data**	**Data**
1 octet	1 octet	var. length	6 octets	1 octet	6 bits	10 bits	var. length
4Ah			000000000000h		000000b		

Figure D.4 — cEMI T_Data_Individual.req frame

D.1.1.6.4.5 cEMI T_Data_Individual.ind frame

If the remote user of the Transport Layer in the cEMI Server device applies the T_Data_Individual.req service primitive then the cEMI Server shall send the cEMI T_Data_Individual.ind frame to the cEMI Client.

If the cEMI Server is satisfied with the transmission, it shall apply the T_Data_Individual.con service primitive to the remote cEMI Transport Layer user.

If the cEMI Client receives a cEMI T_Data_Individual.ind frame, it shall pass the contained APDU to the local Application Layer by a T_Data_Individual.ind service primitive.

Message Code	Additional Info Length	Additional Information	Reserved	TPDU			
MC	**AddIL**	**…**		**L**	**reserved**	**APCI /Data**	**Data**
1 octet	1 octet	var. length	6 octets	1 octet	6 bits	10 bits	var. length
94h			000000000000h		000000b		

Figure D.5 — cEMI T_Data_Individual.ind frame

D.1.1.6.5 T_Data_Connected service

D.1.1.6.5.1 Implementation and usage

The implementation of the T_Data_Connected service depends on the cEMI Profile.

D.1.1.6.5.2 Basic frame structure for T_Data_Connected messages

The basic frames structure shall be as specified in D.1.1.6.1 (Basic frame structure for cEMI Transport Layer messages).

D.1.1.6.5.3 Message flow

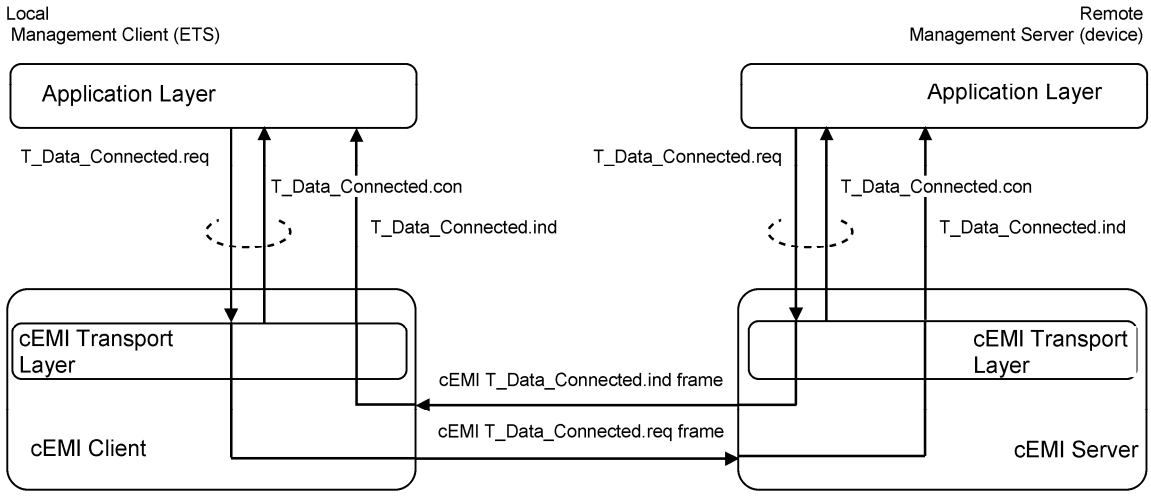


Figure D.6 — Message flow for T_Data_Connected

D.1.1.6.5.4 cEMI T_Data_Connected.req frame

If the local user of the Transport Layer in the cEMI Client device applies the T_Data_Connected.req service primitive then the cEMI Client shall send the cEMI T_Data_Connected.req frame to the cEMI Server.

If the cEMI Client is satisfied with the transmission, it shall apply the T_Data_Connected.con service primitive to the local cEMI Transport Layer user.

If the cEMI Server receives a cEMI T_Data_Connected.req frame, it shall pass the contained APDU to the remote Application Layer by a T_Data_Connected.ind service primitive.

Message Code	Additional Info Length	Additional Information	Reserved	TPDU			
MC	**AddIL**	**...**		**L**	**reserved**	**APCI /Data**	**Data**
1 octet	1 octet	var. length	6 octets	1 octet	6 bits	10 bits	var. length
41h			000000000000h		000000b		

Figure D.7 — cEMI T_Data_Connected.req frame

D.1.1.6.5.5 cEMI T_Data_Connected.ind frame

If the remote user of the Transport Layer in the cEMI Server device applies the T_Data_Connected.req service primitive then the cEMI Server shall send the cEMI T_Data_Connected.ind frame to the cEMI Client.

If the cEMI Server is satisfied with the transmission, it shall apply the T_Data_Connected.con service primitive to the remote cEMI Transport Layer user.

If the cEMI Client receives a cEMI T_Data_Connected.ind frame, it shall pass the contained APDU to the local Application Layer by a T_Data_Connected.ind service primitive.

Message Code	Additional Info Length	Additional Information	Reserved	TPDU			
MC	**AddIL**	**...**		**L**	**reserved**	**APCI /Data**	**Data**
1 octet	1 octet	var. length	6 octets	1 octet	6 bits	10 bits	var. length
89h			000000000000h		000000b		

Figure D.8 — cEMI T_Data_Connected.ind frame

D.1.1.7 Services for Local device management

D.1.1.7.1 Introduction

This clause specifies the communication relevant parts (services) for local device management.

The local management services shall be confirmed services. The M_PropRead.req as well as the M_PropWrite.req service primitives shall always be followed by the corresponding M_PropRead.con or M_PropWrite.con service primitives containing information about success of the request. The M_FuncPropCommand.req as well as the M_FuncPropStateRead.req service primitives shall always be followed by the corresponding M_FuncPropCommand.con servie service primitive containing information about success of the request.

Messages generated by the cEMI Server, e.g. for event notification, shall use the unconfirmed M_PropInfo.ind service.

D.1.1.7.2 Management services flow control

cEMI Client

To keep the flow control for management services as simple as possible (this allows for a simple flow control state machine in the cEMI client), it is recommended that:

— a cEMI client sends a new request only when the confirmation of the preceding request is received, or

— a request-to-confirmation timeout is recognised; the recommended value for this time out for the cEMI client is 1 s.

A cEMI client shall be able to accept an indication message from the cEMI Server at any time.

Behaviour of the cEMI Server

A cEMI Server device shall have a receive buffer for one or more cEMI messages. The cEMI Server shall accept new frames from the cEMI client only if the receive buffer is not full. The request frames in the receive buffer shall be treated sequentially according to a FIFO queue (first in, first out).

NOTE For cEMI Servers with a receive buffer for more than one message, the order of received frames can be any concerning the type of received messages (Data Link Layer or Management).

During treatment of a request that is not yet confirmed to the cEMI client, the cEMI Server shall accept a new request from the cEMI client. This is used e.g. for L_Data.req service primitives during an M_PropRead.req, M_PropRead.con, M_PropWrite.req M_PropWrite.con cycle.

D.1.1.7.3 Data Properties

D.1.1.7.3.1 Basic message structure for Data Properties

Table D.24

Message Code	Interface Object Type		Object Instance	Property ID	number of elements	start index	Data
MC	**IOTH**	**IOTL**	**OI**	**PID**	**NoE**	**SIx**	**Data**
1 octet	2 octets		1 octet	1 octet	4 bit	12 bits	var. length

MC: message code

IOTH: Interface Object Type, high octet

IOTL: Interface Object Type, low octet

OI: Object Instance; 0 = reserved (not used); 1 = 1st instance; 2 = 2nd instance; 3 = 3rd instance, …

PID: Property Identifier

NoE: Number of elements for an array structured Property; if the Property is not an array: NoE = 1

SIx: Start Index within an array-structured Property, the first element shall be placed at index 1; the array element "0" shall contain the current number of valid array elements (unsigned 16 bit value).

Data: This is the Data field of the message. The length of the data field shall depend on the data format of the Property and in case of an array structured Property value also on the number of array elements that are accessed.

D.1.1.7.3.2 M_PropRead.req

The M_PropRead.req message shall be applied by the management client to read the value of a Property of an Interface Object in the management server. The Interface Object of the partner shall be addressed with an Object Type and an Object Instance number. A Property within an Interface Object can be structured as an array of elements. Therefore, the Property within the Interface Object shall be addressed with a Property ID, the number of elements (NoE) and a start index (SIx). NoE shall indicate the number of array elements starting with the given SIx in the Property value that the management client wants to read. The management server shall confirm the request with an M_PropRead.con message. The M_PropRead.req shall not contain any further data.

Table D.25 — M_PropRead.req message

Message Code	Interface Object Type	Object Instance	Property ID	number of elements	start index
MC	**IOTH IOTL**	**OI**	**PID**	**NoE**	**SIx**
FCh	…	≥ 1	…	> 0	…

For reading the currently valid number of array elements, NoE shall be set to 1 and start index to 0.

D.1.1.7.3.3 M_PropRead.con

The management server shall confirm an M_PropRead.req message with an M_PropRead.con message. This response shall contain the requested number of elements beginning at the requested start index within the requested Property of the addressed Interface Object.

Table D.26 — M_PropRead.con message, positive response

Message Code	Interface Object Type	Object Instance	Property ID	number of elements	start index	Data
MC	**IOTH IOTL**	**OI**	**PID**	**NoE**	**SIx**	**Data**
FBh	as in .req	as in .req	as in .req	as in .req	as in .req	…

Table D.27 — M_PropRead.con message, number of valid array elements

Message Code	Interface Object Type	Object Instance	Property ID	number of elements	start index	Data
MC	**IOTH IOTL**	**OI**	**PID**	**NoE**	**SIx**	**Number of elements**
FBh	as in .req	as in .req	as in .req	1	0	number (unsigned 16 value)

Error Handling

If the management server has a problem, e.g. Interface Object or Property does not exist, then the NoE shall be set to zero and the Start Index of the response shall be set to same value as received with the request. The data field of a negative response shall contain error information. The error information of a negative confirmation shall be a one octet long enumerated data field.

Table D.28 — M_PropRead.con message, negative response

Message Code	Interface Object Type	Object Instance	Property ID	number of elements	start index	Error information
MC	**IOTH IOTL**	**OI**	**PID**	**NoE**	**SIx**	**Error Code**
FBh	as in .req	as in .req	as in .req	0	as in .req	…

D.1.1.7.3.4 M_PropWrite.req

The M_PropWrite.req message shall be applied by the management client to modify the value of a Property of an Interface Object in the management server. The Interface Object of the partner shall be addressed with an Object Type and an Object Instance number. A Property within an Interface Object can be structured as an array of elements. Therefore, the Property within the Interface Object shall be addressed with a Property ID, the number of elements (NoE) and a start index (SIx). NoE shall indicate the number of array elements starting with the given SIx in the Property value that the management client wants to write. The management server shall confirm the request with a M_PropWrite.con message.

Table D.29 — M_PropWrite.req message

Message Code	Interface Object Type	Object Instance	Property ID	number of elements	start index	Data
MC	**IOTH IOTL**	**OI**	**PID**	**NoE**	**SIx**	**Data**
F6h	…	≥ 1	…	> 0	> 0	…

D.1.1.7.3.5 M_PropWrite.con

The management server shall confirm an M_PropWrite.req message with an M_PropWrite.con message. The (positive) response shall not contain any data, i.e. the data field shall not be present. The number of elements (value of NoE) and the start index (value of SIx) shall be set to the same value as in the request.

Table D.30 — M_PropRead.con message, positive response

Message Code	Interface Object Type	Object Instance	Property ID	number of elements	start index
MC	**IOTH IOTL**	**OI**	**PID**	**NoE**	**SIx**
F5h	as in .req	as in .req	as in .req	as in .req	as in .req

Error Handling

If the management server has a problem, e.g. Interface Object or Property does not exist, then the NoE shall be set to zero and the Start Index of the response shall be set to same value as received with the request. The data field of a negative response shall contain error information. The error information of a negative confirmation shall be a one octet long enumerated data field.

Table D.31 — M_PropWrite.con message, negative response:

Message Code	Interface Object Type	Object Instance	Property ID	number of elements	start index	Error information
MC	**IOTH IOTL**	**OI**	**PID**	**NoE**	**Six**	**Error Code**
F5h	as in .req	as in .req	as in .req	0	as in .req	…

M_PropWrite.con shall contain information about the result of the Property access in the cEMI Server's database, not only about the writing of the Property alone, if the full error handling is supported, especially if error code 04 "memory error" is supported.

The reaction time for the M_PropWrite.con to be sent shall be specified for the host protocol on which cEMI is used.

D.1.1.7.3.6 M_PropInfo.ind

The M_PropInfo.ind message shall be applied by the management server to send an event notification to the management client, e.g. about a changed management Property value.

After reception of the M_PropInfo.ind, the management client shall check whether the contained data is relevant to one or more of the management procedures it supports. If so, these procedures shall be called with the received data. If no, the message shall be ignored.

M_PropInfo.ind shall always be an unconfirmed service.

Table D.32 — M_PropInfo.ind message

Message Code	Interface Object Type	Object Instance	Property ID	number of elements	start index	Data
MC	**IOTH IOTL**	**OI**	**PID**	**NoE**	**SI**	**Data**
F7h	…	≥ 1	…	> 0	> 0	…

D.1.1.7.3.7 cEMI Server exception handling after management service request

D.1.1.7.3.7.1 Error handling

If a confirmed service (M_PropRead.req, M_PropWrite.req) cannot be executed successfully by the cEMI management server, the cEMI Server shall generate a negative confirmation as an 'Error'. Such an error shall be transmitted to the cEMI management client within the negative response PDU.

As the minimum requirement, an "Unspecified Error" shall be returned if any problem occurs.

D.1.1.7.3.7.2 Error Code Set

The Error Code Set uses an 8 bit enumeration data type (N_8).

Table D.33 — Error Code Set

Error Code	Error Type	Description	Service
00h	Unspecified Error	unknown error	R/W
01h	Out of Range	write value not allowed (general, if not error 2 or 3)	W
02h	Out of MaxRange	write value to high	W
03h	Out of MinRange	write value to low	W
04h	Memory Error	memory can not be written or only with fault(s)	W
05h	Read Only	write access to a 'read only' or a write protected Property	W
06h	Illegal COMMAND	COMMAND not valid or not supported	W
07h	Void DP	read or write access to an non existing Property	R/W
08h	Type Conflict	write access with a wrong data type (Datapoint length)	W
09h	Prop. Index Range Error	read or write access to a non existing Property array index	R/W
0Ah	Value temporarily not writeable	The Property exists but can at this moment not be written with a new value	W

In case of multiple errors, the error type in the negative confirmation shall be given by the error testing sequence: the first detected error shall be the error indication in the negative confirmation.

D.1.1.7.3.7.2 cEMI Server behaviour after reception of a management service request

If only the minimum requirements concerning error handling are supported, the following applies.

a) If any problem is detected, the access shall be confirmed by the negative response 'Unspecified Error", else continue with 2).

b) Send a positive service confirmation.

If a more sophisticated error handling is supported, providing more differentiating error identification, the following applies.

1) If the accessed (read or write) Property does not exist in the cEMI Server, the access shall be confirmed with the negative response 'Void DP'; otherwise continue with 2).

2) If the array field(s) within the accessed (read or write) Property causes a problem, e.g. too many elements are accessed, the access shall be confirmed with the negative response "Property Index/Range Error"; otherwise continue with 3) for write, respectively 8) for read access.

3) If write access on the write accessed Property is not allowed, the write access shall be confirmed by the negative response 'Read Only'; otherwise continue with 4).

4) If the command accompanying a Property value (write access) is not supported by the cEMI Server, the service request shall be confirmed with the negative response ‘Illegal CMD’; otherwise continue with 5).

5) If the Data (Point) Type of the received Property value does not comply with one of the Properties that is requested to be written (detected by different data lengths), the service request shall be confirmed by the negative response ‘Type Conflict’; otherwise continue with 6).

6) If the Property value to be written is out of the allowed value range, the service request shall be confirmed by the negative response 'Out of Range', 'Out of MaxRange' or 'Out of MinRange' as appropriate; otherwise continue with 7).

7) If the Property value to be written cannot be stored successfully in the cEMI Server’s device database, the service request shall be confirmed by the negative response ‘Memory Error’; otherwise continue with 8).

8) Send a positive service confirmation.

For each cEMI Server implementation, the supported Error Codes shall be declared in the corresponding cEMI Server profile.

D.1.1.7.4 Function Properties

D.1.1.7.4.1 M_FuncPropCommand.req

The M_FuncPropCommand.req message shall be applied by the Management Client to call a Property Function of an Interface Object in the Management Server. The Interface Object of the partner shall be addressed with an Object Type and an Object Instance number, the Property shall be addressed with the Property Identifier (PID). The content of the data part depends on the Property function.

The Management Server shall confirm the request with a M_FuncPropCommand.con message. The data part of the confirmation is also dependant from the function called.

Message Code	Interface Object Type		Object Instance	Property Identifier	Data
MC	**IOTH**	**IOTL**	**OI**	**PID**	**Data**
1 octet	2 octets		1 octet	1 octet	n octets
F8h	…		≥ 1	…	…

Figure D.9 — M_FuncPropCommand.req message

D.1.1.7.4.2 M_FuncPropCommand.con

The Management Server shall confirm an M_FuncPropCommand.req message with an M_FuncPropCommand.con message. This response shall contain the return_code and additional data, which are both dependent on the specific Property Function that is called.

Message Code	Interface Object Type		Object Instance	Property Identifier	return_-code	Data
MC	**IOTH**	**IOTL**	**OI**	**PID**	**Ret**	**Data**
1 octet	2 octets		1 octet	1 octet	1 octet	n octets
FAh	as in req.		as in req.	as in req.	xx	…

Figure D.10 — M_FuncPropCommand.con message

NOTE The M_FuncPropCommand.con message and the M_FuncPropStateRead.con message have the same message code and -format.

D.1.1.7.4.3 M_FuncPropStateRead.req

The M_FuncPropStateRead.req message shall be applied by the Management Client to perform a status read on a Property Function of an Interface Object in the Management Server. The Interface Object of the partner shall be addressed with an Object Ype and an Object Instance number, the Property shall be addressed with the Property Identifier (PID). The contents of the data part depends on the Property Function.

The Management Server shall confirm the request with an M_FuncPropStateRead.con message. The data part of the confirmation is also dependant from the function called.

Message Code	Interface Object Type		Object Instance	Property Identifier	Data
MC	**IOTH**	**IOTL**	**OI**	**PID**	**Data**
1 octet	2 octets		1 octet	1 octet	n octets
F9h	…		≥ 1	…	…

Figure D.11 — M_FuncPropStateRead.req message

D.1.1.7.4.4 M_FuncPropStateRead.con

The Management Server shall respond to an M_FuncPropStateRead.req message with an M_FuncPropStateRead.con message.

This M_FuncPropStateRead.con shall contain the return_code and additional data, which are both dependent on the specific Function Property of which the state is read.

Message Code	Interface Object Type		Object Instance	Property Identifier	return_-code	Data
MC	**IOTH**	**IOTL**	**OI**	**PID**	**Ret**	**Data**
1 octet	2 octets		1 octet	1 octet	1 octet	n octets
FAh	as in req.		as in req.	as in req.	xx	…

Figure D.12 — M_FuncPropStateRead.con message

NOTE The M_FuncPropCommand.con message and the M_FuncPropStateRead.con message have the same message code and -format.

D.1.1.7.4.5 Error and exception handling for cEMI Function Properties

If the Interface Object Property accessed by M_FuncPropCommand.req or by M_FuncPropStateRead.req is not a Function Property, the remote application shall respond with a M_FuncPropCommand.con or M_FuncPropStateRead.con respectively, with empty return_code (i.e. the returned PDU shall not contain the field return_code) and no data (i.e. the returned PDU shall not contain the field data).

In case the remote application is able to successfully call a Function Property, the Function Property shall deliver a return_code in the field return_code. The following rules shall apply for all functions:

— return_code = 00h: function successfully executed, i.e. the return code 00h shall be the indication of the positive result of the function;

— return_code ≠ 00h: error.

Error codes are defined in a function specific way.

In case an Interface Object Property that is a Function Property and that is accessed via M_PropRead.req or M_PropWrite.req, the Application Layer shall respond with an M_PropRead.con or M_PropWrite.con with the standard error handling.

D.1.1.7.5 Further cEMI services for local device management

D.1.1.7.5.1 M_Reset.req

The M_Reset.req message shall be used to restart the cEMI Server device on initiation by the cEMI client. For a simple data interface device, this is without any application Interface Objects, restart means that the cEMI Server device executes the same actions as after a power up.

Table D.34 — M_Reset.req message

Message Code
MC
F1h

Reception of this message in the cEMI Server device shall be enabled at any time and shall lead to a re-initialisation of the cEMI Server software. This means that the communication on bus and cEMI side shall be aborted and both communication stacks shall be completely reset.

D.1.1.7.5.2 M_Reset.ind

The M_Reset.ind message shall be used to indicate to the cEMI client a reset or start-up of the cEMI Server device.

A M_Reset.ind shall be sent from the cEMI Server to the cEMI client after any of the following events:

— a M_Reset.req is received from the cEMI client and execution of this request is done;

— a bus power down-/up cycle, only in case the cEMI Server device is powered from the bus;

— after a power up of the cEMI Server device by any other reason, e.g. disconnecting/connecting cycle of the connection between cEMI Server and client, if also powering the cEMI device (e.g. USB).

Table D.35 — M_Reset.ind message

Message Code
MC
F0h

Exceptions

— M_Reset.ind is not mandatory for a USB powered device due to reasons on USB protocol/management level.

— M_Reset.ind is not mandatory for a KNXnet/IP server device as the execution of the reset will break the KNXnet/IP connection to the KNXnet/IP Client and the KNXnet/IP standard does not include automatic reconnecting to the KNXnet/IP client by the KNXnet/IP server.

NOTE 1 A M_Reset.con service is not applicable as the response/confirmation after a M_Reset.req. After a reset, a (simple) device does not know by what reason the reset is caused: by the M_Reset.req or any other reason. Therefore, the M_Reset.con is not applicable and therefore it is also not specified.

NOTE 2 An A_Reset.req from the bus will be routed to the cEMI client within a L_Data-message. It is in the responsibility of the cEMI client, what has to be done after reception of such a M_Reset.req, i.e. to send a M_Reset.req to the cEMI Server or not.

D.1.2 Common EMI: Local Device Management

D.1.2.1 Introduction

Management and operation of the cEMI Server are defined with the approach to be as stateless as possible.

Management and operation messages to and from the cEMI Server can be interlaced, without switching the mode, e.g. from a “management mode” to any communication layer or vice versa.

A generic management interface based on Interface Objects is chosen for the management of the cEMI-Server.

These are some examples for local device management functions.

— Obtaining information from the connected device, e.g. Individual Address, Domain Address, maximum APDU-Length, connected medium, state of the device, possible communication modes of the device.

— Setting of the Individual Address.

— Setting of the Domain Address.

— Setting of the device’s communication layer mode; e.g. raw data (Busmonitor mode), Data Link Layer, Transport Layer.

D.1.2.2 Generic management based on Interface Objects

D.1.2.2.1 Introduction

For the generic management interface, services with own message codes are used to address the cEMI Server’s „local” Interface Objects and Properties. Access to Properties shall be done with M_PropRead/Write/Info services.

The following subclauses give an overview on the Interface Object Types (and their Properties) that can be used for cEMI Server management.

This document does not specify which Properties are mandatory or optional. This depends on the use of the cEMI protocol (e.g. the host protocol, the connected KNX medium, etc.).

For each (open) cEMI Server implementation, the supported Properties shall be stated in a corresponding cEMI Server Profile document.

Device Object

Some of the relevant local management Properties are placed within the Device Object. The Device Object is one of the System Interface Objects.

For cEMI Server local management, the Device Object may hold the Properties.

Table D.36 — Properties in the Device Object for the cEMI Server

Property name	PID	Description, remark
PID_SERVICE_CONTROL	8	Device flags
PID_SERIAL_NUMBER	11	KNX Serial Number
PID_DEVICE_CONTROL	14	Device flags
PID_MAX_APDULENGTH	56	Maximum APDU-Length (for extended frame format)
PID_SUBNET_ADDR	57	To manage the Individual Address (Subnetwork Address part)
PID_DEVICE_ADDR	58	To manage the Individual Address (Device Address part)
PID_DOMAIN_ADDR	70	Domain Address of a PL medium (cEMI Server) device. The value of the Property shall be the Domain Address of the cEMI Server device itself, this is if it should be in only one domain as a management server, seen from the bus media
PID_IO_LIST	71	List of Interface Objects in the (cEMI Server) device

This lists only the cEMI Server relevant Properties of the Device Object.

D.1.2.2.2 PID_IO_LIST - Object scan mechanism

For cEMI Server devices supporting more than the minimum required two Interface Objects (Device Object and cEMI Server Object), it shall be possible to scan the available Interface Objects.

EXAMPLE It shall be possible for a tool to check which Interface Objects are located in the device.

To this purpose, the Property PID_IO_LIST in the Device Object shall be used.

The Property PID_IO_LIST is mandatory for cEMI Server devices supporting other Interface Objects than only the Device Objects and the cEMI Server Object.

The cEMI client shall read out the present Interface Objects with the Local Device Management services. If the Property is not present, then only the Device Object and the cEMI Server Object shall be present in the cEMI Server device.

D.1.2.2.3 cEMI Server Object

D.1.2.2.3.1 Overview

For cEMI Server local management, the cEMI Server Object may hold the Properties.

Table D.37 — Properties in the cEMI Server Object for the cEMI Server

Property Name	PID	Description, remark
PID_MEDIUM_TYPE	51	Media Type(s) supported by cEMI Server
PID_COMM_MODE	52	Data Link Layer / Raw (Busmonitor) / Transport L.
PID_MEDIUM_AVAILABILITY	53	Bus available (1) or not (0) ?
PID_ADD_INFO_TYPES	54	cEMI supported Additional Information Types
PID_TIME_BASE	55	Time base used in Extended relative timestamp.
PID_TRANSP_ENABLE	56	LL Transparency Mode of cEMI Server
PID_CLIENT_SNA	57	Reserved for cEMI Client's Subnetwork Address.
PID_CLIENT_DEVICE_ADDRESS	58	Reserved for cEMI Client's Device Address.
reserved	61	reserved

D.1.2.2.3.2 Communication Mode (Data Link Layer/Raw/Transport Layer) (PID_COMM_MODE) (PID = 52)

D.1.2.2.3.2.1 General requirements

— Property name: Communication Mode

— Datapoint Type: DPT_CommMode (DPT_ID = 20.1000)

The Property PID_COMM_MODE shall control the communication mode of the cEMI Server.

The Management Client shall set PID_COMM_MODE to change the communication mode of the cEMI Server. It shall also read PID_COMM_MODE to learn the current communication mode of the cEMI Server.

A change of the communication mode in the cEMI Server shall change the presentation of the frames from the cEMI Server to the cEMI Client. Every cEMI frame shall contain a Message Code; this Message Code shall be related 1 to 1 with the Communication Mode, except for the "Services for Local device management" (M_PropRead, M_PropWrite).

EXAMPLE PID_COMM_MODE shall control whether a telegram received from the bus shall be presented to the cEMI Client as an L_Raw.ind or a L_Data.ind message.

In the direction from cEMI Client to the bus, the value of PID_COMM_MODE shall not affect anything. The 'communication mode' shall be included in every message received from the cEMI client: the communication layer is indicated by the used message code.

After a power on or a reset, the cEMI Server device may be in an undefined communication mode. If the cEMI Client prior to any other communication to or from the bus does not set the Property Communication Mode, then the behaviour of the cEMI Server is undefined.

If the Property is not present, the cEMI server shall only support Data Link Layer communication mode (this means it shall support the L_Data service) as the minimum requirement.

The value of the Property PID_COMM_MODE shall be formatted as an 8 bit enumeration datatype, identified as DPT_CommMode.

The interpretation of PID_COMM_MODE depends on the device in which the cEMI Server is hosted and is specified in the following clauses.

Table D.38 — Interpretation of DPT_CommMode of PID_COMM_MODE

Value	Communication Mode	Dest. Layer	Description
00h	Data Link Layer	LL	Data Link Layer
01h	Data Link Layer Busmonitor	LLB	Busmonitor
02h	Data Link Layer Raw Frames	LLR	Data Link Layer Raw Frames
03h	not used	…	reserved for Network Layer
04h	not used	…	reserved for TL group oriented
05h	not used	…	reserved for TL connection oriented
06h	cEMI Transport Layer		establishes a connection to the cEMI Transport Layer
07h to EFh	not used	…	reserved for other ‘destination layers’
F0h to FEh	reserved		reserved for manufacturer specific use
FFh	"no layer"	No Layer	allowed as initial value of communication layer after a power up or after an M_Reset

D.1.2.2.3.2.2 cEMI communication mode "cEMI Transport Layer"

cEMI Client sets PID_COMM_MODE to "cEMI Transport Layer"

The cEMI Client shall use the cEMI communication mode “cEMI Transport Layer” to activate the cEMI Transport Layer in the cEMI Server and establish a cEMI Transport Layer connection.

D.1.2.2.3.3 cEMI Client Individual Address

Properties with PID = 57 and PID = 58 are reserved for future use as the cEMI client’s Individual Address (client SNA and Client Device Address), which may be needed to be held as copy in the cEMI Server.

D.1.2.2.4 Address Filtering

D.1.2.2.4.1 Group Address Filter Table Object(s)

Filtering of (logical) Destination Addresses is an optional cEMI feature. Filtering in this context means, only frames in the direction from bus to cEMI client shall be filtered.

If filtering is supported, the address filter table(s) shall be structured as the Interface Objects used in the Routers.

If filtering is not supported, all group frames shall be sent to the cEMI client.

A cEMI Server to/from media with time critical L2-acknowledge (e.g. TP1) and without multicast address filtering shall send a non-selective L2-acknowledge to all multicast-addressed frames (AT-bit set to 1), i.e. independent of the multicast Destination Address.

If an address filter table Interface Object (for Group Addresses) is present in the cEMI Server device, the corresponding Properties provided by these Interface Objects shall be used for features like ‘blocking of addresses in one direction’ or other.

(Default) Filter Mechanism

Filter algorithms shall be the same as for the filtering in Routers.

Addresses entered in the Filter Table shall be passed; all other addresses shall not be passed.

Annex E
(normative)

Coupler Resources

E.1 Introduction

A KNX Coupler shall be usable as a TP Line Coupler, as a TP Backbone Coupler or as a TP Repeater. The Individual Address of the device shall determine the mode. Additional settings that specify the mode shall be possible with the Interface Objects.

This Annex E specifies standard Interface Objects and Properties for memory independent Coupler management.

Values in the column “default” are those property values in the ex-factory state and after a master reset of the device. The KNX Coupler should have a master reset facility.

E.2 Device Object

Table E.1

Property Identifier (PID)	Access R/W [a]	Access via NP services [b]	Req.	Value
1= PID_OBJECT_TYPE	3/-			DEVICE_OBJECT: 0000h
8= PID_SERVICE_CONTROL	3/0		O	Permanent control field of the device
9= PID_FIRMWARE_REVISION	3/-		O	revision number of the firmware
11= PID_SERIAL_NUMBER	3/0		M	serial number
12= PID_MANUFACTURER_ID	3/0		M	manufacturer identifier
14= PID_DEVICE_CONTROL	3/0		O	temporary control field for the device
15= PID_ORDER_INFO	3/0		O	
19= PID_MANUFACTURER_DATA	3/0		O	
51= PID_ROUTING_COUNT	3/0		M	default routing counter
53= PID_ERROR_FLAGS	3/0		O	error flags
54= PID_PROGMODE	3/0		O	00h
56= PID_MAX_APDULENGTH	3/-		O	55
57= PID_SUBNET_ADDR		X	M	SubNetAddress

[a] Used access level for Read/Write access. 0 means the highest level and 3 means the lowest (default) access level where no authorisation is necessary. In case of a dash (‘-‘) in the place of a write access level the property is read only.

[b] X in this column means that access to this property is only possible via the A_NetworkParameter_Read and the A_NetworkParameter_Write-services.

E.2.1 PID_SERVICE_CONTROL (PID = 8)

The property PID_SERVICE_CONTROL shall be a permanent control field of the device.

Table E.2

Bit	Name	Description	Coding	default
0	USER_STOPPED_INFO_EN	User stopped serviceInfo enable	not used	-
1	OWNINDIVIDUAL_ADDRESS_RECEIVED _EN	Enable the generation of a ServiceInfo Report in case of reception of a frame with own Individual Address.	0 = disable 1 = enable	0 = disable
2	INDIVIDUAL_ADDRESS_WRITE_EN	enables or disables the setting of the Individual Address via program mode or serial number.	0 = disable 1 = enable	1 = enable
3-15		reserved	shall be 0	

E.2.2 PID_DEVICE_CONTROL (PID = 14)

The property PID_DEVICE_CONTROL shall be a temporary control field of the device

Table E.3

Bit	Name	Description	Coding	default
0	USER_STOPPED	If user stopped → message will be triggered	not used	-
1	OWNINDIVIDUAL_ADDRESS	if frame with own physical address received → message will be triggered	0 = default 1 = trigger	n.a. (temporary)
2	VERIFY_MODE	verify mode on	0 = no verify 1 = verify	n.a. (temporary)
3-7		reserved		

E.2.3 PID_ROUTING_COUNT (PID = 51)

The property PID_ROUTING_COUNT shall be the standard value of the routing counter sent by the Coupler Network Layer (not the routed telegrams).

E.2.4 PID_ERROR_FLAGS (PID = 53)

Table E.4

Bit	Name	Description	Coding	default
0	SYSTEM	internal system error: message buffer corrupt → forces a system reset	0 = error 1 = ok	1 = ok
1	ILLEGAL_SFR	Illegal value of the Special Function Register (SFR)	0 = error 1 = ok	1 = ok
2	MEMORY_ERROR	verify error after flash write operation	0 = error 1 = ok	1 = ok
3	STACK	stack overflow → forces a system reset	0 = error 1 = ok	1 = ok
4		reserved		Must be 1
5	TRANS	transceiver error (no transmission until system reset)	0 = error 1 = ok	1 = ok
6	SYSTEM2	Internal system error (warning only)	0 = error 1 = ok	1 = ok
7	SYSTEM3	Internal system error (warning only)	0 = error 1 = ok	1 = ok

E.2.5 PID_PROGMODE (PID = 54)

Table E.5

Bit	Name	Description	Coding	default
0	PROGMODE	set the device in programming mode	0 = no 1 = progmode	0
1-7		reserved		

E.3 Router Object

E.3.1 General

All properties of this Interface Object shall be used both in Line Coupler as in Backbone Coupler mode and as well in Repeater mode.

Table E.6

Property ID	Access R/W	Access via NP services	Req.	Value
1 = PID_OBJECT_TYPE	3/-	X[a]	M	ROUTER_OBJECT: 0006h
5 = PID_LOAD_STATE_CONTROL	3/0		M	See clause E.3.1 "PID_LOAD_STATE_CONTROL (PID = 5)"
51 = PID_LINE_STATUS	3/-	X[a]	M	Status of the subline
52 = PID_MAIN_LCCONFIG	3/0		O	Defines the handling of individually addressed and broadcast frames from main line
53 = PID_SUB_LCCONFIG	3/0		O	Defines the handling of individually addressed and broadcast frames from sub line
54 = PID_MAIN_LCGRPCONFIG	3/0		O	Defines the handling of group addressed frames from main line
55 = PID_SUB_LCGRPCONFIG	3/0		O	Defines the handling of group addressed frames from sub line
56 = PID_ROUTETABLE_CONTROL	3/0		O	Set of methods to modify the routing table
57 = PID_COUPL_SERV_CONTROL	3/0		M	Inconsistency and Subnetwork Address (SNA) mechanisms
[a] Access shall also be possible via property services.				

E.3.2 PID_LOAD_STATE_CONTROL (PID = 5)

The load state machine in the Router Object shall control the access to the standard routing table.

The following load controls shall be supported:

Table E.7

load control	coded
EV_NOP	00h
EV_START_LOAD	01h
EV_LOAD_COMPLETE	02h
EV_UNLOAD	04h

The following load states may be returned:

Table E.8

load state	coded
LS_UNLOADED	00h
LS_LOADED	01h
LS_LOADING	02h
LS_ERROR	03h

If the Group Address filter table should be checked on received frames and the load state is not LS_LOADED at the same time, no frames shall be routed.

Upon the event EV_UNLOAD the state machine is in the state LS_UNLOADED. It shall not be relied on the fact that the filter table is erased in memory. PID_ROUTETABLE_CONTROL serves to erase the filter table in memory.

The default load state shall be LS_LOADED and the routing table cleared.

E.3.3 PID_LINE_STATUS (PID = 51)

This property shall include the status of the subline.

Table E.9

Bit	Name	Description	Coding
0	POWER_DOWN_SUBLINE	report a power down in the subline	0 = power up subline 1 = power down sub line
1-7		reserved	

The A_NetworkParameter_Write service into the primary side with object_type = GroupAddress FilterObject and PID = PID_SUBLINE_STATUS shall be used in broadcast communication mode and shall be triggered due to a power down or a power restart on the subline once only.

E.3.4 PID_MAIN_LCCONFIG (52) / PID_SUB_LCCONFIG (PID = 53)

Two properties (for each line one property) shall affect the handling of frames in point to point (connectionless and connection oriented) or broadcast communication mode.

Table E.10

Bit	Name	Description	Coding	Default
0-1	PHYS_FRAME	Specifies the static handling for receiving individually addressed frames	0 = not used 1 = PHYS_UNLOCK: all individually addressed frames shall be routed, **independent of the coupler mode!** 2 = PHYS_LOCK: no frames shall be routed, **independent of the coupler mode!** 3 = PHYS_ROUT: routing depends on destination address, coupler address and the coupler mode (normal operation mode)	3
2	PHYS_REPEAT	Repetition of individually addressed frames in case of transmission errors	**Independent of the coupler mode!** 0 = no repetitions 1 = frames will be repeated in case of transmission errors (up to 6 times for BUSY acknowledged frames and up to 3 times for other acknowledged or not acknowledged frames)	1
3	BROADCAST_LOCK	switch ON or OFF the routing of broadcast addressed frames	**independent of the coupler mode:** 0 = normal: broadcast frames will be routed 1 = broadcast frames will be locked (useable during system configuration)	0
4	BROADCAST_REPEAT	repetition of broadcast addressed frames in case of transmission errors	**independent of the coupler mode:** 0 = no repetition 1 = frames will be repeated in case of transmission errors (up to 6 times for BUSY acknowledged frames and up to 3 times for other acknowledged or not acknowledged frames)	1
5	GROUP_IACK_ROUT	specifies the immediate acknowledge of received group addressed frames Switch ON or OFF the immediate acknowledge	**independent of the coupler mode:** 0 = all group addressed frames will be acknowledged, independent of the routing! (useful only to avoid the repetitions of misrouted frames) 1 = normal mode (all frames which will be routed will also be acknowledged)	1
6-7	PHYS_IACK	specifies the immediate acknowledge of received individually addressed frames Switch ON or OFF the immediate acknowledge	**independent of the coupler mode:** 0 = not used 1 = normal mode (all frames which will be routed or which are addressed to the Line Coupler itself will be acknowledged) 2 = all frames will be acknowledged (useful only to avoid the repetitions of misrouted frames) 3 = INACK for all frames, protection (useful to prevent all parameterisation in one line, the coupler is protected too. A typical use case is the protection of a subline, which is located outside a building)	1

E.3.5 PID_MAIN_LCGRPCONFIG (54) / PID_SUB_LCGRPCONFIG (PID = 55)

Two properties (for each line one property) affect the handling of group addressed frames (independent of the coupler mode!).

Table E.11

Bit	Name	Description	Coding	Default
0-1	GROUP_6FFF	specify the handling of group addressed frames ≤ 6FFFh	0 = not used 1 = GROUP_UNLOCK6FFF All frames shall be routed. **(Typical for repeater mode.)** 2 = GROUP_LOCK6FFF No frames shall be routed. 3 = GROUP_ROUT6FFF Routing shall depend on routing table.	3
2-3	GROUP_7000	specify the handling of group addressed frames ≥ 7000h	0 = not used 1 = GROUP_UNLOCK7000 All frames shall be routed. **(Typical for repeater mode.)** 2 = GROUP_LOCK7000 No frames shall be routed. 3 = GROUP_ROUT7000 Routing shall depend on routing table.	1
4	GROUP_REPEAT	repetition of group addressed frames in case of transmission errors	0 = No repetition. 1 = Frames shall be repeated in case of transmission errors (up to 6 times for BUSY acknowledged frames and up to 3 times for other acknowledged or not acknowledged frames).	1
5-7		not used		

E.3.6 PID_ROUTETABLE_CONTROL (PID = 56)

E.3.6.1 General

This property shall be used to handle the routing (filter) table for standard KNX Group Addresses. Due to the property type PDT_Function the configuration is much faster and more flexible than the standard access to a fixed routing address field.

Table E.12 — Structure of the property function for reading or writing

Octet:	10	11	12	13…21
	00h / return_code	ServiceID	ServiceInfo1	ServiceInfo2 ... ServiceInfo11

The ServiceID specifies the selected method. For some methods additional ServiceInfos are necessary.

E.3.6.2 ServiceID: 1 (SRVID_CLEAR_ROUTINGTABLE)

Table E.13

Octet:	10	11
	00h	01h

— No additional Infos are used.

— used with A_FunctionPropertyCommand-PDU: function clears all addresses in the filter table;

— used with A_FunctionPropertyState_Read-PDU: function checks if all addresses in the filter table are cleared.

Table E.14 — Response A_PropertyValue_Response PDU

Octet:	10	11
	Return_code	1

— After A_FunctionPropertyCommand PDU:

  — return_code FFh means error occurs during clearing filter table (verify error);

  — return_code 00h means clearing filter table is successfully done.

— After A_FunctionPropertyState_Read PDU:

  — return_code FFh means not all addresses in the filter table are cleared;

  — return_code 00h means all addresses in the filter table are cleared successfully.

E.3.6.3 ServiceID: 2 (SRVID_SET_ROUTINGTABLE)

Table E.15

Octet:	10	11
	00h	02h

— No additional Infos are used:

— Used with A_FunctionPropertyCommand PDU: function sets all addresses in the filterable;

— Used with A_FunctionPropertyState_Read PDU: function checks if all addresses in the filterable are set.

Table E.16 — Response A_PropertyValue_Response PDU

Octet:	10	11
	Return_code	2

— After A_FunctionPropertyCommand PDU:

  — return_code FFh means error occurs during setting filter table (verify error);

  — return_code 00h means setting filter table is successfully done.

— After A_FunctionPropertyState_Read PDU:

  — return_code FFh means not all addresses in the filter table are set;

— return_code 00h means all addresses in the filter table are set successfully.

E.3.6.4 ServiceID: 3 (SRVID_CLEAR_GROUPADDRESS)

Table E.17

Octet:	10	11	12	13	14	15
	00h	3	START ADDRESS HIGH OCTET	START ADDRESS LOW OCTET	END ADDRESS HIGH OCTET	END ADDRESS LOW OCTET

— Used with A_FunctionPropertyCommand PDU:

— START ADDRESS is the first address to be modified.

— END ADDRESS is the last address to be modified.

— If only one address is to be modified then START ADDRESS and ENDADDRESS is equal.

— If START ADDRESS is greater than END ADDRESS no address will be modified (error response).

— Used with A_FunctionPropertyState_Read PDU:

— START ADDRESS is the first address to be checked.

— END ADDRESS is the last address to be checked.

— If only one address is to be checked then START ADDRESS and ENDADDRESS is equal!

— If START ADDRESS is greater than END ADDRESS no address will be checked.

Table E.18 — Response A_PropertyValue_Response PDU

Octet:	10	11	12	13	14	15
	Return_code	3	START ADDRESS HIGH OCTET	START ADDRESS LOW OCTET	END ADDRESS HIGH OCTET	END ADDRESS LOW OCTET

— After A_FunctionPropertyCommand PDU:

— return_code FFh means error occurs during clearing address(es) (verify error);

— return_code 00h means clearing addresses is successfully done.

— After A_FunctionPropertyState_Read PDU:

— return_code FFh means not all tested addresses are cleared;

— return_code 00h means all tested addresses are successfully cleared.

E.3.6.5 ServiceID: 4 (SRVID_SET_GROUPADDRESS)

Table E.19

Octet:	10	11	12	13	14	15
	00h	4	START ADDRESS HIGH OCTET	START ADDRESS LOW OCTET	END ADDRESS HIGH OCTET	END ADDRESS LOW OCTET

— Used with A_FunctionPropertyCommand PDU:

  — START ADDRESS is the first address to be modified.

  — END ADDRESS is the last address to be modified.

  — If only one address is to be modified then START ADDRESS and ENDADDRESS is equal.

  — If START ADDRESS is greater than END ADDRESS no address will be modified (error response).

— Used within A_FunctionPropertyState_Read PDU:

  — START ADDRESS is the first address to be checked.

  — END ADDRESS is the last address to be checked.

  — If only one address is to be checked then START ADDRESS and ENDADDRESS is equal.

  — If START ADDRESS is greater than END ADDRESS no address will be checked.

Table E.20 — Response A_PropertyValue_Response PDU

Octet:	10	11	12	13	14	15
	Return_code	4	START ADDRESS HIGH OCTET	START ADDRESS LOW OCTET	END ADDRESS HIGH OCTET	END ADDRESS LOW OCTET

— Used after A_FunctionPropertyCommand PDU:

  — return_code FFh means error occurs during setting address(es) (verify error);

  — return_code 00h means setting addresses is successfully done.

— Used after A_FunctionPropertyState_Read PDU:

  — return_code FFh means not all tested addresses are be set;

  — return_code 00h means all tested addresses are successfully be set.

E.3.6.6 Error handling

— The reaction to an unknown service ID is defined as follows:

Table E.21

Octet:	10	11	...
	00h	undefined	...

— used within A_FunctionPropertyCommand-PDU or A_PropertyValue_Read-PDU.

Table E.22 — Response A_PropertyValue_Response PDU

Octet:	10	11
	Return_code	undefined

— return_code FFh: error.

E.3.7 PID_COUPL_SERV_CONTROL (PID = 57)

Table E.23

Bit	Name	Description	Coding	Default
0	EN_SNA_INCONSISTENCY_CHECK	Enables the inconsistency-check of individually addressed frames: possible trigger of a SubnetAddressWrite This feature will only work the coupler is not in repeater mode!	0 = disable 1 = enable	0
1	EN_SNA_HEARTBEAT	Enables the heartbeat: SNA_UpdateWrite into subline (only useful if coupler is not in repeater mode)	0 = disable 1 = enable	0
2	EN_SNA_UPDATE_WRITE	Enables SNA_UpdateWrite after changing physical address via program button or serial number (only useful if coupler is not in repeater mode)	1 = enable 0 = disable	0
3	EN_SNA_READ	Enables SNA_Read functionality. If this feature is enabled, the SNA_Read service from the subline will be accepted and a SNA Response will be generated (this feature doesn't work in repeater mode, in repeater mode the service will be always ignored, independently of the property setting)	0 = disable 1 = enable	1
4	EN_SUBLINE_STATUS	Enables a SUB_LineStatus after power failure or power restart in subline. (independent of the coupler mode)	0 = disable 1 = enable	0
5 - 7	reserved			

EN_SNA_INCONSISTENCY_CHECK

— Feature is automatically disabled in repeater mode;

— the Line Coupler detects inconsistencies in source Subnetwork Address (SNA) filed of all individually addressed frames (point to point);

— messages with inconsistent source SNA will be acknowledged but won't be routed and either;

  — an A_NetworkParameter_Write (SNA Update) in broadcast communication mode on secondary side (sub line) with local SNA will be triggered if the inconsistent SNA originated from the sub line or;

  — an A_NetworkParameter_Write (SNA Update) in point to point communication mode on primary side (main line) with SNA 0.0 will be triggered if the inconsistent SNA originated from the main line.

EN_SNA_READ (indication)

— Service may be used by new devices on the subnetwork to get the actual SNA.

— If service is received from coupler secondary side (sub line), it will be answered.

— If service is received from coupler primary side (main line), it will be ignored.

  Exception on the backbone line: A response containing the SNA 0.0 is also generated on the primary side of Routers with SNA x.0 (x > 0). This is, in a hierarchical network with several main lines N responses with the information SNA 0.0 will be generated by the N routers which are connected to the backbone line.

— If service is received from repeater (coupler works in repeater mode), it will be ignored.

Bibliography

[1] Postel, J., *"User Datagram Protocol"*, STD 6, RFC 768, USC Information Sciences Institute, August 1980 http://www.ietf.org/rfc.html

[2] Postel, J., *"Transmission Control Protocol"*, STD 7, RFC 793, USC Information Sciences Institute, September 1981 http://www.ietf.org/rfc.html

[3] Postel, J., *"Internet Protocol"*, STD 5, RFC 791, USC Information Sciences Institute, September 1981 http://www.ietf.org/rfc.html

[4] Croft, W.J., Gilmore, J., *"Bootstrap Protocol"*, RFC 951, September 1985 http://www.ietf.org/rfc.html

[5] Droms, R., *"Dynamic Host Configuration Protocol"*, RFC 2131, Bucknell University, March 1997 http://www.ietf.org/rfc.html

[6] EN 13321-1, *Open data communication in building automation, controls and building management — Home and building electronic system — Part 1: Product and system requirements*

[7] ISO/IEC 8859-1:1998 *Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*

[8] EN ISO 16484-5, *Building automation and control systems — Part 5: Data communication protocol (ISO 16484 - 5)*

This page deliberately left blank

British Standards Institution (BSI)

BSI is the national body responsible for preparing British Standards and other standards-related publications, information and services.

BSI is incorporated by Royal Charter. British Standards and other standardization products are published by BSI Standards Limited.

About us

We bring together business, industry, government, consumers, innovators and others to shape their combined experience and expertise into standards -based solutions.

The knowledge embodied in our standards has been carefully assembled in a dependable format and refined through our open consultation process. Organizations of all sizes and across all sectors choose standards to help them achieve their goals.

Information on standards

We can provide you with the knowledge that your organization needs to succeed. Find out more about British Standards by visiting our website at bsigroup.com/standards or contacting our Customer Services team or Knowledge Centre.

Buying standards

You can buy and download PDF versions of BSI publications, including British and adopted European and international standards, through our website at bsigroup.com/shop, where hard copies can also be purchased.

If you need international and foreign standards from other Standards Development Organizations, hard copies can be ordered from our Customer Services team.

Subscriptions

Our range of subscription services are designed to make using standards easier for you. For further information on our subscription products go to bsigroup.com/subscriptions.

With **British Standards Online (BSOL)** you'll have instant access to over 55,000 British and adopted European and international standards from your desktop. It's available 24/7 and is refreshed daily so you'll always be up to date.

You can keep in touch with standards developments and receive substantial discounts on the purchase price of standards, both in single copy and subscription format, by becoming a **BSI Subscribing Member**.

PLUS is an updating service exclusive to BSI Subscribing Members. You will automatically receive the latest hard copy of your standards when they're revised or replaced.

To find out more about becoming a BSI Subscribing Member and the benefits of membership, please visit bsigroup.com/shop.

With a **Multi-User Network Licence (MUNL)** you are able to host standards publications on your intranet. Licences can cover as few or as many users as you wish. With updates supplied as soon as they're available, you can be sure your documentation is current. For further information, email bsmusales@bsigroup.com.

BSI Group Headquarters

389 Chiswick High Road London W4 4AL UK

Revisions

Our British Standards and other publications are updated by amendment or revision.

We continually improve the quality of our products and services to benefit your business. If you find an inaccuracy or ambiguity within a British Standard or other BSI publication please inform the Knowledge Centre.

Copyright

Useful Contacts:

Customer Services
Tel: +44 845 086 9001
Email (orders): orders@bsigroup.com
Email (enquiries): cservices@bsigroup.com

Subscriptions
Tel: +44 845 086 9001
Email: subscriptions@bsigroup.com

Knowledge Centre
Tel: +44 20 8996 7004
Email: knowledgecentre@bsigroup.com

Copyright & Licensing
Tel: +44 20 8996 7070
Email: copyright@bsigroup.com

bsi.

...making excellence a habit.™