

NORMA
EUROPEA

**Comunicazione aperta dei dati per l'automazione, la
regolazione e la gestione tecnica degli edifici - Sistemi
elettronici per le case e gli edifici - Parte 2:
Comunicazione KNXnet/IP**

**UNI EN ISO
22510**

APRILE 2020

Open data communication in building automation, controls and
building management - Home and building electronic systems -
KNXnet/IP communication

La norma definisce l'integrazione delle implementazioni del
protocollo KNX con le reti IP (Internet Protocol) e descrive un
protocollo unificato per i dispositivi KNX connessi ad una rete IP,
definiti dispositivi KNXnet/IP. La rete IP opera (se paragonata con
la velocità di trasmissione KNX) come dorsale (backbone) veloce
nelle installazioni KNX.

TESTO INGLESE

La presente norma è la versione ufficiale in lingua inglese della
norma europea EN ISO 22510 (edizione febbraio 2020).

La presente norma sostituisce la UNI EN 13321-2:2013.

ICS 35.240.67; 91.040.01

PREMESSA NAZIONALE

La presente norma costituisce il recepimento, in lingua inglese, della norma europea EN ISO 22510 (edizione febbraio 2020), che assume così lo status di norma nazionale italiana.

La presente norma è stata elaborata sotto la competenza dell'ente federato all'UNI

CTI - Comitato Termotecnico Italiano

La presente norma è stata ratificata dal Presidente dell'UNI ed è entrata a far parte del corpo normativo nazionale il 9 aprile 2020.

Le norme UNI sono elaborate cercando di tenere conto dei punti di vista di tutte le parti interessate e di conciliare ogni aspetto conflittuale, per rappresentare il reale stato dell'arte della materia ed il necessario grado di consenso.

Chiunque ritenesse, a seguito dell'applicazione di questa norma, di poter fornire suggerimenti per un suo miglioramento o per un suo adeguamento ad uno stato dell'arte in evoluzione è pregato di inviare i propri contributi all'UNI, Ente Italiano di Normazione, che li terrà in considerazione per l'eventuale revisione della norma stessa.

Si richiama l'attenzione sulla possibilità che alcuni degli elementi del presente documento possono essere oggetto di brevetti. UNI non deve essere ritenuto responsabile di aver citato tali brevetti.

Le norme UNI sono revisionate, quando necessario, con la pubblicazione di nuove edizioni o di aggiornamenti.

È importante pertanto che gli utilizzatori delle stesse si accertino di essere in possesso dell'ultima edizione e degli eventuali aggiornamenti.

Si invitano inoltre gli utilizzatori a verificare l'esistenza di norme UNI corrispondenti alle norme EN o ISO ove citate nei riferimenti normativi.

EUROPEAN STANDARD
NORME EUROPÉENNE
EUROPÄISCHE NORM

EN ISO 22510

February 2020

ICS 35.240.67; 91.040.01

Supersedes EN 13321-2:2012

English Version

Open data communication in building automation, controls
and building management - Home and building electronic
systems - KNXnet/IP communication (ISO 22510:2019)

Réseau ouvert de communication de données pour
l'automatisation, la régulation et la gestion technique
du bâtiment - Systèmes électroniques pour les foyers
domestiques et les bâtiments - Communication KNX/IP
(ISO 22510:2019)

Offene Datenkommunikation für die
Gebäudeautomation und Gebäudemanagement -
Elektrische Systemtechnik für Heim und Gebäude - Teil
2: KNXnet/IP-Kommunikation (ISO 22510:2019)

This European Standard was approved by CEN on 1 December 2019.

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, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Turkey and United Kingdom.



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

CEN-CENELEC Management Centre: Rue de la Science 23, B-1040 Brussels

Contents	Page
European foreword.....	3

European foreword

This document (EN ISO 22510:2020) has been prepared by Technical Committee ISO/TC 205 "Building environment design" in collaboration with 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 August 2020, and conflicting national standards shall be withdrawn at the latest by August 2020.

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

This document supersedes EN 13321-2:2012.

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

Endorsement notice

The text of ISO 22510:2019 has been approved by CEN as EN ISO 22510:2020 without any modification.

INTERNATIONAL STANDARD

**ISO
22510**

First edition
2019-11

Open data communication in building automation, controls and building management — Home and building electronic systems — KNXnet/IP communication

*Réseau ouvert de communication de données pour l'automatisation,
la régulation et la gestion technique du bâtiment — Systèmes
électroniques pour les foyers domestiques et les bâtiments —
Communication KNX/IP*



Reference number
ISO 22510:2019(E)

© ISO 2019



COPYRIGHT PROTECTED DOCUMENT

© ISO 2019

All rights reserved. Unless otherwise specified, or required in the context of its implementation, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Fax: +41 22 749 09 47
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

	Page
Foreword	v
Introduction	vi
1 Scope	1
2 Normative references	1
3 Terms and definitions	1
4 Abbreviated terms	4
5 Requirements	5
5.1 Overview.....	5
5.1.1 KNXnet/IP document parts.....	5
5.1.2 Mandatory and optional implementation of IP protocols.....	7
5.2 Core.....	8
5.2.1 Use.....	8
5.2.2 KNXnet/IP frames.....	9
5.2.3 Host protocol independence.....	10
5.2.4 Discovery and self description.....	11
5.2.5 Communication channels.....	13
5.2.6 General implementation guidelines.....	15
5.2.7 Data Packet structures.....	19
5.2.8 IP Networks.....	38
5.2.9 Minimum supported services.....	47
5.3 Device management specification.....	48
5.3.1 Use.....	48
5.3.2 KNXnet/IP device management.....	48
5.3.3 Implementation rules and guidelines.....	59
5.3.4 Data packet structures.....	60
5.3.5 Minimum profiles.....	63
5.4 Tunnelling.....	64
5.4.1 Use.....	64
5.4.2 Tunnelling of KNX telegrams.....	64
5.4.3 Configuration and management.....	68
5.4.4 Frame structures.....	70
5.4.5 Minimum profiles.....	77
5.5 Routing.....	78
5.5.1 Use.....	78
5.5.2 KNXnet/IP routing of KNX telegrams.....	78
5.5.3 Implementation rules and guidelines.....	88
5.5.4 Configuration and management.....	91
5.5.5 Data packet structures.....	91
5.5.6 Minimum profiles.....	93
5.6 Remote diagnosis and configuration.....	94
5.6.1 Use.....	94
5.6.2 Remote diagnosis of KNXnet/IP devices.....	95
5.6.3 Configuration and management.....	95
5.6.4 Data packet structures.....	96
5.6.5 Certification.....	101
5.7 Secured communication.....	101
5.7.1 Use.....	101
5.7.2 Stack and communication.....	102
5.7.3 Management procedures.....	143
5.7.4 Synchronizing timers.....	146
Annex A (normative) List of codes	148
Annex B (informative) Binary examples of KNXnet/IP frames	155

Annex C (normative) KNXnet/IP parameter object	175
Annex D (normative) Common external messaging interface (cEMI)	178
Annex E (normative) Coupler resources	210
Bibliography	221

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by the European Committee for Standardization (CEN) Technical Committee CEN/TC 247, *Building Automation, Controls and Building Management*, in collaboration with ISO Technical Committee TC 205, *Building environment design*, in accordance with the agreement on technical cooperation between ISO and CEN (Vienna Agreement).

A list of all parts in the ISO 16484 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

This document 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.

KNXnet/IP is a protocol designed to transport KNX home and building electronic system (HBES) control frames over an IP network. It is used as an infrastructure backbone for connecting KNX sub-networks, as a communication medium for KNX-IP devices and to provide IP based services for clients (e.g. connecting a tool software to a KNX installation). The main advantages of using IP for these purposes are that IP network infrastructure is inexpensive, available almost everywhere and that the distance of two communication parties on an IP network is virtually unlimited.

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);
- an interface to super ordinate building management and energy management systems.

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-RF, KNX-PL110);
- 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.).

KNXnet/IP differentiates between unicast and multicast services. KNXnet/IP unicast services are used to connect a single client to a single KNXnet/IP server (e.g. KNXnet/IP Tunnelling). KNXnet/IP multicast services are mainly used to connect different KNX sub-networks using IP communication on the KNX backbone. The KNXnet/IP routing services are defined for this purpose. KNXnet/IP multicast services build on top of IP multicast.

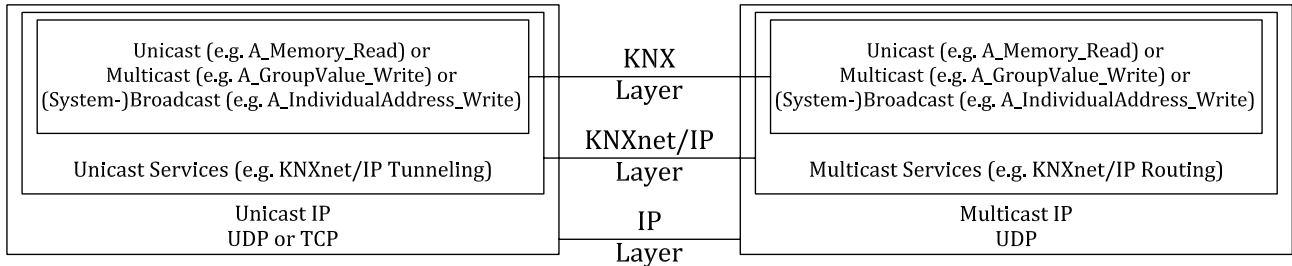


Figure 1 — Unicast and multicast in the sense of KNX, KNXnet/IP and IP

[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.

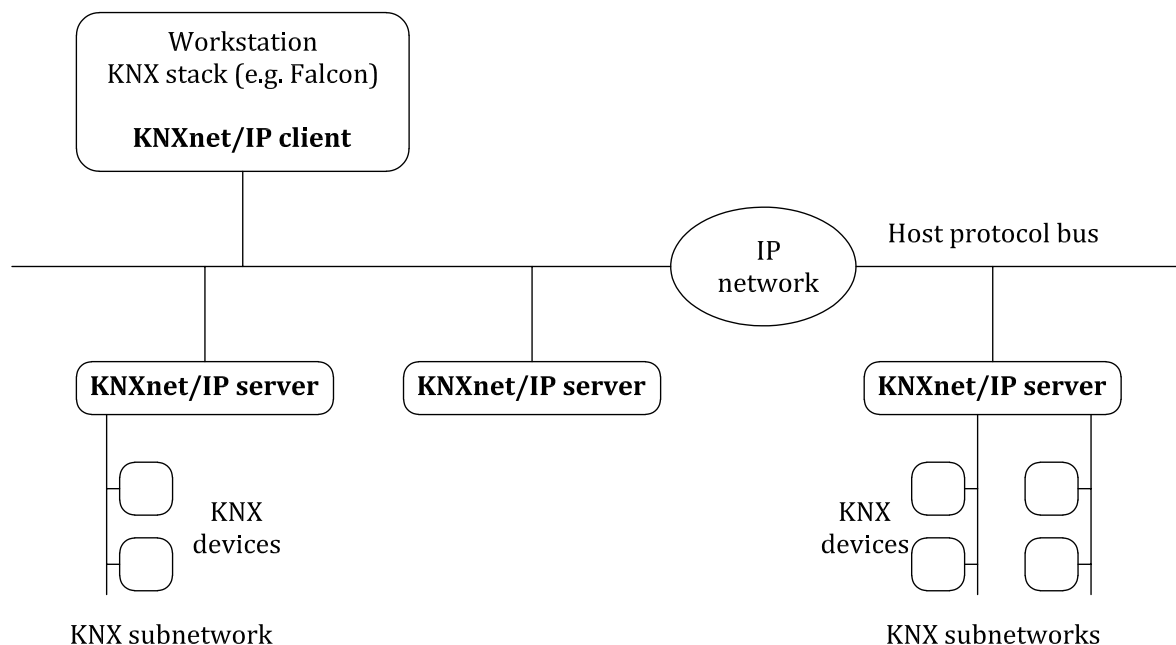


Figure 2 — Device types and configuration examples

[Figure 2](#) shows device types and configuration examples. This document 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.

Open data communication in building automation, controls and building management — Home and building electronic systems — KNXnet/IP communication

1 Scope

This document 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 twisted pair transmission speed) backbone in KNX installations.

2 Normative references

There are no normative references in this document.

3 Terms and definitions

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

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <https://www.iso.org/obp>
- IEC Electropedia: available at <http://www.electropedia.org/>

3.1

backbone key

key used for encryption and message authentication of secure KNXnet/IP multicast communication in a KNXnet/IP routing multicast group, configured by ETS and a shared secret between all members of the secure KNXnet/IP routing multicast group

3.2

cipher

generic term that denotes the encrypted data

Note 1 to entry: Cipher is the opposite of *plain* (3.22).

3.3

common external message interface

generic structure for medium independent *KNX* (3.14) messages

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

3.4

communication channel

logical connection between a *KNXnet/IP client* (3.16) and a *KNXnet/IP server* (3.20) 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 **dynamic host configuration protocol**

communication protocol for automatic assignment of IP address settings

3.6 **domain name system**

assigns Internet names to IP addresses

3.7 **engineering tool software**

software used to configure *KNX* (3.14) devices

3.8 **european installation bus**

predecessor protocol to *KNX*

Note 1 to entry: Standard for building controls (EN 50090).

3.9 **host protocol address information**

structure holding the IP host protocol address information used to address a *KNXnet/IP* endpoint on another *KNXnet/IP* device (3.17)

3.10 **individual address**

unique *KNX* (3.14) address of a *KNX* device in a *KNX* system

3.11 **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.

3.12 **Internet control message protocol**

extension to the Internet protocol (IP) for error, control, and informational messages

Note 1 to entry: ICMP is defined by RFC¹⁾ 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**

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 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 **KNX**

standard for building controls

Note 1 to entry: See EN 50090, ISO/IEC 14543-3-1 to ISO/IEC 14543-3-7.

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

3.15

KNX node

device implementing a *KNX* (3.14) protocol stack and fulfilling the requirements according to the KNX standard

3.16

KNXnet/IP client

application using the KNXnet/IP client protocol to get access to a *KNX* (3.14) subnetwork over an IP network channel

3.17

KNXnet/IP device

implementation of KNXnet/IP services on a *KNX node* (3.15) (*KNXnet/IP server* (3.20)) or any other hardware (*KNXnet/IP client* (3.16))

3.18

KNX/IP domain

all *KNXnet/IP devices* (3.17) in the same network with the same multicast address and the same backbone encryption (either no encryption or encryption with the same key)

3.19

KNXnet/IP router

special type of *KNXnet/IP device* (3.17) that routes *KNX* (3.14) protocol packets between KNX sub-networks

3.20

KNXnet/IP server

KNX (3.14) device with physical access to a KNX network implementing the KNXnet/IP server protocol to communicate with *KNXnet/IP clients* (3.16) 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* (3.15).

3.21

KNXnet/IP tunnelling

services for point-to-point exchange of *KNX* (3.14) telegrams over an IP network between a *KNXnet/IP device* (3.17) acting as a server and a *KNXnet/IP client* (3.16)

3.22

plain

generic term that denotes unencrypted data, of which the content depends on the service and the user and not of confidentiality and authentication

Note 1 to entry: Plain is the opposite of *cipher* (3.2).

3.23

secure session

authenticated, authorized and encrypted *communication channel* (3.4) between one *KNXnet/IP client* (3.16) and one *KNXnet/IP server* (3.20) for unicast communication

3.24

session key

key used for encryption and message authentication in a *secure session* (3.23) between two KNXnet/IP communication parties, created using ECDH in the secure session setup procedure (providing perfect forward secrecy) and only valid for this individual session

3.25

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.

3.26

time to live

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).

3.27

transmission control protocol over Internet protocol

connection-oriented communication over the Internet

3.28

user datagram protocol over Internet protocol

connection-less communication over the Internet

4 Abbreviated terms

Abbreviated term	Description
AddIL	Length of additional information
AES	Advanced Encryption Standard
Cn	Conditions are specified under note “n”
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
cEMI	Common External Message Interface
CTR	Counter Mode (of Operation)
DDoS	Distributed Denial of Service
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DoS	Denial of Service
ECDH	Elliptic Curve Diffie–Hellman
EIB	European Installation Bus
ETS	Engineering Tool Software
FDSK	Factory Default Setup Key
HPAI	Host Protocol Address Information
IA	Individual Address
ICMP	Internet Control Message Protocol
IGMP	Internet Group Management Protocol
IP	Internet Protocol
IV	Initialisation Vector
M	Mandatory
MAC	Message Authentication Code
MC	Message Code
MiM	Man-in-the-Middle
n/a	Not applicable
O	Optional
PBKDF2	Password-Based Key Derivation Function 2
R	Required
SHA	Secure Hash Algorithm

Abbreviated term	Description
TCP/IP	Transmission Control Protocol over Internet Protocol
TTL	Time To Live
X	Not allowed
UDP/IP	User Datagram Protocol over Internet Protocol

5 Requirements

5.1 Overview

5.1.1 KNXnet/IP document parts

5.1.1.1 General

This document defines the integration of KNX protocol implementations within the Internet protocol (IP) named KNXnet/IP.

This document 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 specification consists of the following parts:

- Overview;
- Core specification;
- Device management;
- Tunnelling;
- Routing;
- Remote diagnosis and configuration;
- Secured communication.

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 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 offline, for example generated as an ETS export file, or online 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 $t_{\text{KNX_transfer_timeout}}$ (e.g. network connection via satellite).

5.1.1.2 Overview

"Overview" is this clause.

5.1.1.3 Core specification

“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 specification 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 Device management

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

5.1.1.5 Tunnelling

“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 $t_{\text{KNX-transfer_timeouts}}$.

5.1.1.6 Routing

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

5.1.1.7 Remote diagnosis and configuration

“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.

5.1.1.8 Secured communication

“Secured communication” defines services for a secured point-to-point exchange of KNX telegrams over an IP network between a KNXnet/IP client and a KNX/IP server. The services provide means for establishing secured communication sessions by authorized KNXnet/IP clients.

5.1.1.9 List of codes

[Annex A](#) gives a complete listing of all codes used by KNXnet/IP, to which KNXnet/IP implementations shall conform, depending on the parts of this document supported.

5.1.1.10 Binary examples

[Annex B](#) gives binary examples of KNXnet/IP frames.

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 “n/a”. [Table 1](#) shows KNXnet/IP service types and IP protocols.

Table 1 — KNXnet/IP service types and IP protocols

IP protocol	Service type					
	Core	Device management	Tunnelling	Routing	Remote diagnosis and configuration	Secured communication
ARP	M	M	M	M	M	M
RARP	O	O	O	O	O	O
Support of fixed IP address	M	M	M	M	M	M
BootP (client)	M	M	M	M	M	M
DHCP (client)	M	M	M	M	M	M
UDP	M	M	M	M	M	M
TCP	O	O	O	n/a	n/a	M
ICMP	M	M	M	M	M	M
IGMP	M	M	n/a	M	O	M
It is essential that either BootP or DHCP is implemented by a KNXnet/IP device.						
TCP is mandatory for tunnelling v2 required for secured communication.						

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 document 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.

A KNXnet/IP client shall not assume that a KNXnet/IP server supports KNXnet/IP frames with a total length of more than 508.

5.1.2.3 Network environment

Because KNXnet/IP servers use IP, this document 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 connection, for example PPP, this would be at least 19 200 bit/s; for a network with up to 400 KNXnet/IP servers, this would be at 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 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. KNXnet/IP device classes are shown in [Table 2](#).

Table 2 — KNXnet/IP device classes

Device class	Service type					
	Core	Device management	Tunnelling	Routing	Remote diagnosis and configuration	Secured communication
A (Configuration and system maintenance tools)	M	M	M	M	M	M
B (KNXnet/IP router)	M	M	M	M	O ^a	O ^a
C (any other KNXnet/IP device)	M	M	O	O	O	O
^a If secure communication is supported, support of remote diagnosis and configuration is not allowed.						

5.2 Core

5.2.1 Use

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

This specification 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 document 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 3](#) shows KNXnet/IP frame binary format.

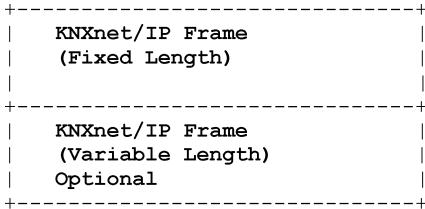


Figure 3 — KNXnet/IP frame binary format

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 4](#) shows the KNXnet/IP header binary format.

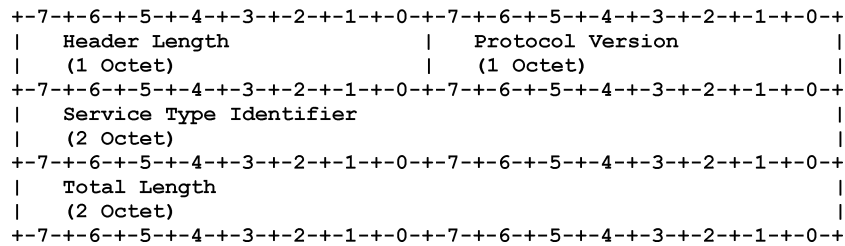


Figure 4 — KNXnet/IP header binary format

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 specification 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, bidirectional and connectionless endpoint for discovery if the host protocol requires discovery services. It shall support at least one bidirectional and connectionless endpoint for controlling and at least one bidirectional and connection-oriented endpoint for service type related data transmission. [Figure 5](#) shows the KNXnet/IP server endpoints sample configuration.

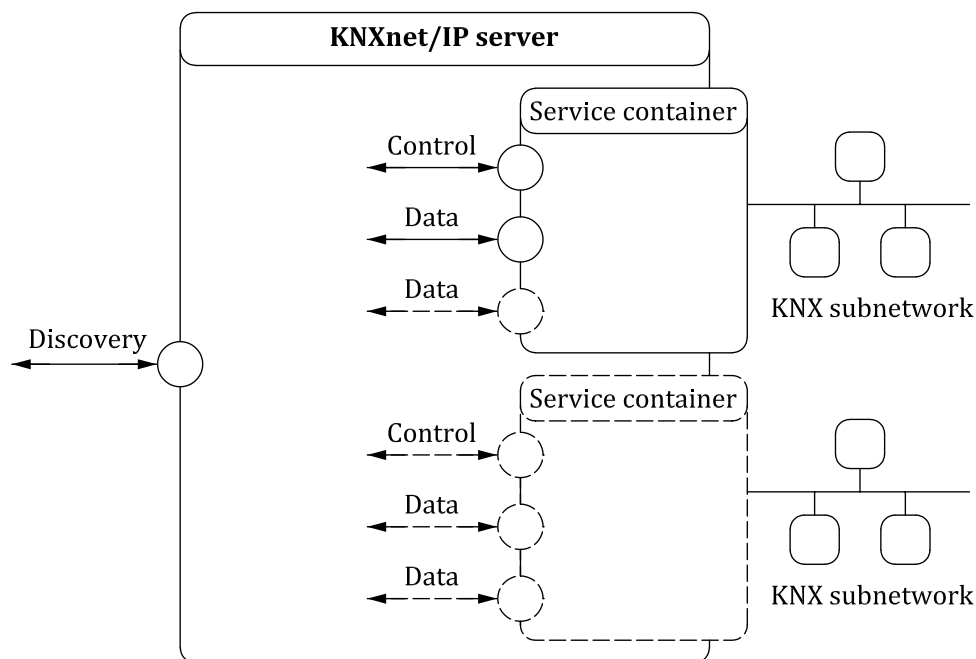


Figure 5 — KNXnet/IP server endpoints sample configuration

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 bidirectional 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 General

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 6](#) shows the discovery procedure.

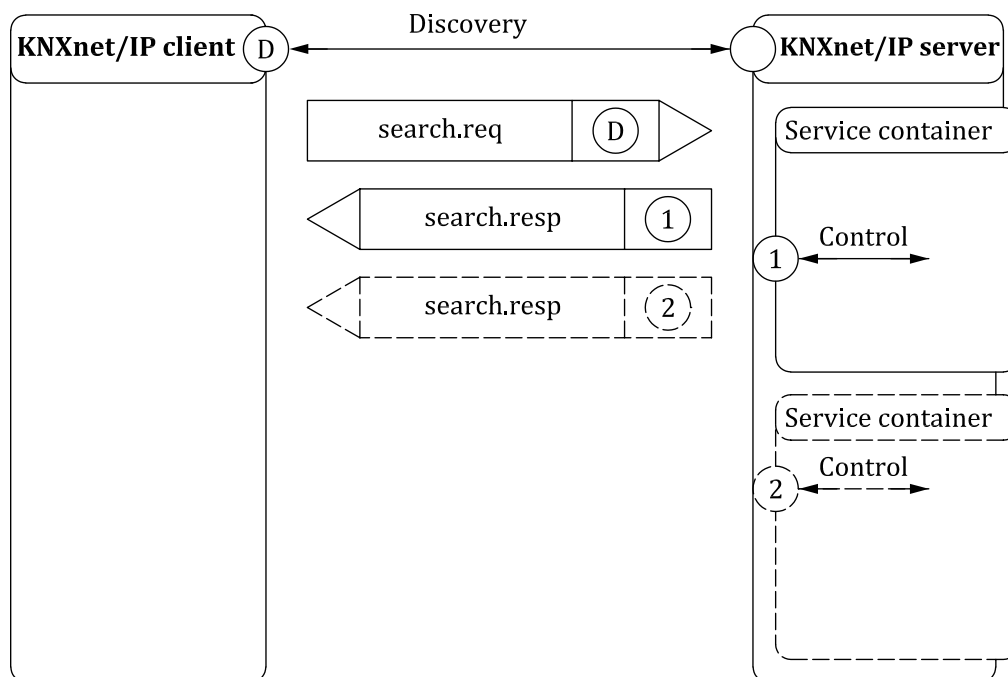


Figure 6 — Discovery procedure

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 General

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, for example 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) and the data endpoint HPAI that the client wants to use for this communication channel.

NOTE 1 The list of supported layers and services is supposed to be extended in further versions.

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

NOTE 2 It should be noted that a KNXnet/IP connection can consist, due to technical reasons, of multiple logical connections at the host protocol layer.

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 specification 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, for example 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. Establishing a data connection is shown in [Figure 7](#).

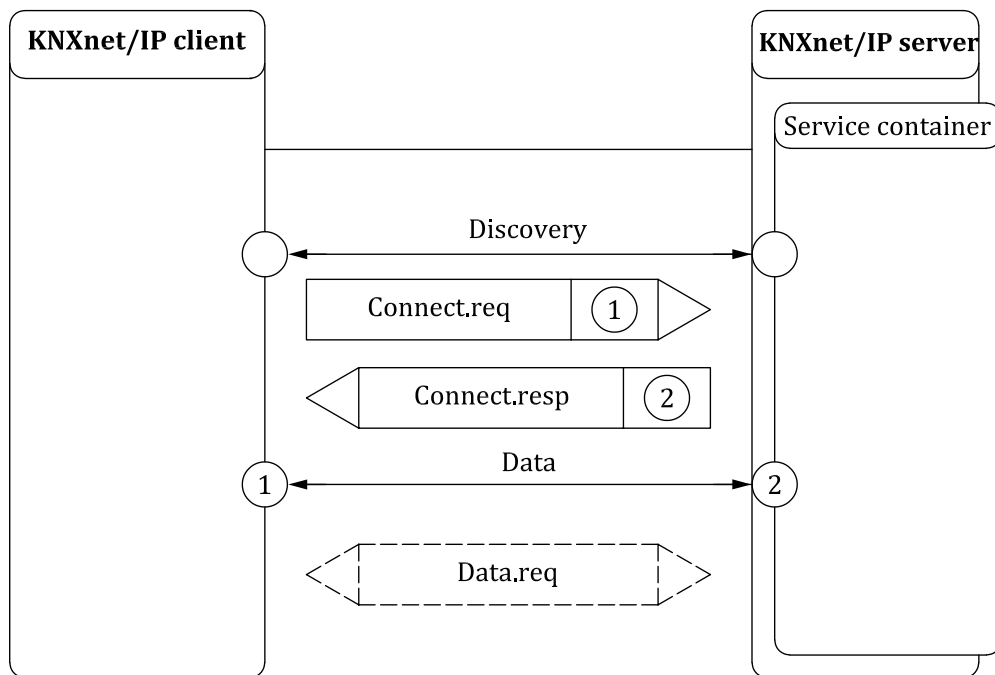


Figure 7 — Establishing a data connection

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. [Figure 8](#) shows a common connection header.

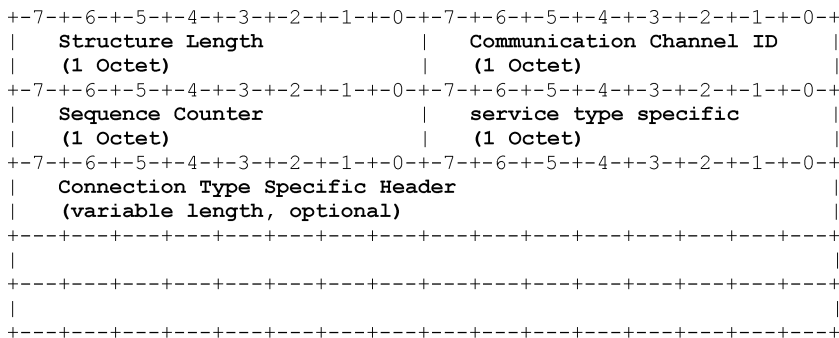


Figure 8 — Common connection header

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 zero, so the first KNXnet/IP frame sent on an established communication channel has a sequence counter of zero.

For connections inside a TCP connection, the sequence counter shall not be evaluated by the receiver. The sender may set the sequence counter to zero.

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 General

This clause defines programming guidelines that shall be taken into account when implementing KNXnet/IP servers or clients, respectively.

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 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 the KNX subnetwork.

5.2.6.4.2 KNXnet/IP router IP address assignment

KNXnet/IP routers shall support plug and play KNXnet/IP 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 9](#) shows a KNX project with multiple installations.

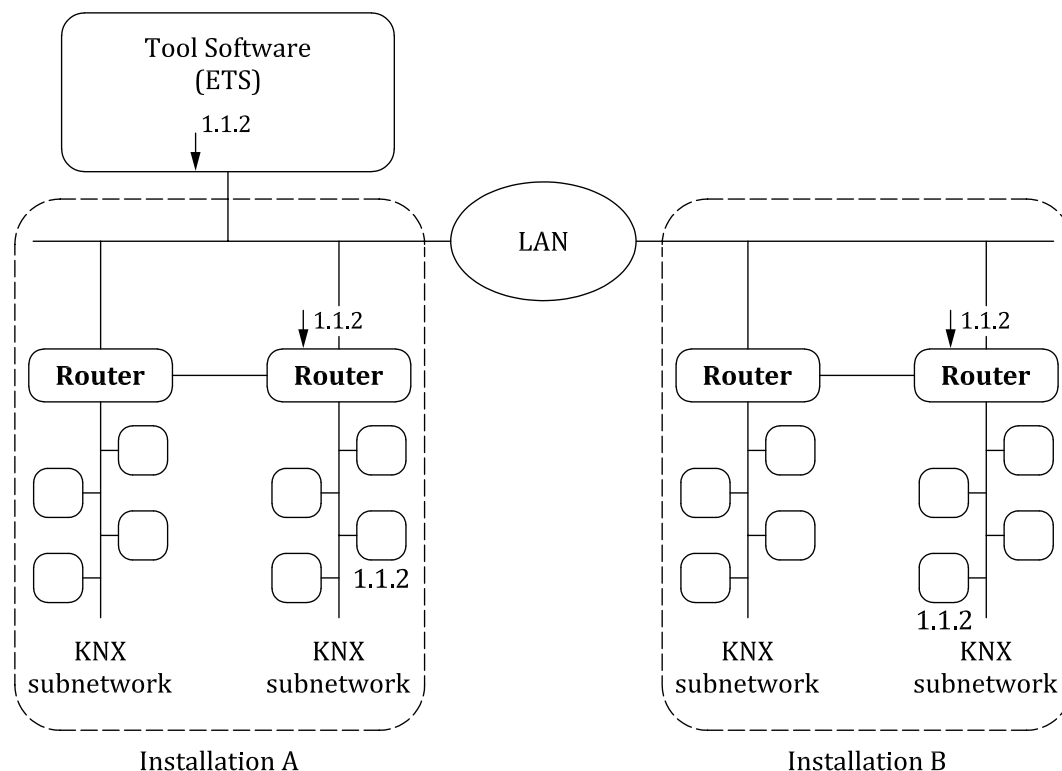


Figure 9 — KNX project with multiple installations

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

- 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.
- 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.
- 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.

- d) If directly connected to the IP network, the tool software can access all KNXnet/IP devices and through configured routers the corresponding KNX subnetworks.
- e) The tool software should use the IP network for the setup of projects containing KNXnet/IP devices.
- f) It is possible to completely configure KNXnet/IP devices in one single configuration procedure (KNX individual address assignment and parameter download).
- g) 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.
- h) 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.
- i) If the SEARCH_RESPONSE answers reveal that KNXnet/IP devices use different IP address spaces, the tool software shall present an error message.
- j) 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:

- a) 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.
- b) 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.
- c) 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.
- d) 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.

- e) The tool software shall establish a KNXnet/IP 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.
- f) 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:

- a) 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.
- b) The tool software shall send a KNXnet/IP SEARCH_REQUEST frame.
- c) 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.
- d) 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.
- e) The tool software shall set the KNX individual address, project identifier and subnetwork information if applicable.
- f) At this point, the tool software can download additional parameters if necessary.
- g) 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 General

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 10](#) shows a KNXnet/IP message header.

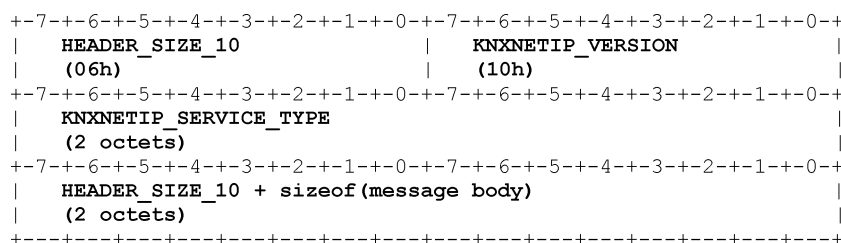


Figure 10 — KNXnet/IP message header

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 Core KNXnet/IP services

5.2.7.4.1 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.

5.2.7.4.2 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.

5.2.7.4.3 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.

5.2.7.4.4 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.

5.2.7.4.5 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.

5.2.7.4.6 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.

5.2.7.4.7 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.

5.2.7.4.8 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.

5.2.7.4.9 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.

5.2.7.4.10 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.

5.2.7.4.11 SEARCH_REQUEST_EXTENDED

This constant with value 020Bh shall identify the KNXnet/IP service type sent by KNXnet/IP client, to search available KNXnet/IP servers and receive extended information about the available KNXnet/IP servers.

5.2.7.4.12 SEARCH_RESPONSE_EXTENDED

This constant with value 020Ch shall identify the KNXnet/IP service type sent by KNXnet/IP server in response to a SEARCH_REQUEST_EXTENDED.

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/IP specification. [Figure 11](#) shows the HPAI structure binary format.

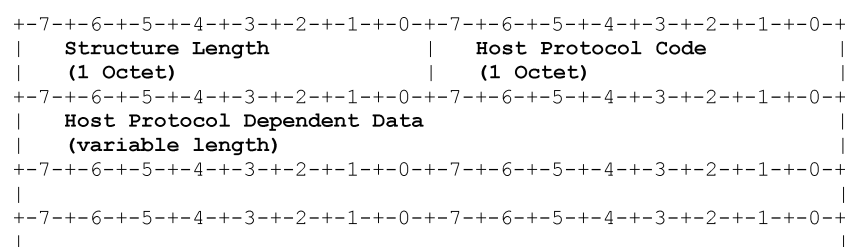


Figure 11 — HPAI structure binary format

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/IP specification. [Figure 12](#) shows the CRI structure binary format.

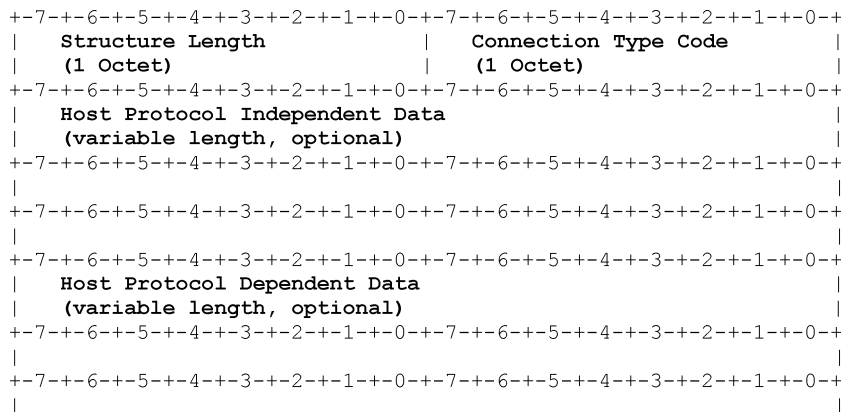


Figure 12 — CRI structure binary format

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 specification. [Figure 13](#) shows the CRD structure binary format.

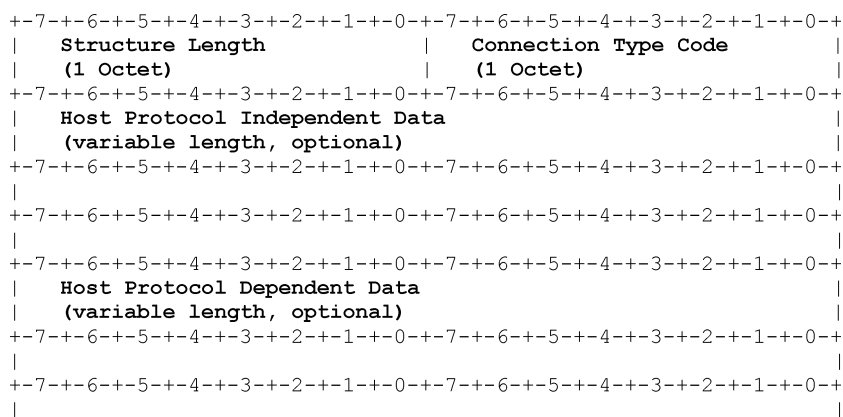


Figure 13 — CRD structure binary format

5.2.7.5.4 Description Information Block (DIB)

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 that may have to be achieved by padding with 00h in the last octet of the DIB structure. [Figure 14](#) gives a description of the structure binary format.

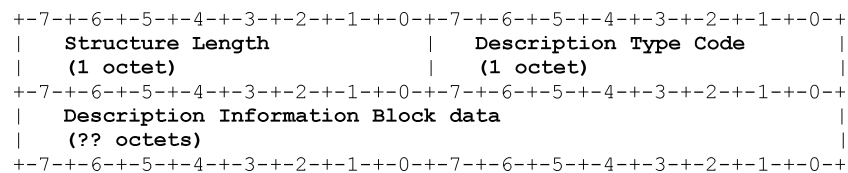


Figure 14 — Description structure binary format

[Table 3](#) lists the valid description type codes.

Table 3 — Description type codes

Description type	Value	Description
DEVICE_INFO	01h	Device information (e.g. KNX medium)
SUPP_SVC_FAMILIES	02h	Service families supported by the device
IP_CONFIG	03h	IP configuration of the device
IP_CUR_CONFIG	04h	Current IP configuration of the device
KNX_ADDRESSES	05h	KNX addresses used by/assigned to the device
SECURED_SERVICES	06h	Secured services
TUNNELLING_INFO	07h	Tunnelling information
EXTENDED_DEVICE_INFO	08h	Extended device information
Reserved	09h–FDh	Reserved for future use
MFR_DATA	FEh	DIB structure for further data defined by device manufacturer
not used	FFh	Not used

The device information DIB shall have the structure as given below in [Figure 15](#).

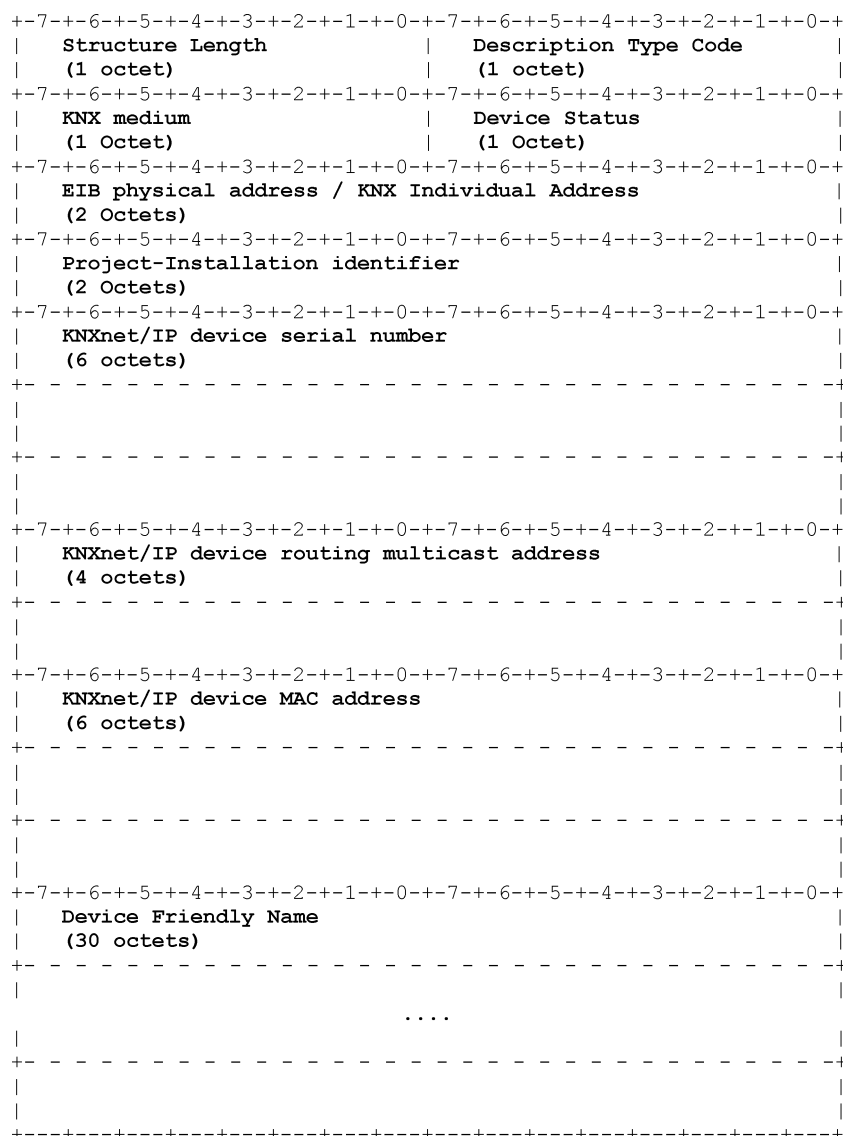


Figure 15 — Structure of device information DIB

a) KNX medium:

The KNX medium codes shall be as specified in [Table 4](#) below.

Table 4 — 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.

b) Device status:

The device status octet shall be as specified in [Table 5](#) below:

Table 5 — Device status octet

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

c) Project-installation identifier:

The project-installation identifier shall be defined as shown in [Table 6](#):

Table 6 — Project-installation identifier

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.

d) 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.

e) 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. 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.

f) 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.

g) 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.

The extended device information DIB shall have the structure as given in [Figure 16](#) below.

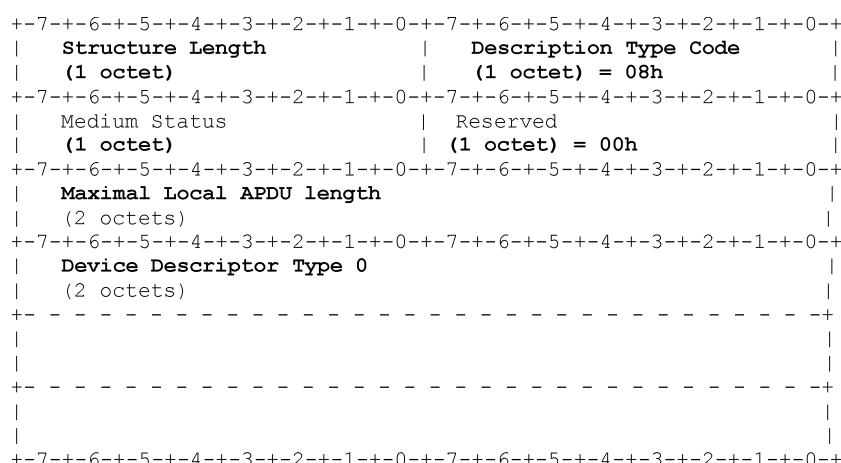


Figure 16 — Extended device information DIB

h) Medium status:

The contents of this field shall be identical as the contents of the Property `PID_MEDIUM_STATUS` (PID: 51) in the router object.

- For an IP/TP1 KNXnet/IP router (mask 091Ah), this shall be the connection state of the TP1 connection at the secondary side.
- For a KNXnet/IP tunnelling server on TP1, this shall be the connection state of the TP1 connection.
- For all other KNXnet/IP devices, the field COMMUNICATION_IMPOSSIBLE shall have the value FALSE.

i) Maximal local APDU length:

This shall be the maximal APDU length that is supported by the KNXnet/IP device.

The contents of this field shall be identical as the contents of the Property PID_MAX_LOCAL_APDU_LENGTH (PID: 69) in the cEMI server object.

NOTE This is not be confused with the PID_MAX_APDU_LENGTH (PID: 58) in the router object and which is available in the KNXnet/IP router.

j) Device descriptor type 0:

This shall be the device descriptor type 0 (mask version) of the KNXnet/IP device.

The contents of this field shall be identical as the contents of the Property `PID_DEVICE_DESCRIPTOR` (PID: 83) in the device object.

The supported service families DIB shall have this structure, see [Figure 17](#):

[illegible]

Figure 17 — Supported services families DIB

The service family IDs shall be the high octet of the service type ID. A list of service type IDs 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. 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.

The service family version shall refer to the implemented version of the KNXnet/IP service family. The service family versions are defined in the specification papers of the services explicitly or through the definition of profiles.

EXAMPLE If Core v2 is supported then the supported service families DIB include 02h for the core service family (02h).

When used within a SEARCH_RESPONSE or DESCRIPTION_RESPONSE frame, only the supported service families indicated in Table 7 shall be included. When used within a SEARCH_RESPONSE_EXTENDED frame, all supported service families shall be included.

Table 7 — Allowed supported service families

Supported family	Code	Allowed in SEARCH_RESPONSE/ DESCRIPTION_RESPONSE	Allowed in SEARCH_RESPONSE_EXTENDED
Core	02h	Yes	Yes
Device management	03h	Yes	Yes
Tunnelling	04h	Yes	Yes
Routing	05h	Yes	Yes
Remote logging	06h	Yes	Yes
Remote configuration and diagnosis	07h	Yes	Yes
Object server	08h	Yes	Yes
Security	09h	No	Yes
Other		No	Yes

NOTE “Other” denotes placeholder for possible future extensions.

The manufacturer data DIB has this structure, see Figure 18:

```

+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|  Structure Length      |  Description Type Code      |
|  (1 octet)            |  (1 octet)            |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|  KNX Manufacturer ID  |                          |
|  (2 Octets)           |                          |
+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+-7-+-6-+-5-+-4-+-3-+-2-+-1-+-0-+
|  Any manufacturer specific data |
|  (?? Octets)          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Figure 18 — 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. Communication to and from the discovery endpoint of KNXnet/IP devices is only allowed using UDP datagrams. 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. If this information contains a TCP address, the SEARCH_REQUEST frame shall be discarded. Figure 19 shows the SEARCH_REQUEST frame binary format.

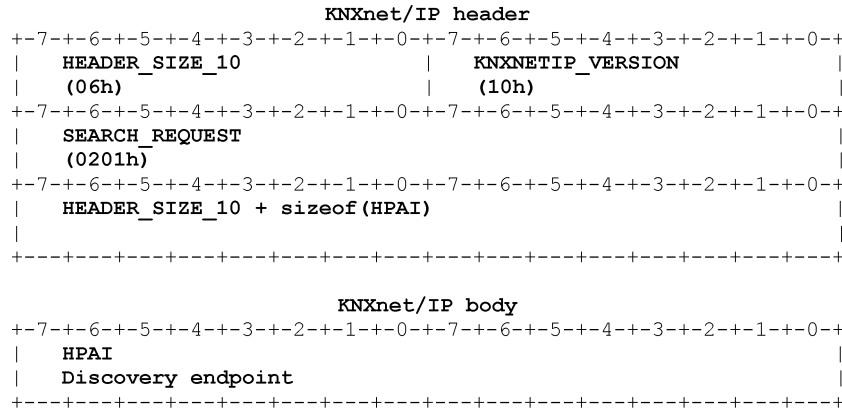


Figure 19 — SEARCH_REQUEST frame binary format

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 DIB, which shall include the actual service family version and is not restricted to 01h.

EXAMPLE If the tunnelling v2 is supported, then the service family version for tunnelling is 02h.

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.

KNXnet/IP servers supporting TCP shall only report the UDP address of their control endpoint in the SEARCH_RESPONSE frame, see [Figure 20](#).



Figure 20 — SEARCH_RESPONSE frame binary format

5.2.7.6.3 SEARCH REQUEST EXTENDED

a) Definition:

The SEARCH_REQUEST_EXTENDED frame shall be sent by a KNXnet/IP client to either the discovery endpoints of any listening KNXnet/IP servers or to the control endpoint of a specific KNXnet/IP server.

Communication to and from the discovery endpoint of KNXnet/IP devices is only allowed using UDP datagrams. As communication with the discovery endpoint shall be connectionless and stateless, the KNXnet/IP client's discovery endpoint address information shall be included in the KNXnet/IP body. If this information contains a TCP address, the SEARCH_REQUEST_EXTENDED frame shall be discarded.

Communication to and from the control endpoint of KNXnet/IP devices is allowed using UDP or TCP.

b) Binary format:

Figure 21 shows the binary format of SEARCH REQUEST EXTENDED frame.

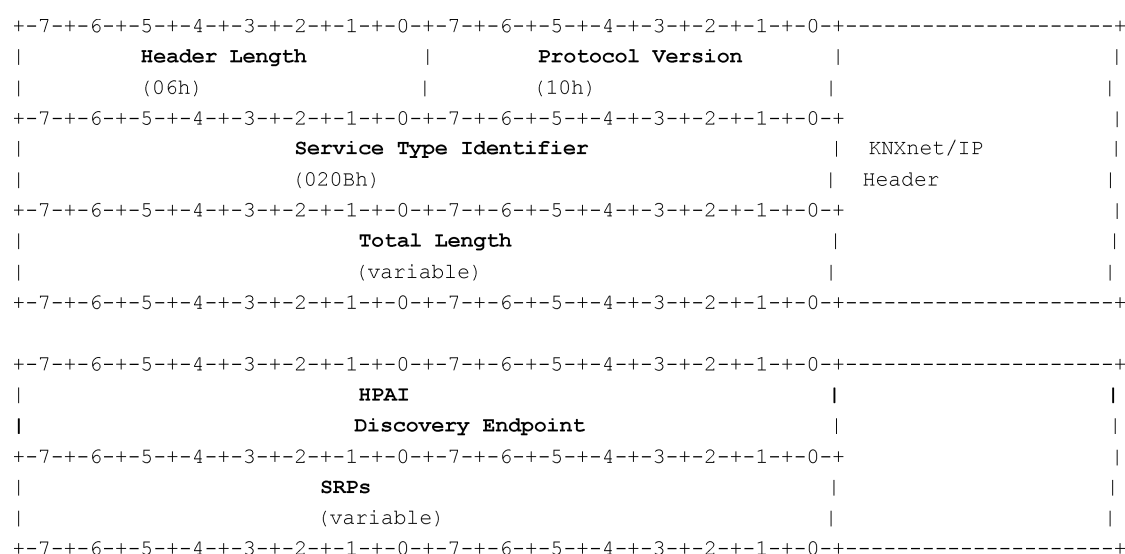


Figure 21 — Binary format of SEARCH REQUEST EXTENDED frame

c) Search Request Parameter (SRP):

The KNXnet/IP client may include zero or more Search Request Parameter Blocks (SRP) to transfer additional information regarding the search.

NOTE 1 This can be used, for example to restrict the set of devices that are expected to respond or to influence the type of DIBs that the client is interested in.

The general format of a SRP block is given in [Figure 22](#):

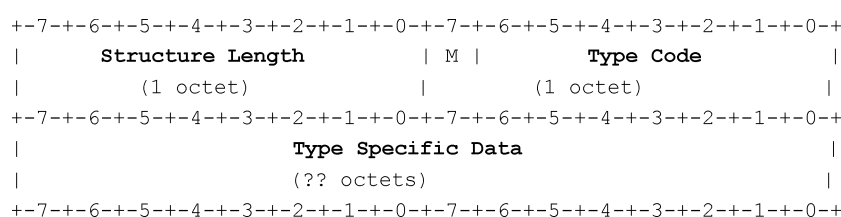


Figure 22 — Binary format of Search Request Parameter Block

The KNXnet/IP server shall interpret the SRPs in the following way:

- If the KNXnet/IP server supports the SRP, it shall apply the selection criteria as specified with the specific SRP type.
- Otherwise:
 - If the M (Mandatory) bit is set, the server shall not respond to the search request.
 - If the M (Mandatory) bit is not set, the server shall ignore this SRP and react as if this SRP were not present.

If multiple SRPs are present, the expected result of their combination is specified with the individual SRP types. If the evaluation of any of the SRPs leads to the decision to not respond to the search request then the following SRPs shall not influence this decision and can thus be skipped.

d) SRP type overview:

The SRP type overview is shown in [Table 8](#).

Table 8 — SRP type overview

SRP type	Code	M/O
Invalid	00h	See below
Select by Programming Mode	01h	M
Select by MAC Address	02h	M
Select by Service SRP	03h	M
Request DIBs	04h	M
Reserved	05h to 7Fh	n/a

The KNXnet/IP client shall never use the invalid code 00h. The KNXnet/IP server shall handle an SRP with the invalid code 00h as any other unknown SRP code, i.e. if the M bit is set, the server shall not respond, if the M bit is not set, the SRP shall be ignored.

NOTE 2 The invalid code 00h serves to test the behaviour of the server for unknown SRPs.

NOTE 3 The codes 01h and 02h match the SELECTOR encoding in KNXnet/IP demote configuration and diagnosis. This does however not mean that these two encodings have to be identical for additional selectors/SRP types.

e) SRP type “Select by Programming Mode SRP”:

The client shall include this SRP to indicate that it is interested only in the response from KNXnet/IP servers in which the programming mode is currently enabled.

If programming mode is not enabled in the KNXnet/IP server then it shall not respond to this search request.

The binary format of select by programming mode SRP is shown in [Figure 23](#).

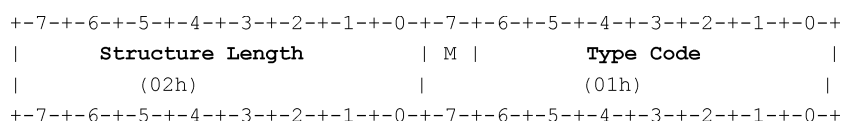


Figure 23 — Binary format of select by programming mode SRP

f) SRP type “Select by MAC Address”:

The client shall include this SRP to indicate that it is interested only in the response from the KNXnet/IP server with the given MAC address.

If the KNXnet/IP server's MAC address is different from the given MAC address then it shall not respond to this search request.

The binary format of Select by MAC address SRP is shown in [Figure 24](#).

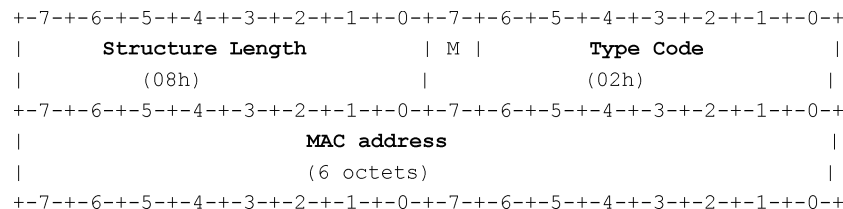


Figure 24 — Binary format of Select by MAC address SRP

g) SRP type "Select by Service":

The client shall include this SRP to indicate that it is interested only in responses from KNXnet/IP servers supporting the given KNXnet/IP service family in at least the given version.

If the KNXnet/IP server does not support the given service family or supports the given service family only in a lower version then it shall not respond to this search request.

The binary format of Select by Service SRP is shown in [Figure 25](#).

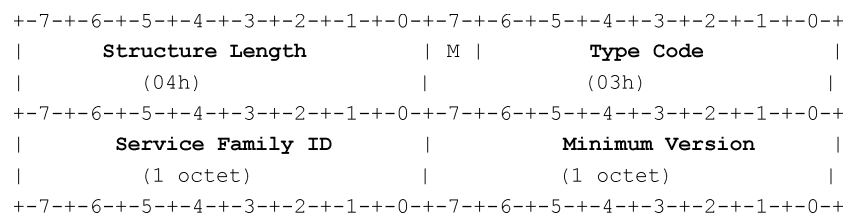


Figure 25 — Binary format of Select by Service SRP

h) SRP type "Request DIBs":

The client shall include this SRP to indicate that it is interested in the listed DIBs. This SRP shall not influence the decision of the KNXnet/IP server whether or not to respond to the search request.

If no "Request DIBs" SRP is present in the request, the KNXnet/IP server shall at least report the basic set of DIBs consisting of the "Device Information DIB", "Extended Device Information DIB" and "Supported Services DIB". The server may include in addition any number of other DIBs in the response.

If a Request DIBs SRP is present in the request then the KNXnet/IP server shall at least report the set of DIBs that are listed in the SRP and that are supported by the server. The server may include in addition any number of other DIBs in the response. The server shall ignore description types that are not recognized or not supported.

This SRP has variable length. If the client is interested in an odd number of DIBs, it shall add an additional description type 0 to make the structure length even.

The binary format of the Request DIBs SRP is shown in [Figure 26](#).

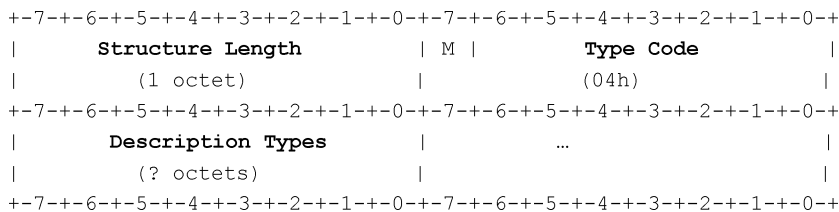


Figure 26 — Binary format of the Request DIBs SRP

5.2.7.6.4 SEARCH RESPONSE EXTENDED

a) Definition:

The KNXnet/IP server shall send the SEARCH_RESPONSE_EXTENDED frame as an answer to a received SEARCH_REQUEST_EXTENDED frame. It shall be addressed to the KNXnet/IP client's discovery endpoint using the HPAI included in the received SEARCH_REQUEST_EXTENDED frame.

The HPAI of the KNXnet/IP server's own control endpoint shall be carried in the KNXnet/IP body of the SEARCH_RESPONSE_EXTENDED 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 KNXnet/IP server shall announce each of its own control endpoints in a single SEARCH_RESPONSE_EXTENDED frame.

KNXnet/IP servers supporting TCP shall only report the UDP address of their control endpoint in the SEARCH_RESPONSE_EXTENDED frame.

The KNXnet/IP server may report the DIBs in the response in any order. Each DIB shall be present only once in the response.

b) Binary format:

The binary format of the SEARCH_RESPONSE_EXTENDED frame is shown in [Figure 27](#).

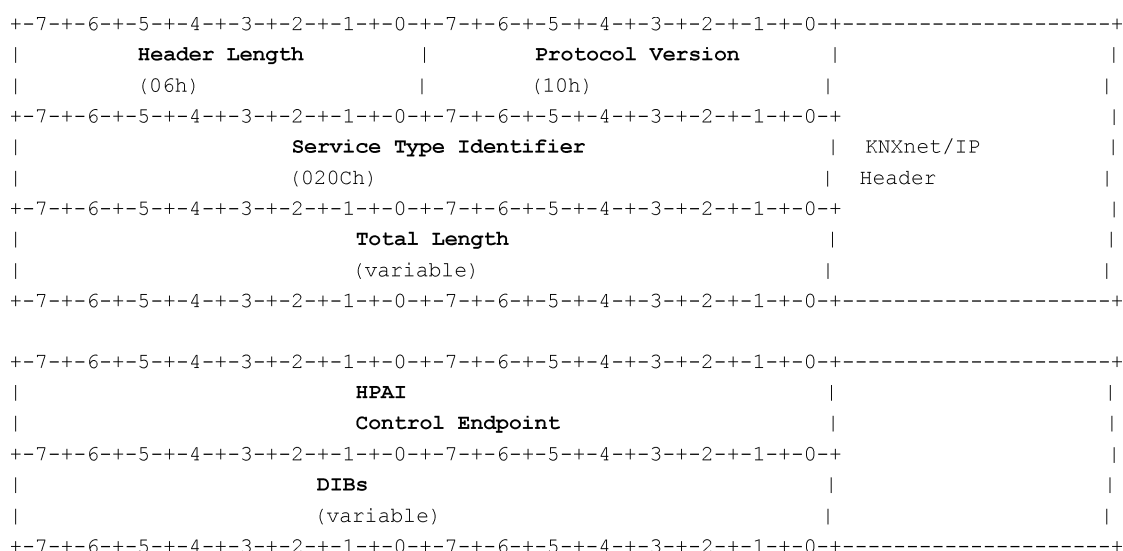


Figure 27 — Binary format of the SEARCH_RESPONSE EXTENDED frame

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.

The DESCRIPTION_REQUEST frame binary format is shown in [Figure 28](#).

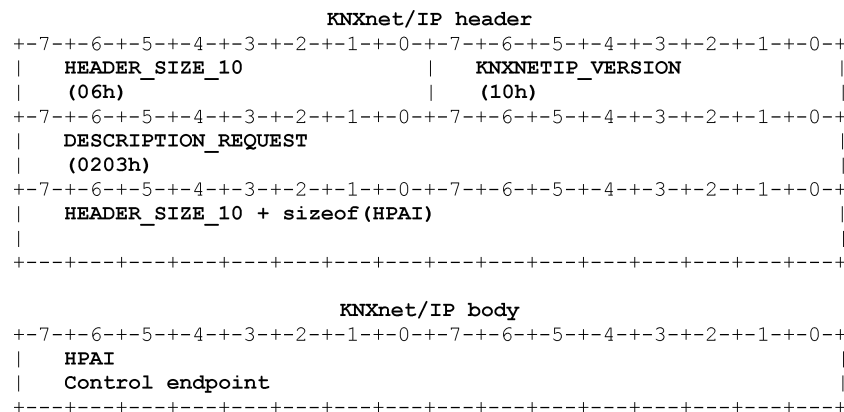


Figure 28 — DESCRIPTION_REQUEST frame binary format

5.2.7.7.2 DESCRIPTION_RESPONSE

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. The supported service families DIB shall include the actual service family version and is not restricted to 01h.

EXAMPLE If the tunnelling v2 is supported, then the service family version for tunnelling is 02h.

The DESCRIPTION_RESPONSE frame binary format is shown in [Figure 29](#).

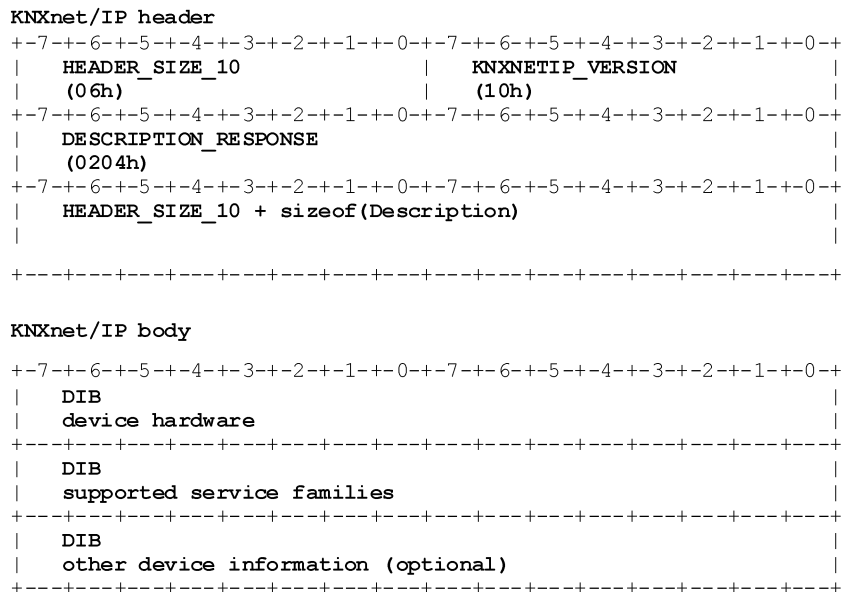


Figure 29 — DESCRIPTION_RESPONSE frame binary format

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, see [Table 9](#).

Table 9 — 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 KNX telegrams between two KNXnet/IP devices.
REMLOG_CONNECTION	06h	1	Data connection used for configuration and data transfer with a remote logging server.
REMCNFG_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 HPAI of the client's data endpoint meant for the requested data connection shall complete the body of the CONNECT_REQUEST frame, see [Figure 30](#).

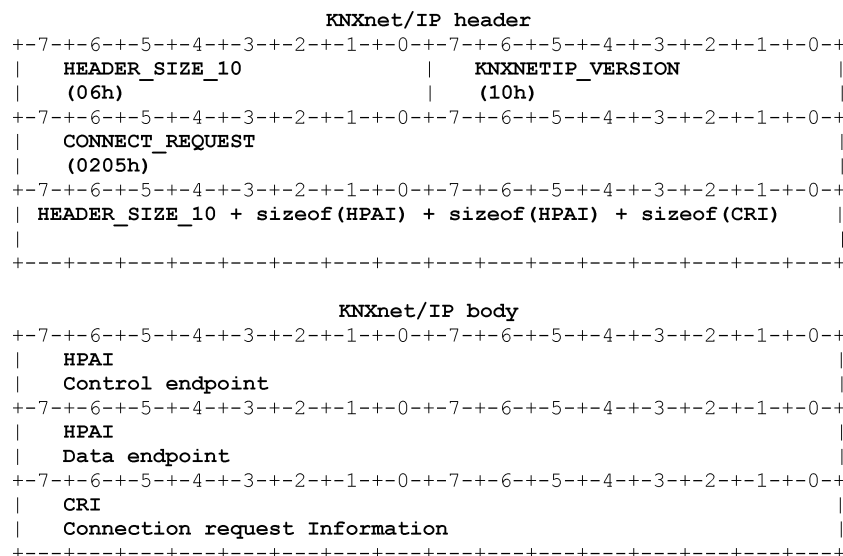


Figure 30 — CONNECT_REQUEST frame binary format

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 10 — 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 HPAI 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, see [Figure 31](#).

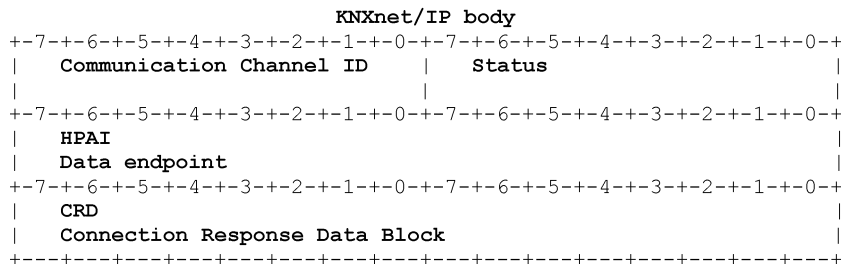


Figure 31 — CONNECT_RESPONSE frame binary format

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.

For the CONNECTIONSTATE_REQUEST frame binary format, see [Figure 32](#).

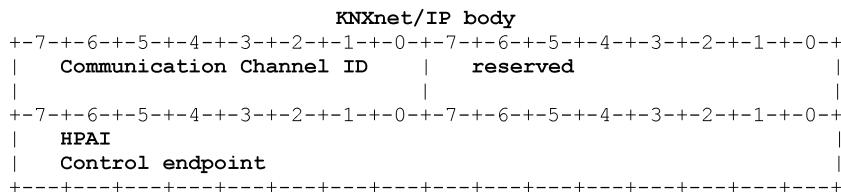


Figure 32 — CONNECTIONSTATE_REQUEST frame binary format

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 11](#) lists the status codes that are defined for the CONNECTIONSTATE_RESPONSE frame in [Figure 33](#):

Table 11 — 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 KNX subsystem connection with the specified ID.

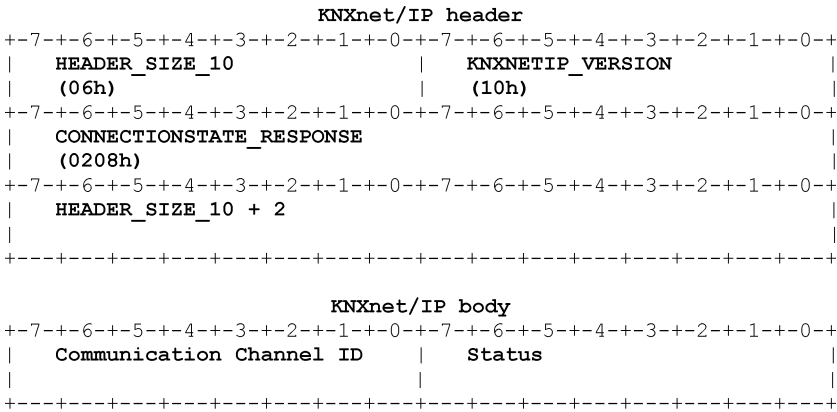


Figure 33 — CONNECTIONSTATE_RESPONSE frame binary format

5.2.7.8.5 DISCONNECT_RESPONSE

The DISCONNECT_REQUEST frame binary format is shown in [Figure 34](#). The DISCONNECT_RESPONSE frame binary format is shown in [Figure 35](#).

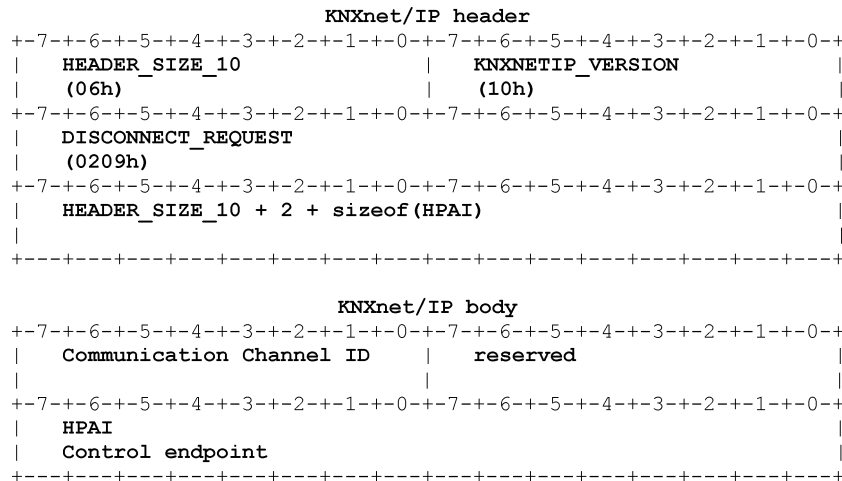


Figure 34 — DISCONNECT_REQUEST frame binary format

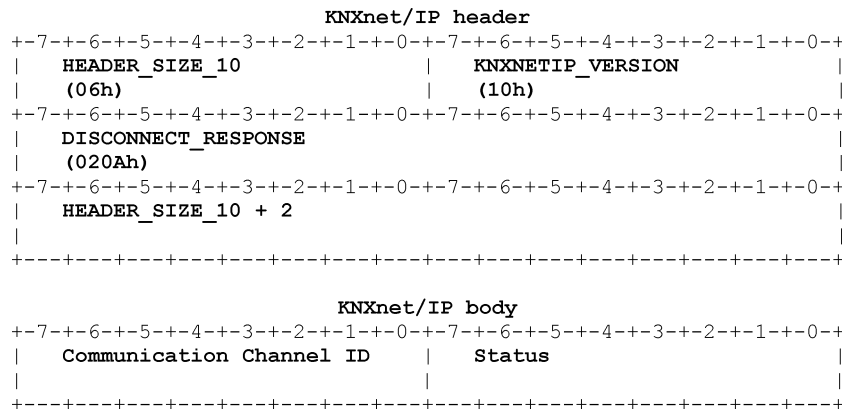


Figure 35 — DISCONNECT_RESPONSE frame binary format

5.2.8 IP Networks

5.2.8.1 General

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 document 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. The IP protocol stack is shown in [Figure 36](#).

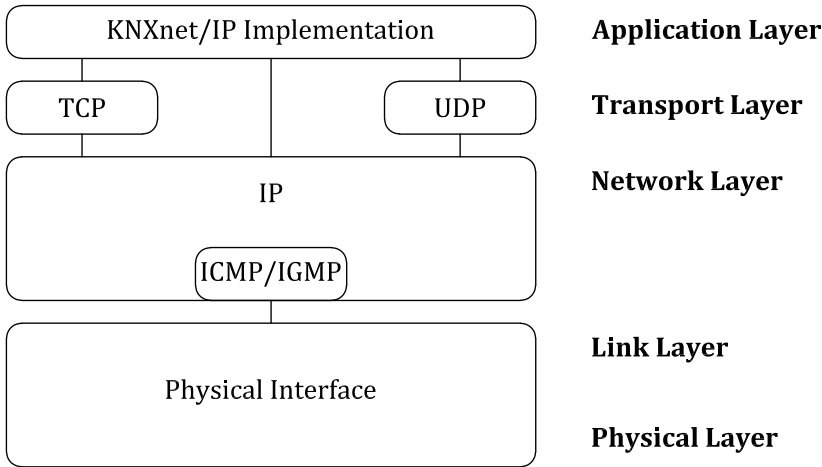


Figure 36 — IP protocol stack

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

5.2.8.4.1 General

Since the KNXnet/IP protocol itself employs end-to-end acknowledgment, it is not necessary to guarantee tunnelling IP packets containing KNX data telegrams 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 multicast 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 multicast 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.

KNXnet/IP servers supporting both UDP and TCP shall accept unicast communication on both IP protocols. Responses and related communication like KNXnet/IP connections shall always use the IP protocol (UDP or TCP) of the original request.

5.2.8.4.2 UDP

To address the reliability and sequencing issue of UDP there shall be acknowledgement frames sent and sequence counters added to the connection header to help in sequencing them.

Using UDP, datagrams can be sent using either unicast or multicast 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 multicast addressing. It is therefore recommended that KNXnet/IP devices support both unicast and multicast IP addressing although it is not required that a KNXnet/IP device supports multicasts in order to inter-operate with a KNXnet/IP device that does.

5.2.8.4.3 TCP

a) Opening of a TCP connection:

TCP connections shall always be initiated by the KNXnet/IP client.

A KNXnet/IP server shall support at least one single TCP connection at a time and may support any number of further additional concurrent TCP connections at the same time. If no more TCP connections are possible, the KNXnet/IP server shall indicate this by not creating the TCP connection. No KNXnet/IP error code is needed for that.

A KNXnet/IP client shall not assume that a KNXnet/IP server supports more than one concurrent TCP connection.

b) Closing of a TCP connection:

1) Sequence of actions to close a TCP connection:

- The **client** shall close a TCP connection in the following order: first any KNXnet/IP connections, then any KNXnet/IP session, at last the TCP connection.

- For the **server**, this order is a recommendation but not a requirement. The client shall not rely on any order of events. This can and shall not be tested.

NOTE Additionally, if the server takes down the TCP connection then the indications of this to the client can however get lost in the process.

2) Client side closing of a TCP connection:

Under normal conditions, TCP connections are closed by the client.

3) Server side closing of a TCP connection

The KNXnet/IP server shall take down the TCP connection under the following exceptional conditions:

- If no KNXnet/IP connection or session is active and no octets have been received for 10 s over this TCP connection then the KNXnet/IP server shall close the TCP connection.
- If the last KNXnet/IP connection or session within a TCP connection is closed by the KNXnet/IP server due to a timeout then the KNXnet/IP server shall close this TCP connection immediately.
- If the last KNXnet/IP connection or session within a TCP connection is closed by the KNXnet/IP client due to timeout, then the KNXnet/IP client shall close this TCP connection immediately.
- If the KNXnet/IP server handles an A_Restart or an M_Reset, then it shall close all KNX TCP connections to the related Control Endpoint — including the TCP connection through which possibly the A_Restart or M_Reset is received — and all its other KNX TCP connections in the same security domain (that is, everything that is affected by the security settings in this KNXnet/IP server). Any client shall assume these connections to be closed, even if the proper indications on this from the server are not received.

c) Sending and receiving frames, frame synchronization and receive timeouts in a TCP receiver

NOTE Because in contrast to the packet-oriented UDP, TCP is a stream-oriented protocol, there is no 1:1 relation between IP packets and KNXnet/IP frames: One IP packet can contain multiple KNXnet/IP frames and one KNXnet/IP frame can span over multiple IP packets.

Senders shall send KNXnet/IP frames without any extra octets between frames.

Receivers shall use the information in the KNXnet/IP header to recover the frame structure from the octet stream.

To receive a KNXnet/IP frame, the following procedure shall be used.

- The receiver shall wait until enough octets are received to interpret the KNXnet/IP frame header, or a timeout occurs as specified below.
- If the receiver detects that the sender has closed his sending side (half-closed TCP connection) the receiver shall discard any pending requests and close the TCP connection immediately.
- If the header is not a well-formed KNXnet/IP header, the receiver shall close the TCP connection.
- If the total length given in the KNXnet/IP header is longer than the available receive buffer space in the receiving device, the receiver shall skip the body of the frame. The frame is effectively ignored. The receiver shall not close the TCP connection. All devices shall support (i.e. shall be able to skip, not necessarily be able to store) frames with a total length of up to 65 535 octets.
- If the service type given in the KNXnet/IP header is supported by the receiver, the receiver shall receive the body of the frame and handle it.

- If the service type given in the KNXnet/IP header is not supported by the receiver, the receiver shall skip the body of the frame and ignore it. The receiver shall not close the TCP connection because of an unsupported or unknown service type.

Receiving or skipping the body of a frame (after having received a KNXnet/IP header) shall be done using this procedure:

- The receiver shall receive or skip the number of octets corresponding to the length of the body ("total length" minus length of header). See below for receive timeout.

Receive timeout: While receiving a KNXnet/IP header or body the reception of data may stall before having received enough data for the header or for the body. The receiver shall wait for the remaining data and shall close the TCP connection when running into the following timeouts:

- When no KNXnet/IP connection and no secure session is active: The receiver shall close the connection when not having received any octets for 10 s. This timeout is restarted with each received octet.
- When at least one KNXnet/IP connection or at least one secure session is active: The receiver shall not close the TCP connection.
- When the last active KNXnet/IP connection or secure session times out, the receiver shall close the TCP connection immediately. This means a timeout of an inner KNXnet/IP connection or secure session is also a timeout for the outer TCP connection, unless there are further KNXnet/IP connections or secure sessions.

The terms 'KNXnet/IP connection is active' and 'secure session is active' in the definition above are defined as follows:

- A KNXnet/IP connection is active after having received a CONNECT_REQUEST until either the connection request times out, is negatively answered or until the connection is disconnected or times out.
- A secure session is active after having received a SESSION_REQUEST, SESSION_RESPONSE or SESSION_AUTHENTICATE until either any of these operations times out, until an authenticated secure session is closed or until an authenticated secure session times out.

NOTE Receivers can get out-of-sync with the frame structure, for example because senders send erroneous data (e.g. a wrong total length) due to software errors or intentionally (attacks), or because of blindly injected TCP segments (attacks). Receivers cannot rely on receiving a frame or even just the KNXnet/IP header in one chunk. TCP does not guarantee preservation of sent chunks on the receiving side. KNXnet/IP frames might be generally assumed to be totally misaligned with any received data chunks. For example, a sender sending a frame as individual 1-octet chunks is a valid sender. A sender sending data chunks of 4 096 in size, generally misaligned with any frame boundary, is a valid sender. None of the above rules apply to UDP communication, which has implicit frame boundaries and which has no underlying connection to be terminated on a timeout.

5.2.8.4.4 Relation between TCP connections, secure sessions and plain KNXnet/IP connections

- One TCP connection may contain multiple concurrent secure session and plain KNXnet/IP connections. The lifetime of these secure session and plain KNXnet/IP connections may overlap in any way.
- One TCP connection may contain multiple sequential secure session and plain KNXnet/IP connections. After closing the last secure session and plain KNXnet/IP connection, a new secure session or plain KNXnet/IP connection can be established in the same TCP connection, if it is requested before the TCP connection runs into its 10 s idle timeout.
- Each secure session and each plain KNXnet/IP connection is associated with exactly one TCP connection and may only be used through this TCP connection. Secure sessions and plain KNXnet/IP connections do not span multiple TCP connections (neither concurrent TCP connections nor

sequential TCP connections). Secure session keys and connection IDs negotiated in one TCP connection are not valid in other TCP connections that may exist in parallel.

- Closing a TCP connection implicitly closes all contained secure sessions and plain KNXnet/IP connections. It also closes secure sessions that are not yet authenticated.
- TCP connections are not associated with a secure session user id, a certain authentication, a certain authorization or any other privilege.

5.2.8.4.5 Relation between secure sessions and KNXnet/IP connections

- a) One secure session may contain multiple concurrent KNXnet/IP connections. The lifetime of these KNXnet/IP connections may overlap in any way.
- b) One secure session may contain multiple sequential KNXnet/IP connections. After closing the last KNXnet/IP connection, a new KNXnet/IP connection can be established in the same secure session, if it is established before the secure session runs into its idle timeout.
- c) Each KNXnet/IP connection is associated with exactly one secure session and may only be used through this secure session. KNXnet/IP connections do not span multiple secure sessions (neither concurrent secure session nor sequential secure sessions).
- d) Closing a secure session implicitly closes all contained KNXnet/IP connections.

5.2.8.5 IP address assignment

5.2.8.5.1 IP unicast address

- a) 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.

- b) 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, 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.

- c) 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.

- d) IP address assignment procedure:

[Table 12](#) describes the address assignment procedure for KNXnet/IP devices.

Table 12 — 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 assigned.
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

a) 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.

b) 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 multicast group, and may be capable of belonging to up to 16 multicast groups. A KNXnet/IP device shall support multicasting.

A KNXnet/IP device shall support these IP protocols: ARP, ICMP, IGMP, BootP/DHCP and UDP.

NOTE As regards BootP/DHCP, it is essential that either one is implemented.

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

a) General:

The HPAI 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, see [Figure 37](#).

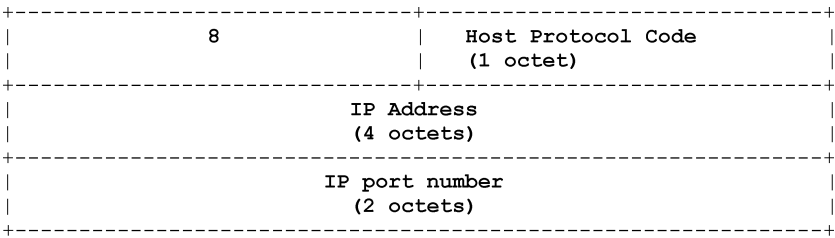


Figure 37 — IP HPAI binary format

The host protocol codes for IP network is given in [Table 13](#).

Table 13 — 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.

For the host protocol code IPV4_TCP only the special “Route Back” encoding (see below) is allowed. All other encodings are invalid and result in discarding the containing KNXnet/IP frame.

b) Route Back HPAIs:

For special communication scenarios, it is not suitable to put information about IP host protocol addresses into the payload of KNXnet/IP frames. If, for example, the frame shall travel across IP routers performing network address translation (NAT), the IP Host Protocol Address Information that the KNXnet/IP server needs to address to send KNXnet/IP frames back to the KNXnet/IP client may not be known to the KNXnet/IP client. Also, for TCP connections the KNXnet/IP client might want to tell the KNXnet/IP server that frames shall be sent on the TCP connection that was already established between them and that a new TCP connection should not be created by the KNXnet/IP server.

For these scenarios, a special encoding of the HPAI with the IP address and IP port number all set to zero shall be used. Such an HPAI is called a “Route Back” HPAI. It is not allowed to set only the IP address or only the IP port number to zero and leave the other address part to a value not equal to zero. Such HPAIs are invalid and the containing KNXnet/IP frames shall be ignored by the receiving KNXnet/IP device.

For UDP “Route Back” HPAIs this means that the KNXnet/IP server shall use the IP address and the port number in the IP package received as the target IP address or port number for the response to the KNXnet/IP client, see [Figure 38](#).

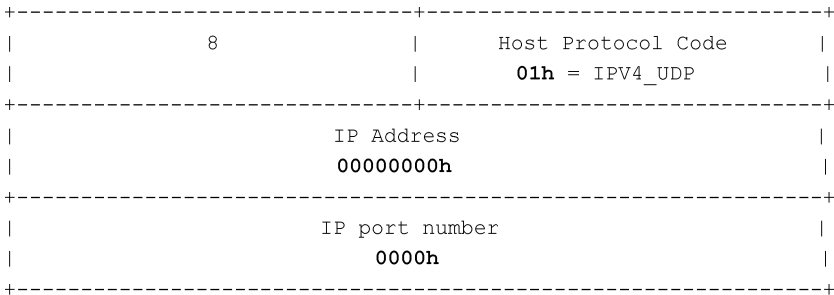


Figure 38 — UDP “Route Back” HPAI binary format

For TCP “Route Back” HPAIs, this means that the KNXnet/IP server shall send his KNXnet/IP frames over the TCP connection that the KNXnet/IP client already has created for this purpose. If no such connection exists (e.g. because the KNXnet/IP client sent the request using UDP), the HPAI is invalid and the request shall be discarded by the KNXnet/IP server. The TCP “Route Back” HPAI binary format is shown in [Figure 39](#).

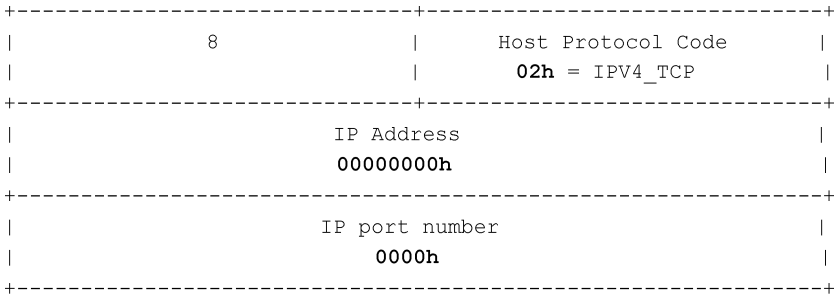


Figure 39 — TCP “Route Back” HPAI binary format

5.2.8.6.3 KNXnet/IP endpoints

a) 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 endpoints too. This is not mandatory. The KNXnet/IP server port number returned in response to a CONNECT_REQUEST may be any valid port number.

It is therefore possible to implement a KNXnet/IP device that uses only one bidirectional 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.

b) 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.

c) Control endpoint:

UDP and TCP are allowed for control endpoint communication. The same port number shall be used for incoming UDP datagrams and TCP connections. KNXnet/IP servers supporting TCP shall accept both, UDP communication and TCP connections to their control endpoints.

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/IP server 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.

d) 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 HPAI structures for KNXnet/IP multicasts. These IP addresses and port numbers are fixed and defined in the corresponding KNXnet/IP service protocol description.

If control endpoint is "Route Back", data endpoint shall also be "Route Back".

If client requested "Route Back" data endpoint, server shall also select "Route Back" data endpoint of same type.

e) 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 Minimum supported services

5.2.9.1 General

5.2.9 provides information on the minimum supported service requirements.

5.2.9.2 Supported services

The supported services for KNXnet/IP and the Internet protocol (IP) are listed in [Tables 14](#) and [15](#). IPv6 will be specified in the future and is listed for completeness only.

Table 14 — Supported services for KNXnet/IP

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
CONNECTIONSTATE_RESPONSE	Server → client	M
DISCONNECT_REQUEST	Client → server	M
	Server → client	
DISCONNECT_RESPONSE	Client → server	M
	Server → client	

Table 15 — Supported services for the Internet protocol (IP)

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

5.3 Device management specification

5.3.1 Use

The “Device management” of the KNXnet/IP specification 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 General

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 General

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.

The KNXnet/IP device endpoints are shown in Figure 40.

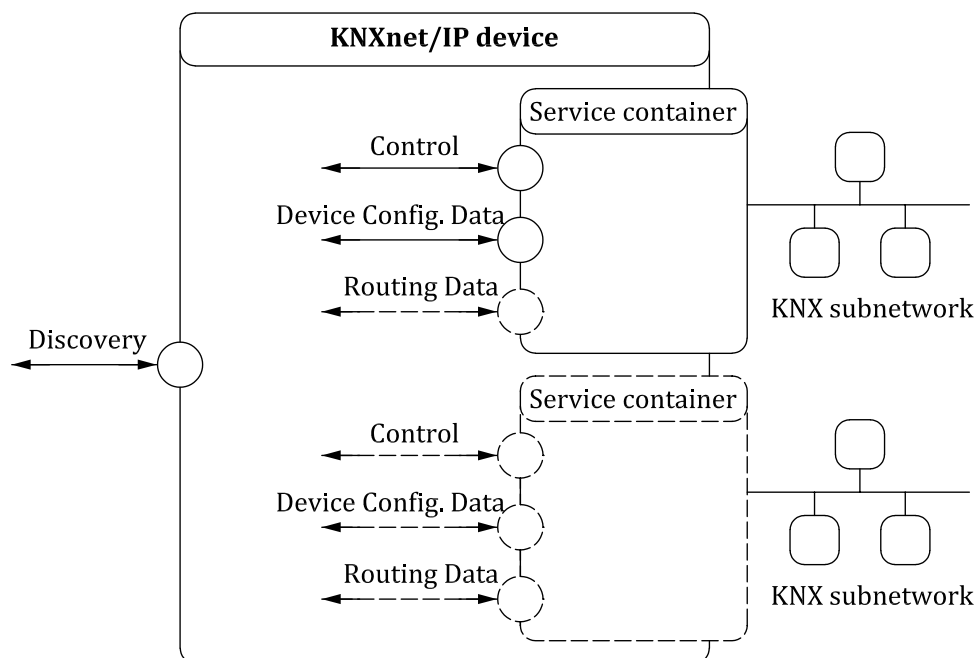


Figure 40 — KNXnet/IP device endpoints

A KNXnet/IP device's service container shall implement a device configuration and management endpoint. Further service endpoints like routing data may be implemented depending on the device function.

The configuration and management endpoint 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. 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.

For TCP communication, no DEVICE_MANAGEMENT_ACK shall be sent. Received DEVICE_MANAGEMENT_ACK frames shall be ignored.

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.

Coming into effect of changed KNXnet/IP Parameters — general rule

The KNXnet/IP Parameters are designed to have immediate effect. Unless specified differently for any further resource, the following general rule shall apply.

The KNXnet/IP server shall assume the changed parameter values either immediately, or at the latest after a restart (power up of the KNX IP part, A_Restart or M_Reset) plus an additional time after the restart has completed of up to 30 s.

The (Management) client shall however expect both the existing, as well as the newly set parameter values and only assume exclusively the new values to have effect after a restart plus the additional time of 30 s.

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. The complete overview of properties that shall be part of the KNXnet/IP parameter object — depending on which part of this document the KNXnet/IP supports — is given in [Annex C](#).

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.

The PID_PROJECT_INSTALLATION_ID is shown in [Figure 41](#).

octet 2								octet 1							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
project number												installation number			

Figure 41 — 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 in [Table 16](#).

Table 16 — IP assignment methods

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 in [Table 17](#).

Table 17 — PID_IP_ASSIGNMENT_METHOD

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, see [Table 18](#).

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 18 — 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 [5.2.8.5](#).

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 [5.2.8.5](#).

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 [5.2.8.5](#).

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 [5.2.8.5](#).

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". See [Table 19](#) for the Device Capabilities.

Table 19 — Device Capabilities

Bit	Description
0	Device management
1	Tunnelling
2	Routing
3	Remote Logging
4	Remote Configuration and Diagnosis
5	Object server
6	Secured communication
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.

The Device State is shown in [Table 20](#).

Table 20 — 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

Table 20 (continued)

Bit	Description
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, see [Table 21](#). 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 21 — Routing Capabilities

Bit	Description
0	Statistics, queue overflow error count, implemented
1	Statistics, transmitted telegram info count, implemented
2	Priority/FIFO implemented
3	Multiple 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 [5.5.3.7.2](#).

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^[2] 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.

5.3.2.5.29 PID_TUNNELLING_ADDRESSES (PID = 79)

- Property name: Tunnelling Addresses;
- Datapoint type: None.

This property shall indicate which individual addresses the management server will use for tunnelling connections from the set of the additional individual addresses and the own individual address. The maximum number of PID_TUNNELLING_ADDRESSES shall equal the number of tunnelling connections that the tunnelling server offers.

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

5.3.2.6 Transport layer interface — Switching to the cEMI transport layer in the cEMI server by opening a KNXnet/IP device management connection

5.3.2.6.1 Use

This shall be implemented by devices that use the KNXnet/IP device management.

5.3.2.6.2 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).

5.3.2.6.3 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_CONFIGURATION_ACK frame.

5.3.2.6.4 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.

5.3.2.6.5 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.

5.3.2.6.6 Controlling sequencing of the cEMI messages

This is guaranteed by the KNXnet/IP device management protocol.

5.3.2.6.7 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.

5.3.3 Implementation rules and guidelines

5.3.3.1 General

This subclause 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 [5.2](#) "Core".

5.3.3.3 Device management

A KNXnet/IP device should implement device management (see [5.1.2.5](#)).

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. Existing services are wrapped in secure wrapper frame and communicated over a secure session.

5.3.3.5 Error handling

5.3.3.5.1 General

Some errors may occur in normal operation, for example 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 be implemented (see 5.3.2.5.21), 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 Table 22 [All values are unsigned, and all counts do not wrap around when the max. is reached (property can then be set to 0).]

Table 22 — 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 General

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, see Table 23.

Table 23 — 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)

Table 23 (continued)

Service name	Code	V.	Description
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. The KNXnet/IP device management CRI binary format is shown in [Figure 42](#).

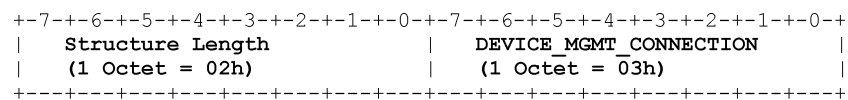


Figure 42 — KNXnet/IP device management CRI binary format

5.3.4.2.4 Connection Response Data Block

The Connection Response Data Block (CRD) shall have no options. The KNXnet/IP device management CRD binary format is shown in [Figure 43](#).

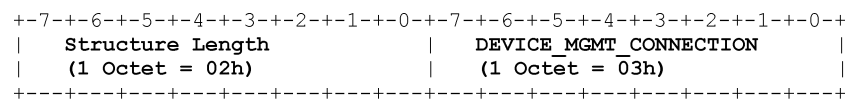


Figure 43 — KNXnet/IP device management CRD binary format

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](#) and [Table 24](#):

Table 24 — Supported cEMI services per device management version

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		

Table 24 (continued)

Supported services	KNXnet/IP	
	Device management version	
	1	2
— 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 Device State property.

M_FuncPropCommand.req, M_FuncPropStateRead.req, and M_FuncPropStateResponse.con are described in [Annex D](#).

5.3.4.2.6 DEVICE_CONFIGURATION_REQUEST

The DEVICE_CONFIGURATION_REQUEST frame binary format is shown in [Figure 44](#).

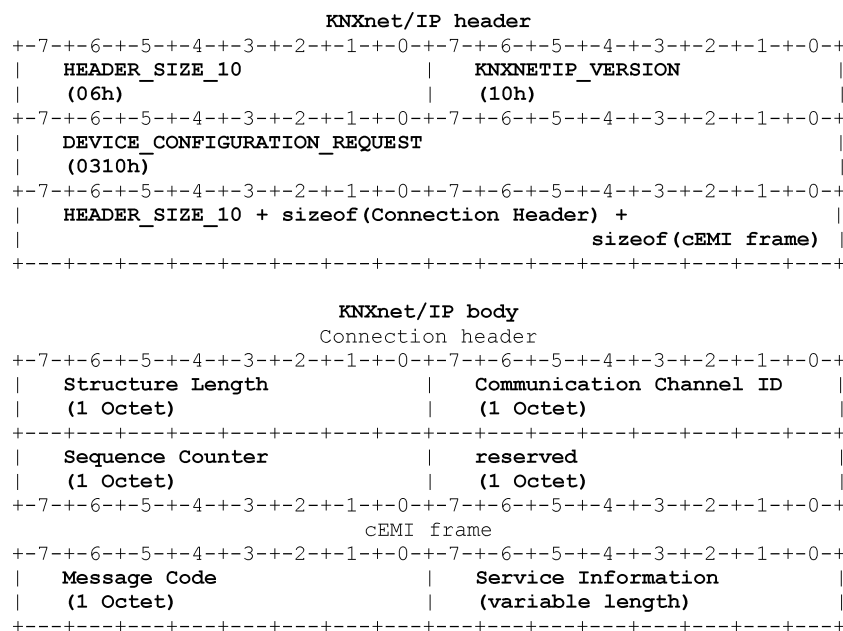


Figure 44 — DEVICE_CONFIGURATION_REQUEST frame binary format

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. The DEVICE_CONFIGURATION_ACK frame binary format is shown in [Figure 45](#) and the configuration status code is given in [Table 25](#).

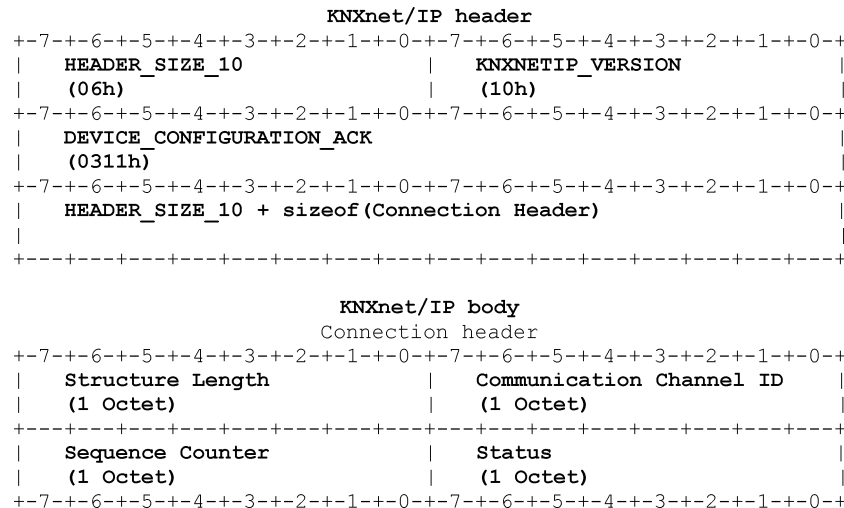


Figure 45 — DEVICE_CONFIGURATION_ACK frame binary format

Table 25 — Configuration status code

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

5.3.5 Minimum profiles

5.3.5.1 General

This clause provides information on the minimum requirements. The service support matrix is given in [Table 26](#).

Table 26 — Service support matrix

Service name	Sent from... to...	Implementation is
DEVICE_CONFIGURATION_REQUEST	Client → server	M
	Server → client	
DEVICE_CONFIGURATION_ACK	Client → server	M
	Server → client	

5.3.5.2 Test cases

5.3.5.2.1 General

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, see [Table 27](#).

Table 27 — 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, see [Table 28](#).

Table 28 — Parameterisation through IP

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 Tunnelling

5.4.1 Use

The "Tunnelling" of the KNXnet/IP specification 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 Clause 6, remote configuration and diagnosis.

5.4.2 Tunnelling of KNX telegrams

5.4.2.1 General

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 46 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 46 B](#).

NOTE Every KNXnet/IP tunnelling connection uses its own (unique) individual address and hence logically appears as another device on the KNX bus.

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 46 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](#)). The tunnelling connections and KNX individual addresses in the KNXnet/IP server are shown in [Figure 46](#).

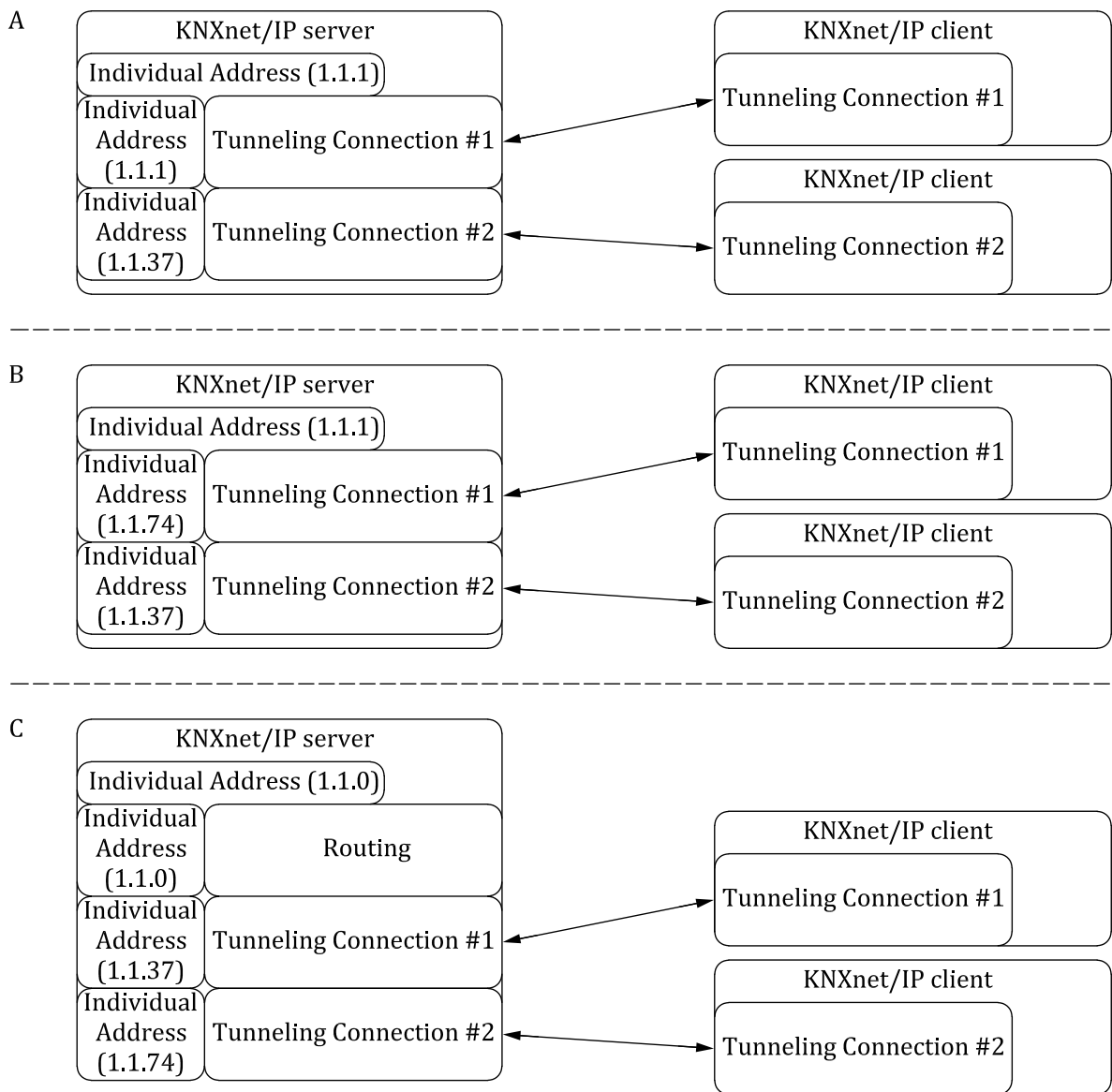


Figure 46 — Tunnelling connections and KNX individual addresses in the KNXnet/IP server

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 specification 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 $t_{\text{Tunnelling_protocol_transfer time}} \ll t_{\text{KNX_transfer_time-outs}}$.

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 specification. 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 specification.

5.4.2.6 Datagram confirmation

TUNNELLING_REQUEST frames shall be confirmed using TUNNELLING_ACK frames.

For TCP communication, no TUNNELLING_ACK shall be sent. Received TUNNELLING_ACK frames shall be ignored.

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

5.4.3.1 General

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.

Under KNXnet/IP secure communication conditions KNXnet/IP tunnelling clients can access specific management and configuration features via KNXnet/IP tunnelling feature services.

The tunnelling client shall use the tunnelling feature services TUNNELLING_FEATURE_GET and TUNNELLING_FEATURE_SET to read and respectively write features related to the tunnelling interface and the tunnelling host device in the tunnelling host device. The tunnelling server shall use the service TUNNELLING_FEATURE_INFO to spontaneously report to the tunnelling client about relevant changes in the state of itself or of the tunnelling connection.

Tunnelling feature services shall be confirmed by the receiver identically as the TUNNELLING_REQUEST service.

NOTE This is, on UDP by a TUNNELLING_ACK frame and on TCP no TUNNELLING_ACK frame.

In all services, the interface feature shall be identified by the interface feature identifier.

5.4.3.2 TUNNELLING_FEATURE_GET

The tunnelling client shall initiate the service TUNNELLING_FEATURE_GET to read the value of an interface feature from the tunnelling server. The tunnelling server shall at the latest 3 s after confirming the TUNNELLING_FEATURE_GET respond with a TUNNELLING_FEATURE_RESPONSE containing the value of the interface feature.

If the tunnelling server does not respond to the TUNNELLING_FEATURE_GET then the tunnelling client may or may not repeat the request.

5.4.3.3 TUNNELLING_FEATURE_SET

The tunnelling client shall initiate the service TUNNELLING_FEATURE_SET to set the value of an interface feature in a tunnelling server. The tunnelling server shall accept the value and — unless specified differently — apply the new value immediately, this is, before the TUNNELLING_FEATURE_RESPONSE is sent or before routing the next TUNNELLING_REQUEST on the bus. The tunnelling server shall at the latest 3 s after confirming the TUNNELLING_FEATURE_GET respond with a TUNNELLING_FEATURE_RESPONSE frame containing the written value and a return code.

- If the tunnelling client requests to set a feature that is available for it but which is read-only, then the tunnelling server shall respond with E_ACCESS_READ_ONLY.
- If the tunnelling client requests to set a feature to a value with a wrong size, then the tunnelling server shall respond with E_DATA_TYPE_CONFLICT.
- If the tunnelling client requests to set the feature to a value that is not valid, then the tunnelling server shall respond with E_DATA_VOID or alternatively with E_DATA_MAX or E_DATA_MIN as appropriate. The field feature value shall repeat the feature value from the request.

5.4.3.4 TUNNELLING_FEATURE_INFO

The tunnelling server shall only send the TUNNELLING_FEATURE_INFO if the interface feature “Interface Feature Info Service Enable” has the value “Enable”. See [5.4.4.4.13 TUNNELING_FEATURE_INFO](#).

If enabled, then the tunnelling server shall initiate the service TUNNELLING_FEATURE_INFO to report on any relevant change of an interface feature to the tunnelling client.

The tunnelling server shall by default report for all its supported interface features. There is no standard way for the tunnelling client to disable the reporting of an individual interface feature. This may be possible through the normal configuration of the tunnelling server device by a management client, e.g. over KNXnet/IP device management.

The tunnelling server shall address the TUNNELLING_FEATURE_INFO frame to the data endpoint of the established tunnelling connection.

5.4.3.5 Error handling for feature services

- If a Tunnelling Feature service is requested from a tunnelling server that does not support the Tunnelling Feature services, then the tunnelling server shall ignore this service request

NOTE 1 This means that no TUNNELLING_ACK will be sent. This is in line with the general requirement that a received invalid data packet is ignored. If this tunnelling feature service is conveyed over a TCP connection, then however obviously a TCP acknowledge will be sent.

NOTE 2 The tunnelling client can in fact know whether the tunnelling server supports the tunnelling feature services, by sending a DESCRIPTION_REQUEST to it and verifying if the DESCRIPTION_RESPONSE contains the TUNNELLING_INFO DIB or not.

- If the requested tunnelling feature service is not available for the requested interface feature then the tunnelling server shall respond with E_ACCESS_DENIED (FCh). The “*Tunnelling Feature service availability matrixes*” in the below interface feature definitions indicate for each tunnelling feature service the availability of that interface feature and the possible conditions.
- If a tunnelling feature service is transported over an unknown communication channel ID, then this shall be ignored.
- If an unknown or not supported interface feature is addressed in the TUNNELLING_FEATURE_SET or TUNNELLING_FEATURE_GET then the tunnelling server shall respond with the return code E_ADDRESS_VOID and the field feature value shall be omitted.

5.4.4 Frame structures

5.4.4.1 General

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 [5.1](#) “Overview” for a list of valid KNXnet/IP common constants.

5.4.4.3 Common error codes

Refer to [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, for example ETS, to the KNXnet/IP server. Refer to [5.2](#) “Core” for details.

5.4.4.4.2 KNXnet/IP services

The following [table 29](#) lists all valid KNXnet/IP service types for KNXnet/IP tunnelling.

Table 29 — Tunnelling KNXnet/IP service type identifiers

Service name	Code	V.	Description
TUNNELLING_REQUEST	0420h	1	Used for sending and receiving single 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.
TUNNELLING_FEATURE_GET	0422h	2	Used by the KNXnet/IP tunnelling client to read out the value of a feature from the KNXnet/IP tunnelling server.
TUNNELLING_FEATURE_RESPONSE	0423h	2	Used by the KNXnet/IP tunnelling server to respond to a feature set or gotten by the KNXnet/IP tunnelling client.
TUNNELLING_FEATURE_SET	0424h	2	Used by the KNXnet/IP tunnelling client to set the value of a feature of the KNXnet/IP tunnelling server.
TUNNELLING_FEATURE_INFO	0425h	2	Used by the KNXnet/IP tunnelling server to inform the KNXnet/IP tunnelling client on a value of an interface feature.

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 basic Connection Request Information (CRI) contains the requested Tunnelling KNX layer, see [Figure 47](#).

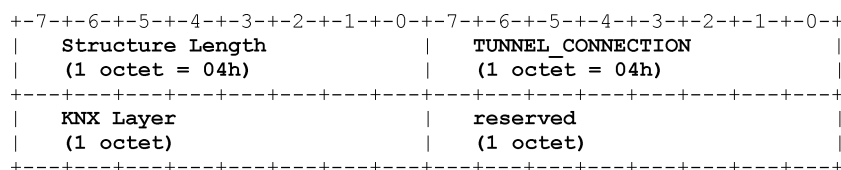


Figure 47 — Tunnelling CRI binary format

The KNXnet/IP tunnelling server assigns to the tunnelling connection an arbitrary individual address from the pool of available IAs upon connecting and returns to the client which IA its requested tunnelling connection will use this time.

5.4.4.4.5 Extended Connection Request Information (Extended CRI)

In KNX data security, a secure device holds the IAs of its trusted communication partners in its Security Individual Address Table. Therefore, it is necessary that the tunnelling client gets precise control over the IA that is used for its tunnelling connection.

The Extended Connection Request Information (CRI) shall contain the requested tunnelling KNX layer and the IA that the client requests to be used on the field medium for the requested tunnelling connection, see [Figure 48](#) and [Tables 30](#) and [31](#).

NOTE It is not possible for the tunnelling client to enter a “don’t care” value for the IA. If the tunnelling client wants to open a tunnelling connection without requesting any specific IA, it has to use a basic CRI.

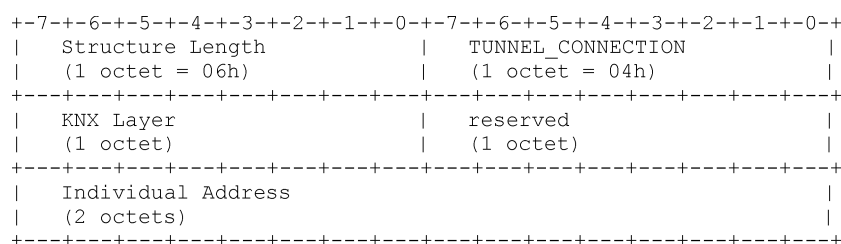


Figure 48 — Tunnelling extended CRI binary format

Table 30 — 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 31 — 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.

5.4.4.4.6 Connection Response Data Block (CRD)

The Connection Response Data Block (CRD) contains the KNX individual address assigned to this tunnelling connection, see [Figure 49](#).

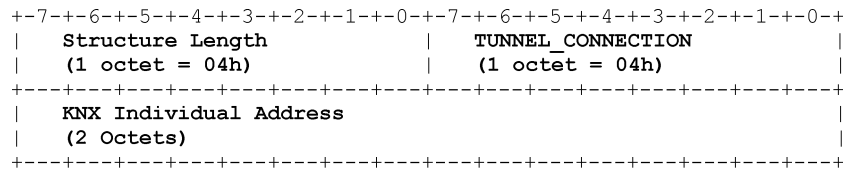


Figure 49 — Tunnelling CRI binary format

5.4.4.4.7 Connection header

The tunnelling connection header binary format is shown in [Figure 50](#).

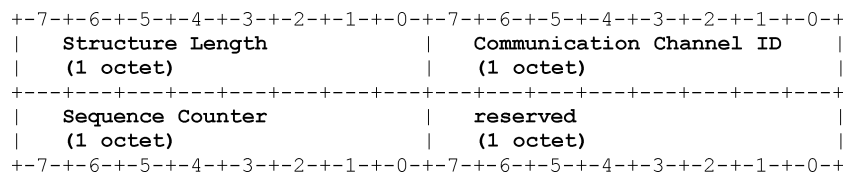


Figure 50 — Tunnelling connection header binary format

5.4.4.4.8 TUNNELLING_REQUEST

The TUNNELLING_REQUEST frame binary format is shown in [Figure 51](#).

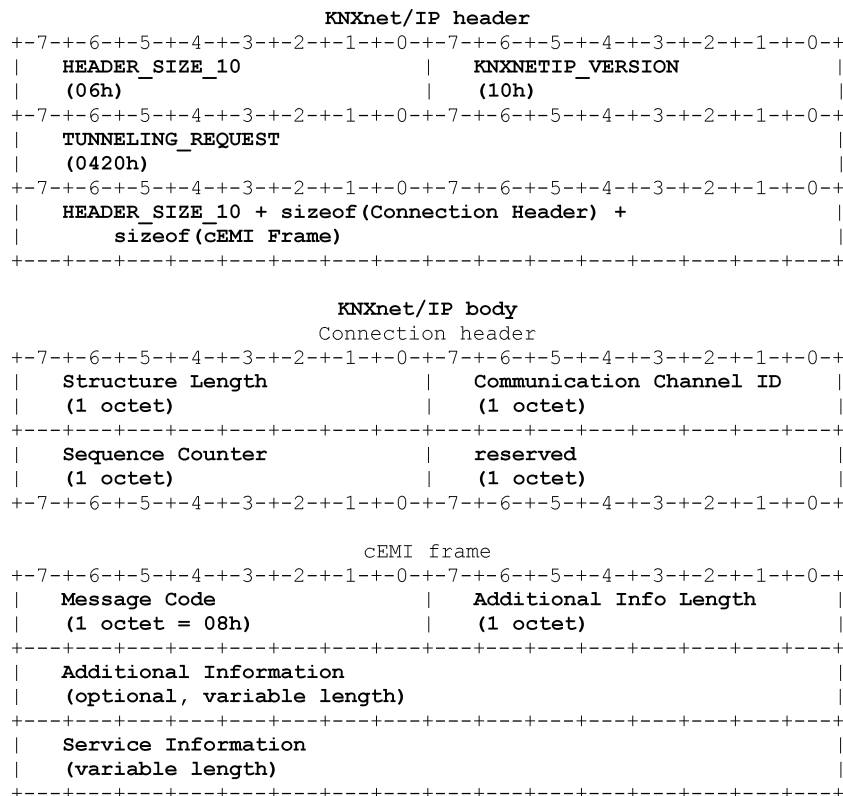


Figure 51 — TUNNELLING_REQUEST frame binary format

5.4.4.4.9 TUNNELLING_ACK

The TUNNELLING_ACK frame binary format is shown in [Figure 52](#).

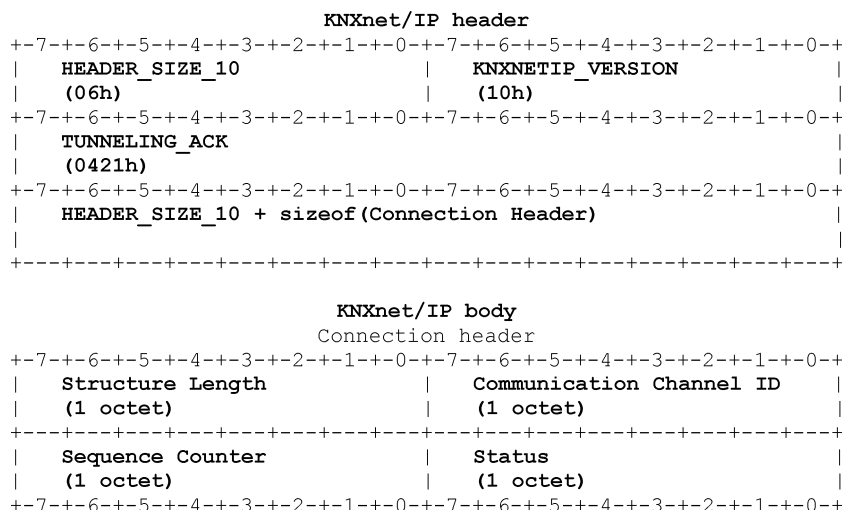


Figure 52 — TUNNELING_ACK frame binary format

5.4.4.4.10 TUNNELING_FEATURE_GET

The TUNNELING_FEATURE_GET frame binary format is shown in [Figure 53](#).

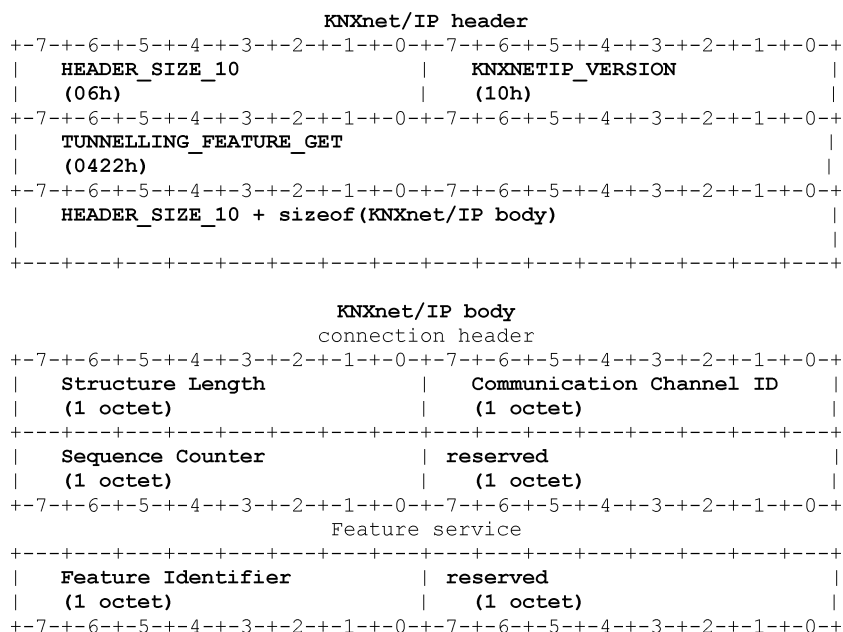


Figure 53 — TUNNELING_FEATURE_GET frame binary format

5.4.4.4.11 TUNNELING_FEATURE_RESPONSE

The TUNNELING_FEATURE_RESPONSE frame binary format is shown in [Figure 54](#).

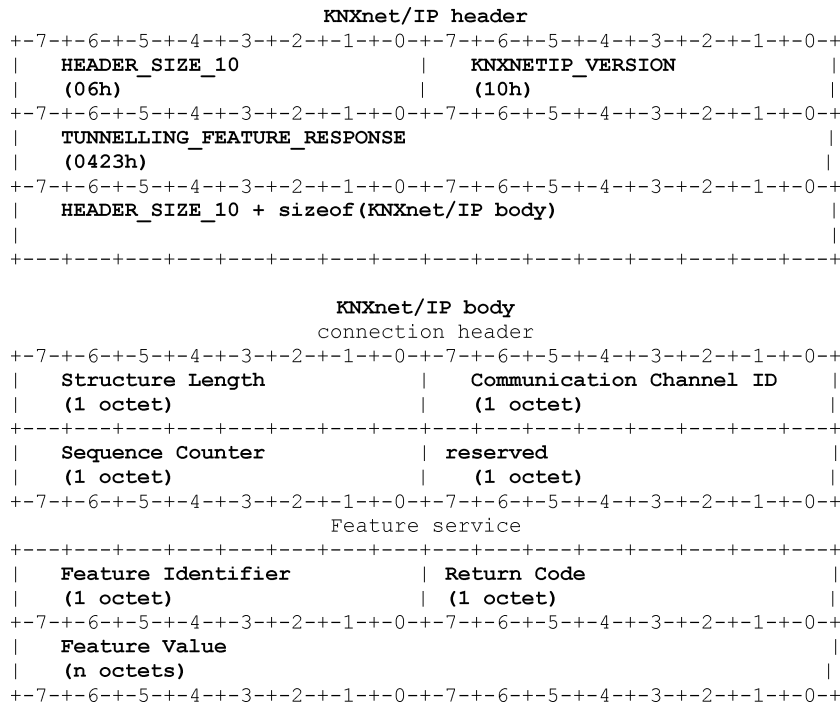


Figure 54 — TUNNELLING_FEATURE_RESPONSE frame binary format

5.4.4.4.12 TUNNELLING_FEATURE_SET

The TUNNELLING_FEATURE_SET frame binary format is shown in [Figure 55](#).

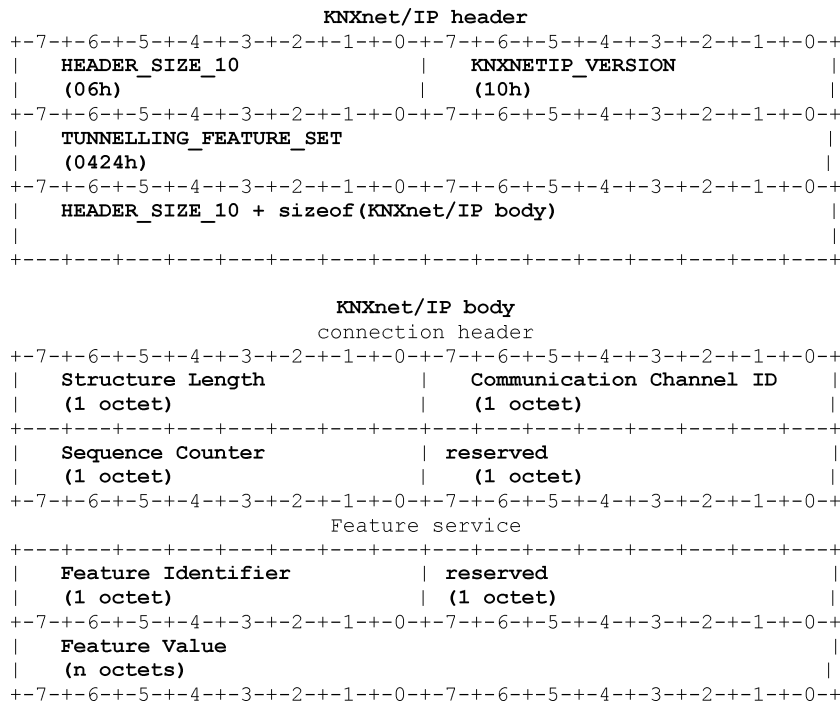


Figure 55 — TUNNELLING_FEATURE_SET frame binary format

5.4.4.4.13 TUNNELLING_FEATURE_INFO

The TUNNELLING_FEATURE_INFO frame binary format is shown in [Figure 56](#).

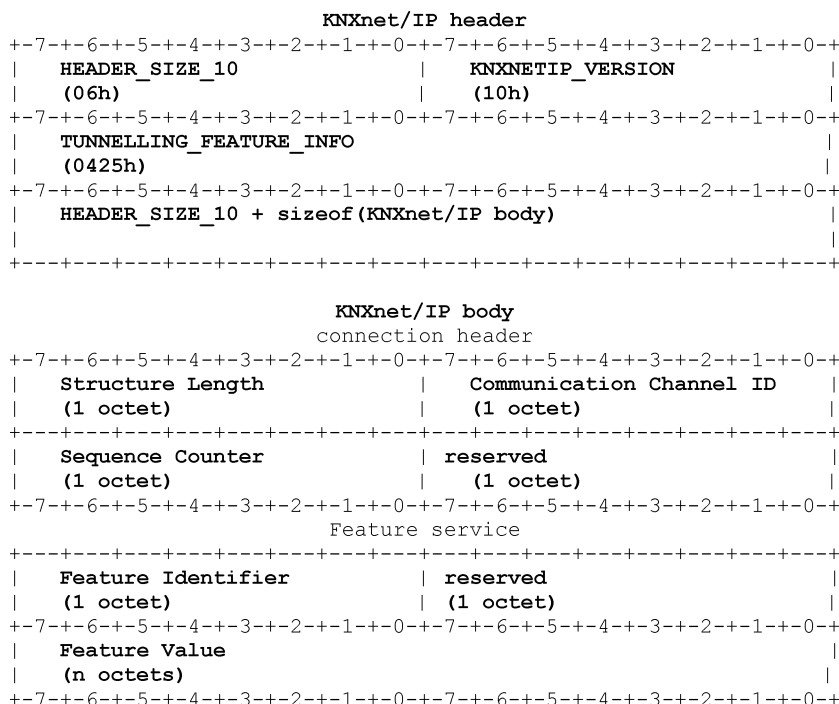


Figure 56 — TUNNELLING_FEATURE_INFO frame binary format

5.4.4.4.14 Interface feature overview

The interface feature overview is given in [Table 32](#).

Table 32 — Interface feature overview

Feature identifier	Feature name	Description	Data length
01	Supported EMI type	Getting the supported EMI type(s)	2 octets (B_{16})
02	Host Device Device Descriptor Type 0	Getting the local device descriptor type 0 for possible local device management.	2 octets ($U_4U_4U_4U_4$)
03	Bus connection status	Getting and informing on the bus connection status.	1 bit (B_1)
04	KNX Manufacturer Code	Getting the manufacturer code of the Bus Access server.	2 octets (U_{16})
05	Active EMI type	Getting and setting the EMI type to use.	1 octet (N_8)
06	Interface Individual Address	IA used by the Bus interface	2 octets (U_{16})
07	Max. APDU Length	The maximal APDU-length that can be transported over this KNXnet/IP tunnelling connection.	2 octets (U_{16})
08	Interface Feature Info service Enable	Control the use of the info feature service by the interface.	1 bit (B_1)

5.4.4.4.15 Tunnelling information DIB

The tunnelling information DIB (see [Figure 57](#)) shall be a variable length DIB (depending on the amount of tunnelling slots the server can manage) that consists of a list of 4 octet elements each containing the 2 octet individual address and 2 status octets.

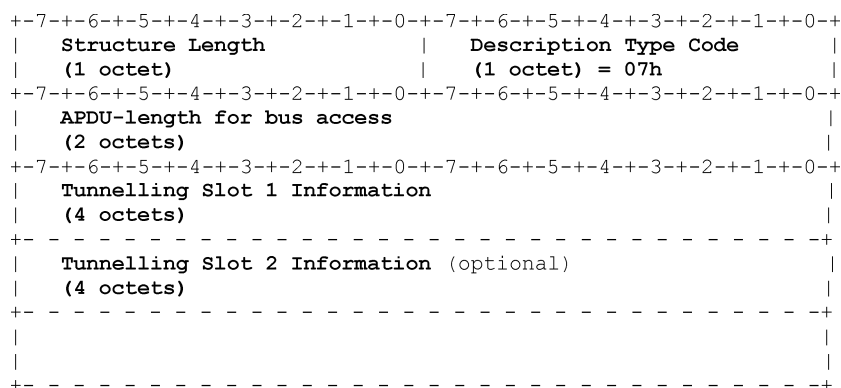


Figure 57 — Tunnelling information DIB

- APDU-length for bus access:

This shall be the maximal APDU-length that the Tunnelling Interface supports for bus access. The value of this field shall equal the value of PID_MAX_INTERFACE_APDU_LENGTH (PID 68) in the cEMI server object.

- Tunnelling slot information:

Each tunnelling slot information shall be structured as follows in [Figure 58](#).

octet 4 (MSB)	octet 3	octet 2	octet 1 (LSB)
Individual Address		Status	

Figure 58 — Tunnelling slot information

- Individual address (octet 4 MSB and octet 3):

This shall be the individual address that is currently assigned to the tunnelling slot.

- Status (octet 2 and octet 1 LSB):

This octet shall describe the current state of the tunnelling slot, see [Figure 59](#).

Bit Nr.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Meaning	r	r	r	r	r	r	r	r	r	r	r	r	r	U	A	F
Value	1	1	1	1	1	1	1	1	1	1	1	1	1			

Figure 59 — Tunnelling slot status

NOTE The encoding of the fields F, U, A and the reserved bits is chosen so that if all bits are set, the tunnelling slot is usable.

- Bit 15 to bit 3:

These fields are reserved. The server shall set these fields to 1.

- Bit 2: Usable (U):

This bit shall indicate whether this tunnelling slot is currently usable or not.

- Mandatory criterion:
If at least one other tunnelling slot is occupied and has the same individual address as this tunnelling slot, then this bit shall be cleared.
- Implementation specific criteria:
If the Tunnelling Slot is not available because of one or more implementation specific criteria, then this shall be signalled in this field.
- Encoding:
 - 0: This slot is not usable.
 - 1: This slot is usable.
- Bit 1: Authorised (A):
This bit shall give indications about the authorisation of this tunnelling slot.
- If this tunnelling information DIB is sent from a server to a client during a KNX IP secure session, then the server shall in this bit indicate whether the client would be authorised to use the tunnelling slot or not.
 - 0: The client would not be authorised.
 - 1: The client would be authorised.
- If this tunnelling information DIB is sent outside a KNX IP secure session, then the server shall in this bit indicate whether this tunnelling slot requires authorisation or not.
 - 0: authorisation is required.
 - 1: no authorisation is required.
- Bit 0: Free (F):
This bit shall indicate whether this tunnelling slot is currently free or not.
 - 0: This slot is not free.
 - 1: This slot is free.
- Support in messages:

[Table 33](#) contains the requirements on the inclusion of the tunnelling information DIB.

Table 33 — Requirements on the inclusion of the tunnelling information DIB

Message	Tunnelling information DIB
SEARCH_RESPONSE	Not allowed
SEARCH_RESPONSE_EXTENDED	As specified in core
DESCRIPTION_RESPONSE	Not allowed

5.4.5 Minimum profiles

This clause provides information on the minimum requirements, see [Table 34](#).

Table 34 — 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 Routing

5.5.1 Use

"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 General

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.

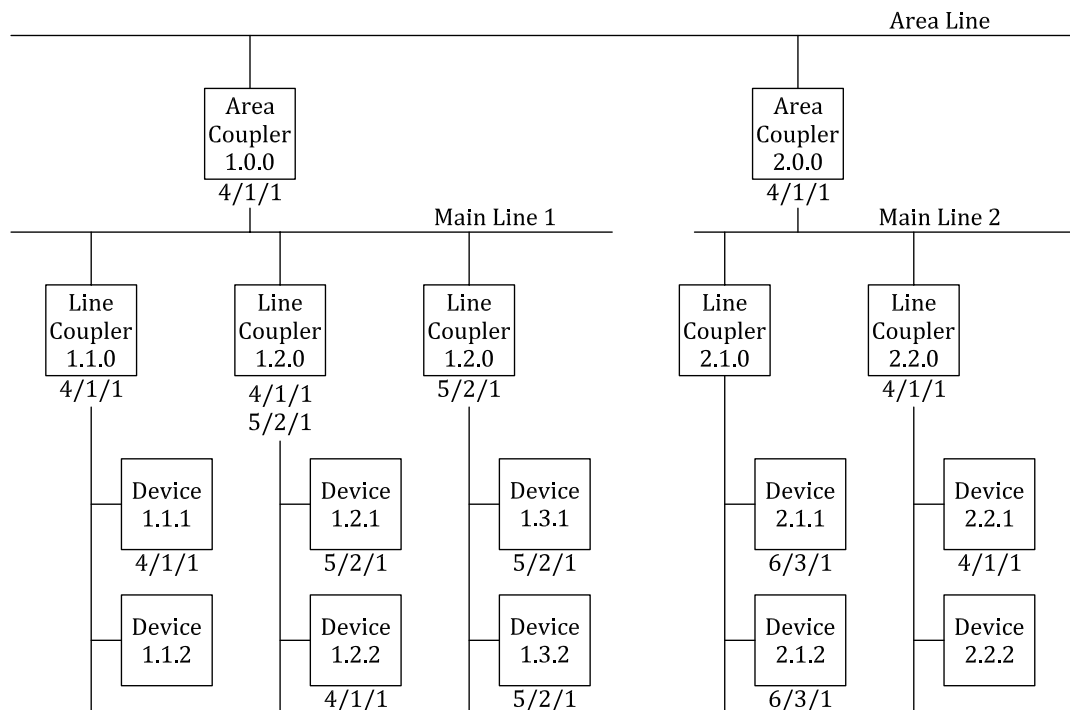


Figure 60 — KNX group telegram routing

In [Figure 60](#), 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 occurs for point-to-point connectionless 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).

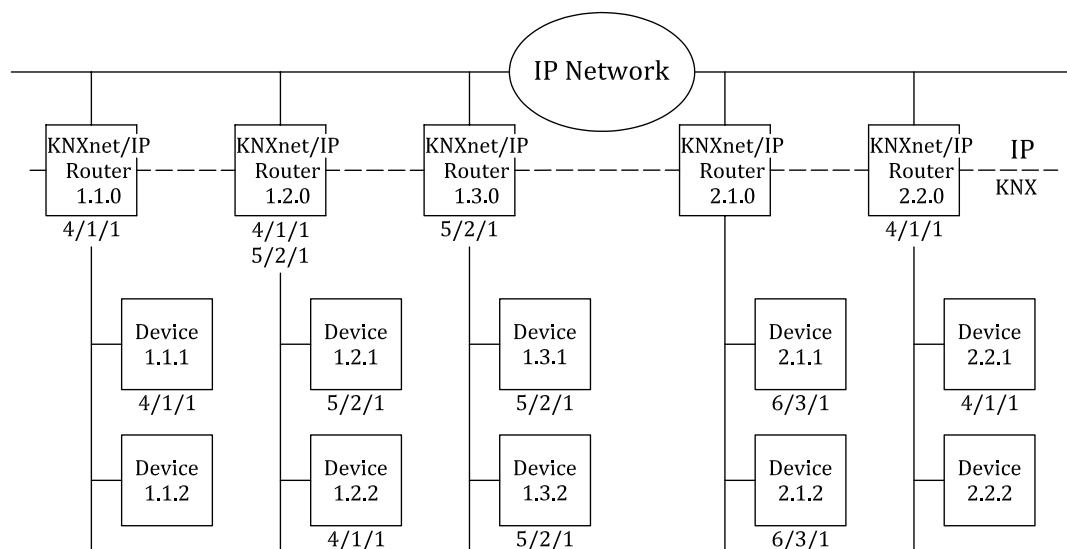


Figure 61 — KNXnet/IP group telegram routing

In [Figure 61](#), the same KNX devices as in [Figure 60](#) 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.

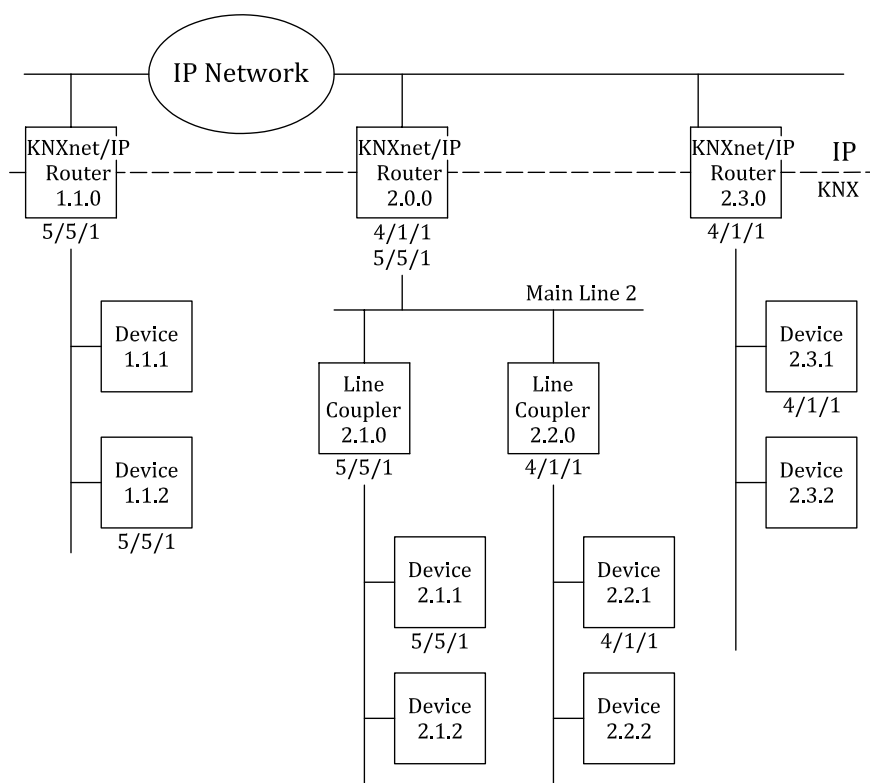


Figure 62 — Mixed topology (undesired subnetwork addressing)

Figure 62 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 the 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 63 shows a mixed topology of desired subnetwork address assignment.

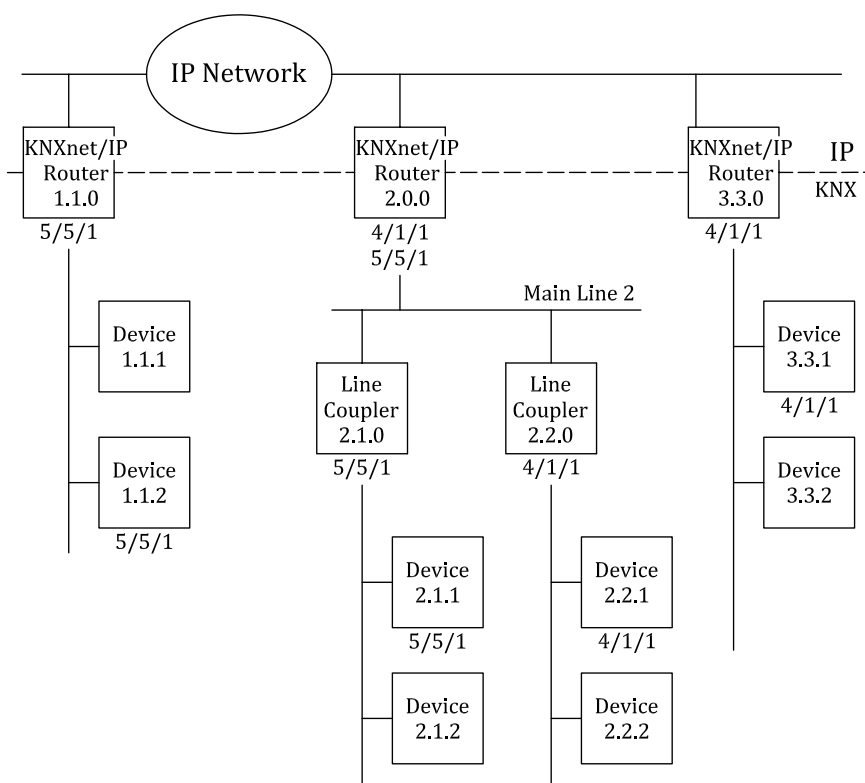


Figure 63 — Mixed topology (desired subnetwork address assignment)

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 $y = [1 \text{ to } 15]$. 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 $y = [1 \text{ to } 15]$.

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) to (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 KNX (TP, RF, or PL) 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 KNX (TP, RF, or PL) installation a KNXnet/IP router is assigned to.

A KNXnet/IP router may support routing from one KNX (TP, RF, or PL) 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 that it may lose datagrams if they do not stop sending. The flow control with `ROUTING_BUSY` is shown in Figure 64.

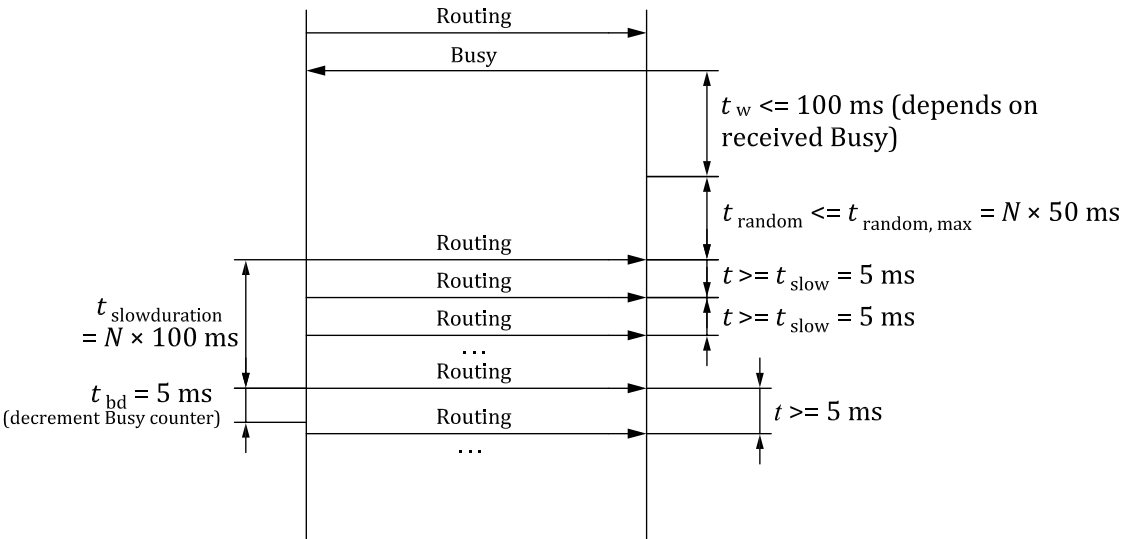


Figure 64 — Flow control with `ROUTING_BUSY`

— 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.

NOTE 1 The recommended values are based on a system simulation assuming that up to 255 devices send 50 ROUTING_INDICATION datagrams per second.

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.

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 calculated with [Formulae \(1\)](#) and [\(2\)](#):

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

$$t_{random} = [0 \dots 1]_{random} \cdot N \cdot 50 \text{ ms}; \{0 \leq t_{random} \leq N \cdot 50 \text{ ms}\} \quad (2)$$

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 have passed since the last ROUTING_BUSY and decremented by one every $t_{bd} = 5 \text{ ms}$ after $t_{slowduration}$ has elapsed. Use [Formula \(3\)](#) to calculate $t_{slowduration}$.

NOTE 2 The value 10 ms 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.

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

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.

5.5.2.4 IP system broadcast

5.5.2.4.1 General

KNX employs a system broadcast to reach all devices within a KNX (TP, RF, or PL) system for system management purposes.

The IP system broadcast fulfils the same purpose for communication over IP networks.

5.5.2.4.2 IP system broadcast frame

An IP system broadcast frame shall be identified by KNXnet/IP service type ROUTING_SYSTEM_BROADCAST (0533h). ROUTING_SYSTEM_BROADCAST frames shall always be sent on the system setup multicast address. Even if secure routing is enabled, a ROUTING_SYSTEM_BROADCAST shall not be wrapped in a SECURE_WRAPPER.

For the cEMI message inside the IP system broadcast frame, the following shall hold true:

- The message code shall be L_Data.ind.
- The SB bit (bit 4) in the cEMI control field 1 as specified in [Annex D](#) shall be set to 0. This value already designates system broadcasts on open media.
- The destination address type bit (bit 7) in the cEMI control field 2 shall be 1, and the destination address shall be 0 (broadcast).
- If the cEMI message is an S-A_Data-PDU, S-A_Sync_Request-PDU or S-A_Sync_Response-PDU, the SBC bit in the Security Control Field (SCF) shall be set to 1.

A receiver shall ignore ROUTING_SYSTEM_BROADCAST frames not received on the system setup multicast address or containing a cEMI message not fulfilling these conditions.

A sender shall not set these fields to values not fulfilling these conditions.

5.5.2.4.3 Handling IP system broadcasts in KNXnet/IP routers

a) IP system broadcast routing mode:

The handling of system broadcasts shall depend on the “*IP System Broadcast Routing Mode*” of the KNXnet/IP router.

b) Routing from TP1 to IP:

If the IP system broadcast routing mode of a KNXnet/IP router has the value “Enable”, received TP1 broadcast frames matching one of the following rules shall be forwarded as IP system broadcast frames on the IP side instead of the normal routing handling.

- A_SystemNetworkParameter_Read-PDU with object_type = 0 (Device Object), PID = 11 (PID_SERIAL_NUMBER) and operand = 01h.
- A_DomainAddressSerialNumber_Write-PDU with 4 octet *Domain Address*
- S-A_Sync_Request-PDU with the SBC bit in the Security Control Field (SCF) equal to 1 and the tool access (T) bit in the Security Control Field (SCF) equal to 1.
- S-A_Data-PDU with authentication and confidentiality and with the SBC bit in the Security Control Field (SCF) equal to 1 and the tool access (T) bit in the Security Control Field (SCF) equal to 1.

Even if secure routing is enabled, this ROUTING_SYSTEM_BROADCAST shall not be wrapped in a SECURE_WRAPPER.

Even if the KNXnet/IP router supports only IP system broadcast but does not support security itself it still shall handle the indicated S-A_Sync_Request-PDUs and S-A_Data-PDUs as specified above.

If the IP system broadcast routing mode of a KNXnet/IP router does not have the value “Enable”, or if a received TP1 broadcast frame does not match the above rules, it shall be forwarded to IP according to the normal routing handling.

c) Routing from IP to TP1:

If the IP system broadcast routing mode of the KNXnet/IP router has the value “Enable”, a received IP system broadcast frame with the contained cEMI message matching one of the following rules shall be forwarded to TP1 as broadcast.

- A_SystemNetworkParameter_Response-PDU with object_type = 0 (Device Object), PID = 11 (PID_SERIAL_NUMBER) and operand = 01h and the tool access (T) bit in the Security Control Field (SCF) equal to 1.
- S-A_Sync_Response-PDU with the SBC bit in the Security Control Field (SCF) equal to 1 and the tool access (T) bit in the Security Control Field (SCF) equal to 1.

If the IP system broadcast routing mode of a KNXnet/IP router does not have the value “Enable”, or if a received IP system broadcast frame does not match the above rules, it shall not be forwarded to TP1.

d) hop_count:

The handling of the hop_count field as specified in [Annex D](#) shall apply also for the special system broadcast routing cases for routing from TP1 to IP and for routing from IP to TP1 described above.

e) Handling in the KNXnet/IP router’s own management server stack:

If the IP system broadcast routing mode of a KNXnet/IP router does not have the value “Enable”, a received IP system broadcast frame from the IP side shall — in addition to the routing behaviour as specified in [5.5.2.4.3 c\)](#) — be handled by the KNXnet/IP router in the same way as by an IP end device ([5.5.2.4.4](#)).

5.5.2.4.4 Handling system broadcasts in IP end devices

a) General:

This subclause is also applicable to the management aspect of KNXnet/IP routers.

b) Handling in data link layer:

If the KNXnet/IP server receives an IP system broadcast frame, the frame shall be passed as L_SystemBroadcast.ind to the local network layer independently of the configured KNXnet/IP routing multicast address, even if secure routing is enabled.

c) Handling in management:

1) Communication mode:

Regarding the reaction to the frames specified in the following clauses, the management server shall handle IP system broadcast frames as defined in [5.5.2.4.2](#) and broadcast frames in a uniform way. In this clause, “broadcast frame” shall mean:

- if routing security is not enabled in the management server: a ROUTING_INDICATION, or
- if routing security is enabled in the management server: a SECURE_WRAPPER encapsulating a ROUTING_INDICATION

and sent on the routing multicast address and containing a broadcast cEMI message; i.e. a cEMI message with all of the following properties:

- the message code is L_Data.ind;
- the SB bit (bit 4) in the cEMI control field 1 as specified in [Annex D](#) is 1;
- the destination address type bit (bit 7) in the cEMI control field 2 is 1, and the destination address shall be 0 (broadcast).
- If the cEMI message is an S-A_Data-PDU, S-A_Sync_Request-PDU or S-A_Sync_Response-PDU, the SBC bit in the Security Control Field (SCF) is 0.

NOTE The routing multicast address and the system setup multicast address are two different variables, but can have the same value.

Any response shall be sent in the same communication mode as the request.

2) A_SystemNetworkParameter_Read/A_NetworkParameter_Read:

If the cEMI message contained in an IP system broadcast frame is an A_SystemNetworkParameter_Read-PDU with object_type = 0 (Device Object), PID = 11 (PID_SERIAL_NUMBER) and operand = 01h, the management server shall respond with its KNX serial number if its programming mode is “enabled”. The response shall be sent as IP system broadcast.

If the cEMI message contained in a broadcast frame is an A_NetworkParameter_Read-PDU with object_type = 0 (Device Object), PID = 11 (PID_SERIAL_NUMBER) and operand = 01h, the management server shall respond with its KNX serial number if its programming mode is “enabled”. The response shall be sent as broadcast.

3) S-A_Sync_Request:

If the cEMI message contained in an IP system broadcast frame or broadcast frame is an S-A_Sync_Request-PDU, the management server shall respond with an S-A_Sync_Response-PDU. The response shall be sent in the same communication mode as the request.

4) A_DomainAddressSerialNumber_Write:

If the security mode in the management server is not enabled and the cEMI message contained in an IP system broadcast frame or a broadcast frame is an A_DomainAddressSerialNumber_Write-PDU with 4 octet domain address, the management server shall set its KNXnet/IP routing multicast address (PID_ROUTING_MULTICAST_ADDRESS) to the 4 octet domain address. The management client shall wait 1 s before assuming that the management server started the process of accepting the new value. Only after 1 s the management client may try to access the management server on the new KNXnet/IP routing multicast address and repeat every second until successful or a timeout of 60 s is reached.

If the security mode in the management server is enabled or if the size of the domain address is not 4 octets, this frame shall be ignored.

5) 4 octet domain address:

If the cEMI message contained in an IP system broadcast frame or a broadcast frame is an S-A_Data-PDU with authentication and confidentiality containing an A_DomainAddressSerialNumber_Write-PDU with 4 octet domain address, the management server shall set its KNXnet/IP routing multicast address (PID_ROUTING_MULTICAST_ADDRESS) to the 4 octet domain address and the routing security version (PID_SECURED_SERVICES property array index 5) to 0. The management server shall disable sending and receiving of encrypted frames on the IP backbone. The management server shall not change its security mode.

The management client shall wait 1 s before assuming that the management server has started the process of accepting the new values. Only after 1 s, the management client may try to access the management server on the new KNXnet/IP routing multicast address with plain frames and repeat every second until successful or a timeout of 60 s is reached.

6) 21 octet domain address:

If the cEMI message contained in an IP system broadcast frame or a broadcast frame is an S-A_Data-PDU with authentication and confidentiality containing an A_DomainAddressSerialNumber_Write-PDU with 21 octet domain address, and the management server supports the routing security version indicated in the message, then the management server shall set its KNXnet/IP routing multicast address (PID_ROUTING_MULTICAST_ADDRESS) to the 4 octet domain address, the routing security version (PID_SECURED_SERVICES property array index 5) to the value of the 1 octet routing security version field, and its backbone key (PID_BACKBONE_KEY) to the 16 octet backbone key. In addition, device security mode shall be.

The management client shall wait 1 s before assuming that the management server started the process of accepting the new value. Only after 1 s the management client may try to access the management server on the new KNXnet/IP routing multicast address and repeat every second until successful or a timeout of 60 s is reached. The management client shall use the 21 octet version in case of secure backbone, even if the desired backbone key is already loaded into PID_BACKBONE_KEY.

If the management server does not support the routing security version indicated in the message, it shall ignore the frame.

7) Other IP system broadcast frames:

All other IP system broadcast frames shall be ignored.

5.5.2.4.5 Error and exception handling

If an incoming system broadcast frame cannot be handled by the KNXnet/IP router or IP end device, for example due to resource limitations, it shall be ignored. Specifically, the ROUTING_BUSY mechanism shall not be applied. In the opposite direction, received ROUTING_BUSY frames shall not influence the sending of *IP system broadcast frames*.

5.5.3 Implementation rules and guidelines

5.5.3.1 General

This subclause 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](#).

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:

- 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).
- 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, 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.

5.5.3.10 Security

Secured communication with KNXnet/IP routing frames is specified in [5.7](#).

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 General

Some errors may occur in normal operation, for example 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 [5.3.2.5](#). 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 [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 35](#) lists the KNXnet/IP routing services.

Table 35 — 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.
ROUTING_SYSTEM_BROADCAST	0533h	1	Used for configuration of devices to be added to a KNX/IP domain.

5.5.5.2 ROUTING_INDICATION

The ROUTING_INDICATION frame binary format is shown in [Figure 65](#).

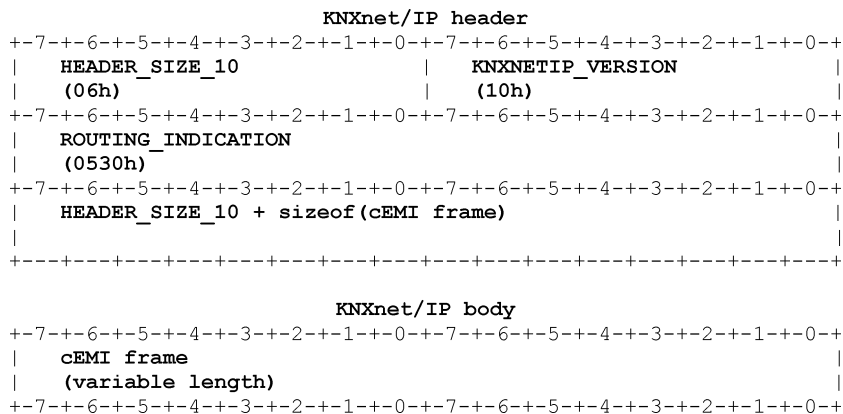


Figure 65 — ROUTING_INDICATION frame binary format

5.5.5.3 ROUTING_LOST_MESSAGE

The ROUTING_LOST_MESSAGE frame binary format is shown in [Figure 66](#).

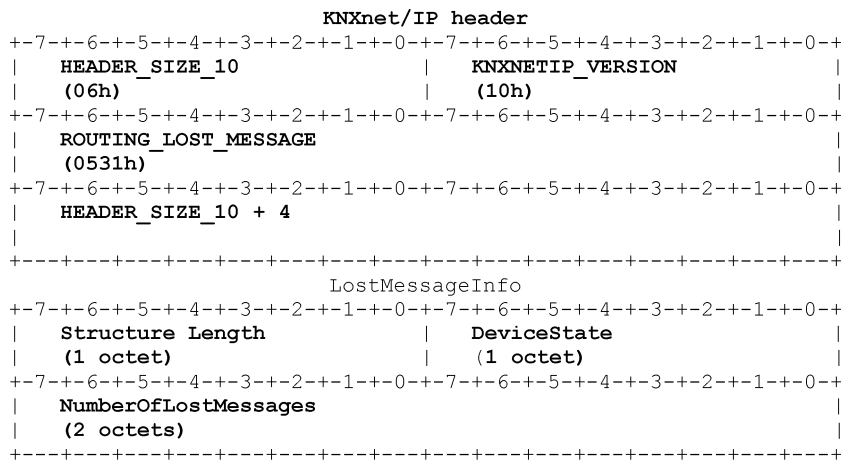


Figure 66 — ROUTING_LOST_MESSAGE frame binary format

5.5.5.4 ROUTING_BUSY

The ROUTING_BUSY frame binary format is shown in [Figure 67](#).

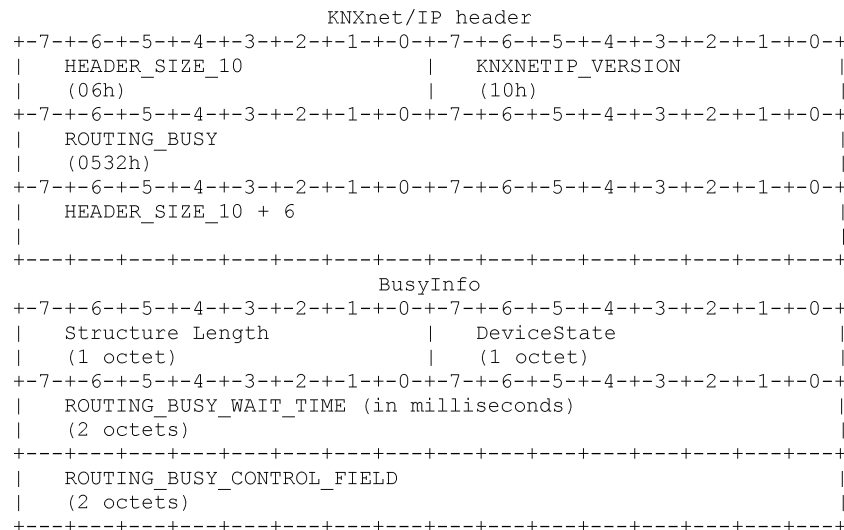


Figure 67 — ROUTING_BUSY frame binary format

5.5.5.5 ROUTING_SYSTEM_BROADCAST

The ROUTING_SYSTEM_BROADCAST frame binary format is shown in [Figure 68](#).

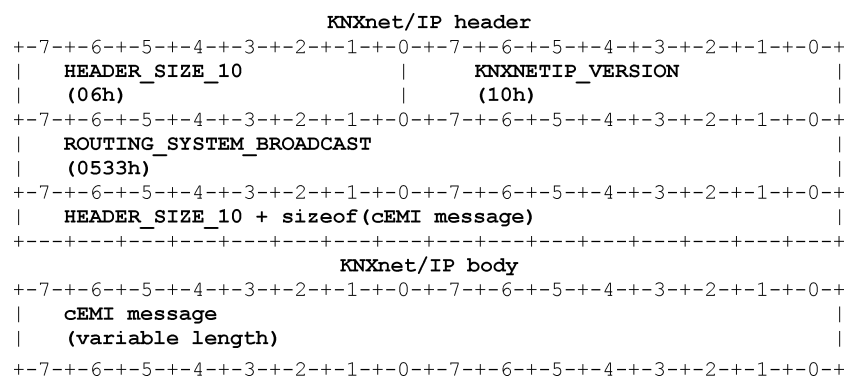


Figure 68 — ROUTING_SYSTEM_BROADCAST frame binary format

5.5.6 Minimum profiles

5.5.6.1 General

This subclause provides information on the minimum requirements, see [Table 36](#).

Table 36 — 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
ROUTING_SYSTEM_BROADCAST	Server → client	M

5.5.6.2 Test cases

5.5.6.2.1 General

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, see [Table 37](#) and [Table 38](#).

Table 37 — 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 38 — 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 Remote diagnosis and configuration

5.6.1 Use

“Remote diagnosis and configuration” of the KNXnet/IP specification 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 General

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 DIB 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 [5.3.2](#), 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 General

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 [5.1](#), Overview, for a list of valid KNXnet/IP common constants.

5.6.4.3 Common error codes

Refer to [5.1](#), 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, see [Figure 69](#).

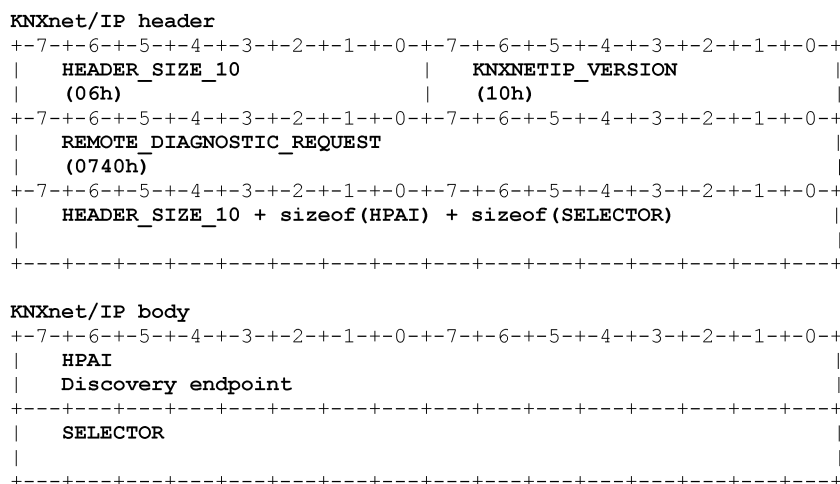


Figure 69 — REMOTE_DIAGNOSTIC_REQUEST frame binary format

5.6.4.4.2 REMOTE_DIAGNOSTIC_RESPONSE

The REMOTE_DIAGNOSTIC_RESPONSE datagram as shown in [Figure 70](#) 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.

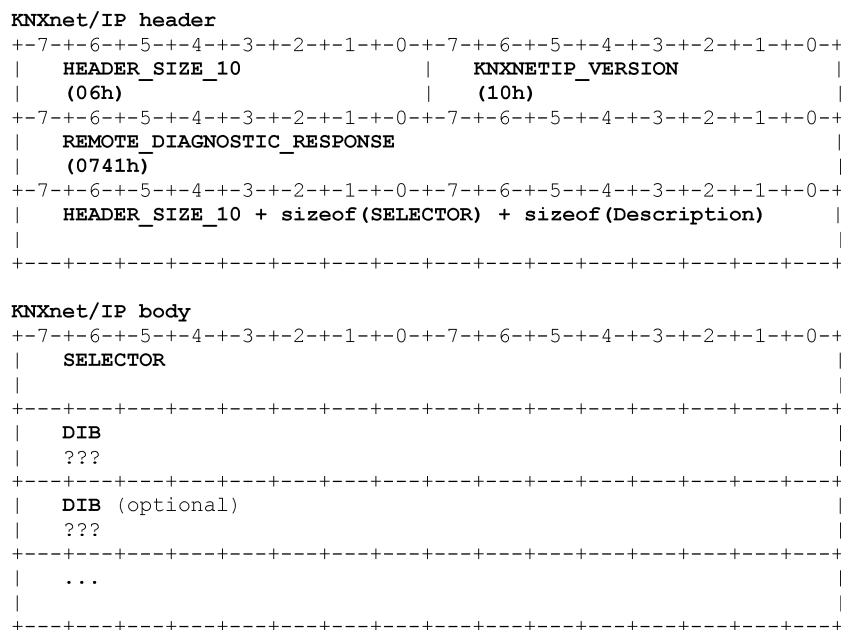


Figure 70 — REMOTE_DIAGNOSTIC_RESPONSE frame binary format

5.6.4.4.3 REMOTE_BASIC_CONFIGURATION_REQUEST

The REMOTE_BASIC_CONFIGURATION_REQUEST datagram as shown in [Figure 71](#) 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 DIB 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.

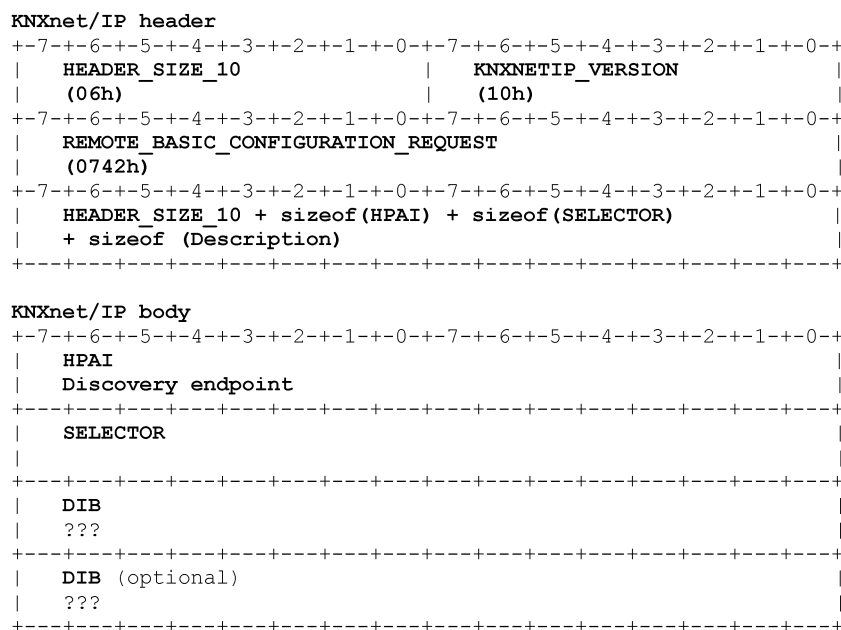


Figure 71 — REMOTE_BASIC_CONFIGURATION_REQUEST frame binary format

5.6.4.4.4 REMOTE_RESET_REQUEST

The REMOTE_RESET_REQUEST datagram as shown in [Figure 72](#) shall be transmitted via multicast or optionally via broadcast. A device that fits the selector shall accept the reset command without sending an acknowledgement.

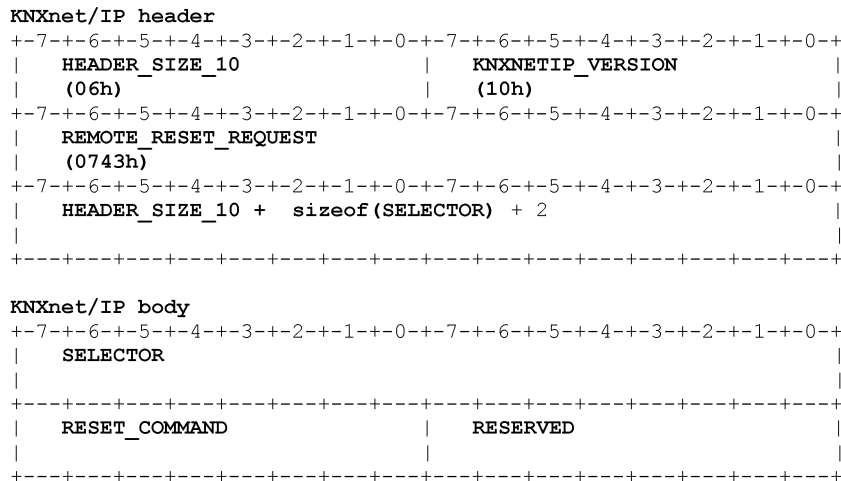


Figure 72 — REMOTE_RESET_REQUEST frame binary format

5.6.4.5 Description Information Block (DIB)

5.6.4.5.1 General

The Description Information Block (DIB) shall be a set of data that shall be accessed using the remote diagnosis and configuration services. While the core services for device discovery and device description 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 [5.2.7.5.4. Table 3](#) lists the description type codes. This list is extended with these description type codes used for remote diagnosis and configuration, see [Table 39](#):

Table 39 — 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 configuration DIB

The IP configuration DIB shall have the following structure, see [Figure 73](#).

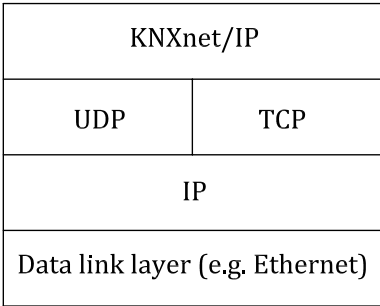


Figure 73 — IP configuration DIB

5.6.4.5.4 IP current configuration DIB

The IP current configuration DIB shall have the following structure, see [Figure 74](#).

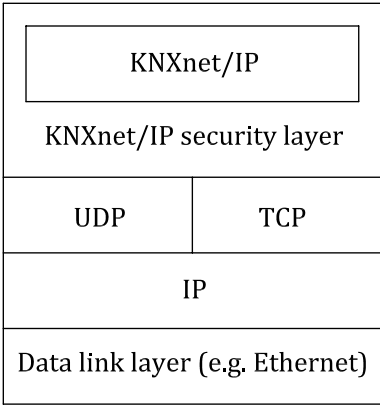


Figure 74 — IP current configuration DIB

5.6.4.5.5 KNX addresses DIB

The KNX address DIB shall have the following structure, see [Figure 75](#).

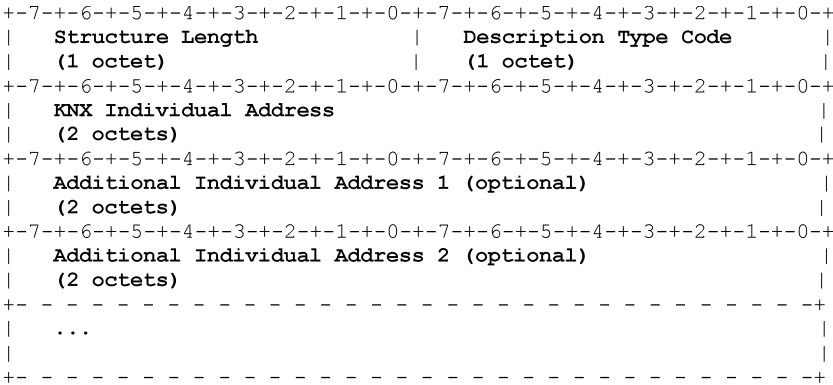


Figure 75 — KNX address DIB

5.6.4.6 Selector

5.6.4.6.1 General

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, see [Table 40](#).

Table 40 — Selector

Description type	Value	Description
PrgMode Selector	01h	Selection of devices in programming mode, see Figure 76
MAC Selector	02h	Selection of a device via MAC address, see Figure 77

5.6.4.6.2 Programming mode selector

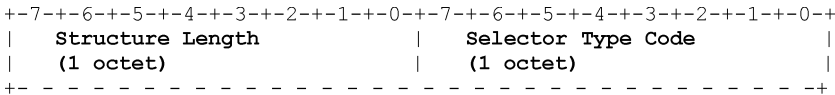


Figure 76 — Programming mode selector

5.6.4.6.3 MAC selector

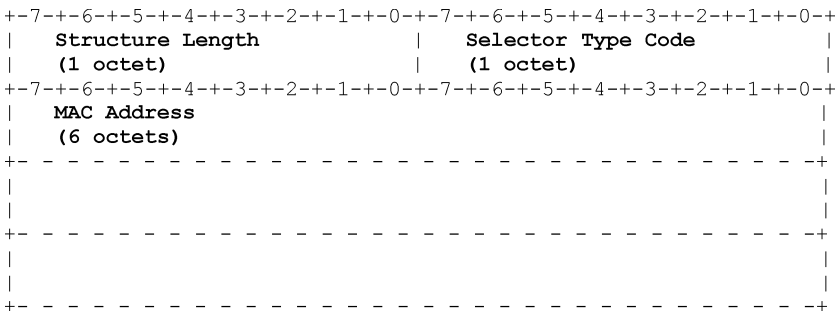


Figure 77 — MAC selector

5.6.4.7 Reset command

The reset command shall be picked from an enumeration.

The reset shall be executed immediately after receiving the command.

See [Table 41](#) for the reset command.

Table 41 — Reset command

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 General

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

5.6.5.2 Support matrix

See [Table 42](#) for the support matrix.

Table 42 — Support matrix

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

5.7 Secured communication

5.7.1 Use

The “Secured communication” of the KNXnet/IP specification defines the security wrapper for securing KNXnet/IP unicast and multicast traffic that puts an additional layer of security — transparent to all existing KNXnet/IP services — around the complete KNXnet/IP traffic.

5.7.1.1 General security goals

The most important goal of securing KNXnet/IP traffic is keeping outside attackers from gaining control over a KNX building automation system while connecting remotely over the Internet. A slightly different and also important scenario is an attacker gaining access over the local (W)LAN at the automation system's site. From the view of a KNXnet/IP device these two attacks are equivalent. Defending against these kinds of attack leads to two main security objectives: **data integrity** and **freshness**.

Ensuring data integrity means keeping an attacker from gaining control by injecting manipulated KNXnet/IP frames. Ensuring freshness means keeping an attacker from recording packets and playing them back at a later time without manipulating the contents. This is called a replay-attack: An attacker records KNXnet/IP traffic and at the same time observes what is happening in the automated building to relate recorded packets to actions. At a later time he can maliciously trigger an action by replaying the appropriate previously recorded packet

A different goal is keeping attackers from deriving knowledge about what is going on inside a building by looking at the KNXnet/IP traffic. This leads to the security objective of ensuring **confidentiality**. Confidentiality is normally achieved by encrypting network traffic to grant an attacker the lowest possible insight into the data actually transferred.

Because KNXnet/IP is implemented in small embedded devices, a security solution should require reasonable processing power and memory consumption.

5.7.1.2 Authentication

5.7.1.2.1 General

Aside from the three security objectives mentioned above, it is important that the communicating peers can trust each other's claimed identity. The security objective for this is called **mutual authentication**. Mutual authentication can also prevent an attack known as man-in-the-middle (MiM) attack. When

conducting a MiM attack, an attacker is logically located in between two communicating parties. He is able to intercept and change the traffic flowing in both directions. He masquerades as legal communication partner to both peers and sets up secured communication channels to both sides. He decrypts traffic coming from one side and re-encrypts it before sending it to the other side. So he is able to intercept and change the clear text communication. For KNXnet/IP security we have to differentiate between three kinds of authentication:

5.7.1.2.2 Server authentication

The goal of authenticating the server (KNXnet/IP device) to the client (tool software) for a KNXnet/IP unicast connection (e.g. tunnelling or device management) is to ensure that a client talking to a KNXnet/IP device (e.g. in order to configure the device) is really talking to the device it thinks it is talking to. Without this authentication an attacker could mimic or fake a specific KNXnet/IP device in order to gain knowledge about sensible configuration data sent to the device.

5.7.1.2.3 Client authentication

The goal of authenticating the client (tool software) to the server (KNXnet/IP device) for a KNXnet/IP unicast connection (e.g. tunnelling or device management) is to ensure that a client talking to a KNXnet/IP device (e.g. in order to access the KNX installation) is really authorized to do so. Today, everybody with knowledge of the IP address of a KNXnet/IP device has full access to the configuration of the device and possibly to the whole KNX installation it is connected to.

5.7.1.2.4 Group authentication

The goal of mutual authentication of the nodes involved in KNXnet/IP multicast communication (like KNXnet/IP routing) is to establish a trusted group of devices communicating together. Membership in this trusted group involves the possibility to authenticate to all other members in this group as legitimate member of this group.

5.7.1.2.5 Multicast communication

A special requirement of securing KNXnet/IP traffic is the support for **secure multicast communication** with more than one sender and more than one receiver where device individual telegram counters (point-to-point) cannot be used to ensure freshness.

5.7.1.2.6 Algorithms and key sizes

As a trade-off between desired security and available processing power as measured in terms of throughput and latency, the key sizes and algorithms of the KNXnet/IP secure standard represent the minimum requirement for supporting secure communication. While this document may be extended with additional key sizes and algorithms in the future, the current set will remain the default. This means that any KNXnet/IP secure device shall at least support CCM (AES-128 CTR and AES-128 CBC-MAC) and SHA-256 as a fall-back, if enabling of other, more advanced algorithms or key sizes is not supported.

5.7.2 Stack and communication

5.7.2.1 KNXnet/IP Security layer

5.7.2.1.1 General requirements and overview

KNXnet/IP is implemented on top of the UDP/TCP network layer. [Figure 78](#) shows the protocol architecture of the current unsecured KNXnet/IP protocol. With the introduction of KNXnet/IP security, new services are defined that continue to use this traditional setup, but may contain encrypted and/or authenticated data as payload. One special service is dedicated to transport unsecured KNXnet/

IP messages securely wrapped into a KNXnet/IP security message. [Figure 79](#) shows how this KNXnet/IP secure wrapper message is integrated into the existing protocol architecture.

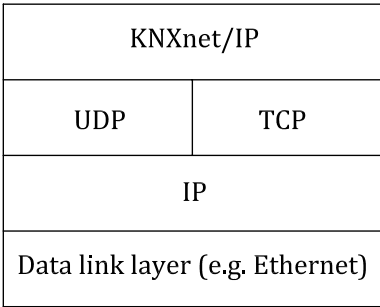


Figure 78 — KNXnet/IP protocol stack

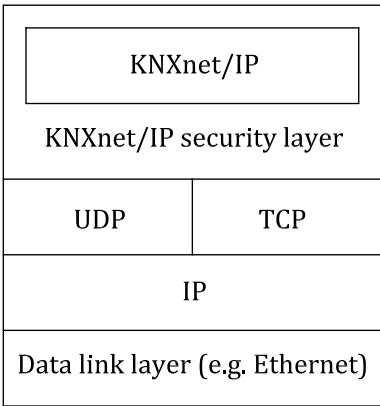


Figure 79 — KNXnet/IP secure wrapper

Locating the security layer in this case between the TCP/IP layer and the original KNXnet/IP layer has the advantage that the TCP/IP standard mechanisms like routing and multicast remain intact and also apply to the secured connection. No changes to existing frames are necessary.

The KNXnet/IP security frames are assigned a service type and a protocol version just like any other KNXnet/IP frame. The encapsulated frame is carrying its own protocol version and service type. This has the advantage that secure wrapper messages are able to carry any existing and future KNXnet/IP service type. Therefore, KNXnet/IP security can be expected to work seamlessly with existing and future extensions to KNXnet/IP.

5.7.2.1.2 Common frame format

a) KNXnet/IP header:

For compatibility reasons the KNXnet/IP security frames shall start with a standard KNXnet/IP header. Using the standard header, it is possible to run secure communication in parallel to insecure communication in the same installation and even on the same endpoint. A KNXnet/IP security-enabled device can distinguish security layer frames from regular frames by looking at the service type identifier located in the header. Non-security-enabled devices will just ignore the security layer frames because the KNXnet/IP security service type identifiers in the header are unknown to them. The format of the KNXnet/IP security header is shown in [Figure 80](#).

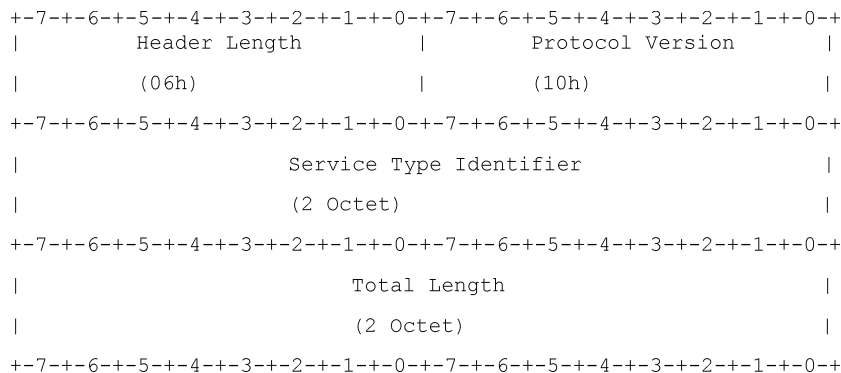


Figure 80 — Format of the KNXnet/IP security header

Header fields:

- Header length: Length of the header in Octets, fixed to 06h.
- Protocol version: KNXnet/IP security frames shall continue to use protocol version 1.0 as well as the frame structure itself because the KNXnet/IP header has not changed.
- Service type identifier: Identifies the type of the frame body following the header. The high octet shall always be 09h for KNXnet/IP security frames.
- Total length: The complete length of the KNXnet/IP security frame including the KNXnet/IP security header.

b) KNXnet/IP body:

1) Secure services:

Messages of the KNXnet/IP security service family have different needs regarding message authentication and confidentiality. Therefore, there is no pattern for a common frame format for all types of KNXnet/IP security frames.

2) Secure wrapper:

When KNXnet/IP frames are to be sent over a secured connection, each frame including the KNXnet/IP header shall be completely encapsulated as encrypted payload inside a SECURE_WRAPPER frame that adds some extra information needed to decrypt the frame and for ensuring data integrity and freshness.

[Figure 81](#) — KNXnet/IP secure wrapper frame detailing encrypted and authenticated parts — shows how a KNXnet/IP frame shall be embedded into a KNXnet/IP secure wrapper frame. It shall apply to secure multicast communication and to secure unicast runtime communication. The encapsulated KNXnet/IP frame in the KNXnet/IP secure wrapper body shall be prefixed with security information and followed by a message authentication code.

Ethernet	IP	TCP/ UDP	KNXnet/IP Secure Wrapper Header	KNXnet/IP Secure Wrapper Body		
				Security Information	Encapsulated KNXnet/IP Frame	MAC
Unencrypted			Authenticated	Authenticated	Authenticated & Encrypted	Encrypted
			Replay protected			

Figure 81 — KNXnet/IP secure wrapper frame detailing encrypted and authenticated parts

Which (plain) KNXnet/IP frames can be encapsulated in a KNXnet/IP secure wrapper frame depends on the KNXnet/IP service family to be secured. It is, for example not possible to embed a KNXnet/IP secure wrapper frame into another KNXnet/IP secure wrapper frame.

The network headers including the TCP/UDP headers shall be unencrypted to take advantage of standard mechanisms like routing, flow control, network address translation, etc. The network headers shall also not be included into the message authentication code to avoid problems with secured frames passing a network address translation.

3) Secure routing:

[Figure 82](#) — Encrypted part of a KNXnet/IP secure routing frame — shows a more detailed view on the encrypted part of a multicast routing frame. Because a routing device receives the complete routing traffic within its multicast group, it is important to be able to filter out unwanted frames as fast as possible. The decision whether a received multicast routing frame has to be processed any further depends on the KNX destination address buried inside the cEMI part of the routing frame.

As the figure shows, the KNX destination address can be found at octet positions 13 and 14 of the encrypted part of the KNXnet/IP secure frame. This means that for a routing decision it is sufficient to decrypt only the first 16 octet block of a received frame. So decrypting the whole frame is only necessary for frames that are already known to be of interest for the receiving device. Most other frames can be discarded after just decrypting the first 16 octets of encrypted data.

KNXnet/IP Routing Header (6 octets)	KNXnet/IP cEMI Message (6 octets before KNX destination address)	KNXnet/IP Secure Wrapper MAC
Authenticated & Encrypted		Encrypted

Figure 82 — Encrypted part of a KNXnet/IP secure routing frame

5.7.2.1.3 SECURE_WRAPPER

The binary format of the KNXnet/IP secure wrapper frame is shown in [Figure 83](#).

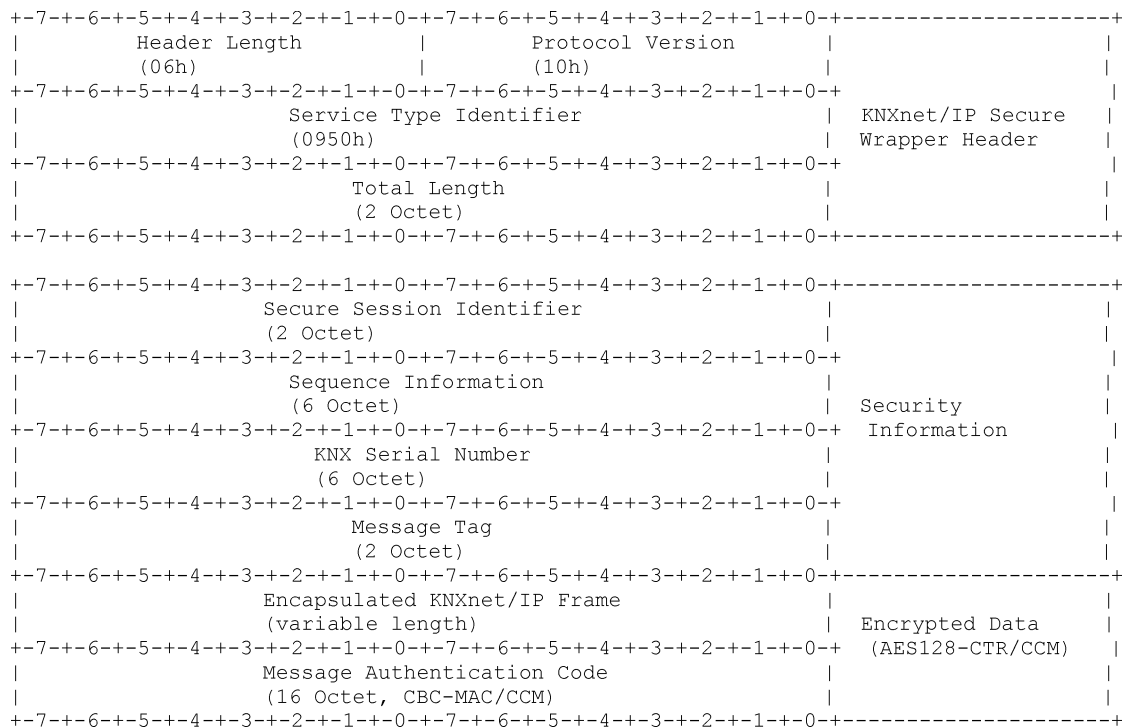


Figure 83 — Binary format of the KNXnet/IP secure wrapper frame

a) Fields:

— Secure Session Identifier:

ID of the secure session needed to decide which key to use. For multicast connections the fixed identifier 0000h shall be used. For unicast connections the ID was established during a previous successful secure session setup procedure.

— Sequence Information:

This field is used to defend against replay attacks. For unicast connections it is a monotonically increasing sequence number assigned by the sender; incremented by the sender after each frame sent. For multicast connections this is the device's current multicast timer value in millisecond resolution.

— KNX Serial Number:

The KNX serial number of the device sending the KNXnet/IP secure wrapper frame.

— Message Tag:

This field contains an arbitrary value to differentiate two KNXnet/IP secure wrapper multicast frames sent by one KNXnet/IP device within the same millisecond (with the same timer value). For unicast connections this field shall be ignored and shall be set to 0000h.

— Message Authentication Code:

The full 128 bit encrypted CBC-MAC from the CCM calculation.

b) Confidentiality and authentication:

SECURE_WRAPPER frames shall provide confidentiality using symmetric encryption and a message authentication code (MAC) shall be appended to every message for ensuring data integrity. For the symmetric encryption the AES algorithm with a key length of 128 bit shall be used in CTR mode

of operation. For calculating the MAC, the CBC-MAC method with AES 128 bit as cipher algorithm shall be used. This combination of AES128 in CTR mode and CBC-MAC is known as CCM.

The below requirements shall apply for the use of the SECURE_WRAPPER frame both in multicast as well as in unicast communication. They shall summarize the KNX specific definitions of the use of CCM in KNXnet/IP security:

- A:
The message authentication code in CCM protects the payload and so called associated data A. For KNXnet/IP secure wrapper frames A = KNXnet/IP secure wrapper header | Secure Session Identifier from the security information of the KNXnet/IP secure wrapper body. The length of the associated data is therefore fixed to a = 8 octets.
- P:
In CCM the payload P is defined as the amount of data that is being authenticated and encrypted. For KNXnet/IP secure wrapper frames P = encapsulated KNXnet/IP frame.
- B₀:
The composition of the first block B₀ for the CBC-MAC calculation in the CCM algorithm is specified in [Figure 84](#) — Format of B₀ for CCM in secure wrapper frames. These fields from the KNXnet/IP SECURE_WRAPPER frame are included here, so that they cannot be altered in the communication path between sender and receiver without being detected.

octet nr									
0	...	5	6	..	11	12	13	14	15
SeqInf			SerialNr			Tag		Q	

Figure 84 — Format of B₀ for CCM in secure wrapper frames

- SeqInf contains the unicast sequence number or multicast timer value from the security information of the KNXnet/IP secure wrapper body.
- SerialNr contains the KNX serial number value from the security information of the KNXnet/IP secure wrapper body.
- Tag contains the message tag value from the security information of the KNXnet/IP secure wrapper body.
- Q shall be the length of the payload P in octets, which is the length of the original, encapsulated KNXnet/IP frame.
- Ctr_i:
The format of the Block Counter Ctr_i is also KNX specific. Ctr_i shall be composed as specified in [Figure 85](#) — Format of Ctr_i for CCM in secure wrapper frames. For Ctr₀ the counter [i] shall be 00h. Each counter value [i] shall be calculated by incrementing the preceding counter value [i-1] by 1.

NOTE For security reasons (B₀ shall not be equal to Ctr₀) the maximum length of the payload P is limited to 65 279 octets (feffh in B₀). But as [j] in Ctr_j is only one octet, this practically limits the length of the payload to 255 × 16 (block size) = 4 080 octets.

octet nr									
0	...	5	6	..	11	12	13	14	15
SeqInf			SerialNr			Tag		ffh	[i]

Figure 85 — Format of Ctr_i for CCM in secure wrapper frames

- SeqInf contains the unicast sequence number or multicast timer value from the security information of the KNXnet/IP secure wrapper body.
- SerialNr contains the KNX serial number value from the security information of the KNXnet/IP secure wrapper body.
- Tag contains the message tag value from the security information of the KNXnet/IP secure wrapper body.

Due to using a nonce in multicast communication mainly consisting of the shared timer value, the device individual serial number and a message tag are added to Ctr_i to ensure that it is different on every encryption, even on different devices in the same millisecond (the message tag shall differentiate two messages sent by the same device in the same millisecond). This makes it harder for an attacker to gain information from traffic analysis.

c) Reception and decoding:

Upon reception of a KNXnet/IP secure wrapper frame the received data is evaluated and decoded in the following steps:

— Frame size:

KNXnet/IP secure wrapper frames are always built around an inner, encapsulated KNXnet/IP frame. As these inner, encapsulated frames have a minimum size of 6 octets (the size of the KNXnet/IP header), the minimum size of a KNXnet/IP secure wrapper frame is therefore 44 octets (6 octets for the secure header, 16 octets for the security information, 6 octets for the encapsulated KNXnet/IP header and 16 octets for the Message Authentication Code). Received KNXnet/IP secure wrapper frames smaller than 44 octets shall be discarded.

— Encryption key:

The secure session identifier indicates the encryption key used for securing the encrypted part of the KNXnet/IP secure wrapper body. If the secure session identifier refers to a non-existing session, the KNXnet/IP secure wrapper frame shall be discarded.

— Sequence number (unicast):

The sending device's sequence number in unicast secure wrapper frames shall only increase within a given session (not necessarily by one). Received secure wrapper frames shall be discarded if the sequence number is less than or equal to the last received number. As no further action is taken upon receiving outdated unicast secure wrapper frames, it is OK to check the sequence number before validating the message authentication code. For details on sequence number handling for unicast connections see [5.7.2.2.2](#).

— Message authentication code:

The received secure wrapper frame is decrypted with the associated key and a message authentication code Y_n over the received and decrypted plaintext data is created. This calculated MAC is then compared to the received and decrypted MAC TR from the secure wrapper frame. If they do not match, the KNXnet/IP secure wrapper frame shall be discarded.

— Sequence information (multicast):

For secure multicast communication even outdated, unexpected timer values are relevant (they trigger (re-)synchronization of the network). Therefore, the message authentication code has to be checked before the multicast timer value can be evaluated. The different possible reactions of the receiving communication partner depending on the received timer value are explained in detail in [5.7.2.2.3](#).

5.7.2.1.4 Access control

a) KNXnet/IP Services:

For each SECURE_WRAPPER received in a secure session, the KNXnet/IP server shall verify if the user authenticated for this session is allowed to request the wrapped KNXnet/IP service and has sufficient access rights on the accessed resources. The management user (= User ID 01h) shall always be granted access to all KNXnet/IP resources. For other users, access shall be granted as specified below.

b) KNXnet/IP core:

The discovery services in KNXnet/IP core do not require authorization. If the wrapped frame therefore is a SEARCH_REQUEST (extended) or DESCRIPTION_REQUEST access shall be granted independent of the authorized user.

If the wrapped frame is a CONNECT_REQUEST, the server shall first evaluate PID_SECURED_SERVICE_FAMILIES. If no security is enforced for this connection type, i.e. the corresponding service family entry in PID_SECURED_SERVICE_FAMILIES is 0, access shall be granted independent of the authorized user. Otherwise, the server shall evaluate the corresponding access control list, depending on the connection type, see [Table 43](#):

Table 43 — Access control list, depending on the connection type

Connection type	Access control list
DEVICE_MGMT_CONNECTION	Only the management user (01h) has access.
TUNNEL_CONNECTION	The management user (01h) always has access. For other users, the PID_TUNNELLING_USERS shall be evaluated; only if the user is found there, access shall be granted.

Unwrapped (plain) CONNECT_REQUESTs are only accepted if and only if the respective service family is set to non-secure in PID_SECURED_SERVICE_FAMILIES. The following [Table 44](#) gives an overview of all possible combinations of incoming CONNECT_REQUESTs depending on the state of PID_SECURED_SERVICE_FAMILIES:

Table 44 — Possible combinations of incoming CONNECT_REQUESTs depending on the state of PID_SECURED_SERVICE_FAMILIES

CONNECT_REQUEST of type ->	DEVICE_MGMT_CONNECTION		TUNNEL_CONNECTION	
	non-secure (0) for DEV_MGMT	secure (1) for DEV_MGMT	non-secure (0) for TUNNELLING	secure (1) for TUN- NELING
Unwrapped CONNECT_REQUEST	Access granted	Access denied	Access granted	Access denied
Wrapped CONNECT_REQUEST in dev-mgmt user (1) secure session	Access granted	Access granted	Access granted	Access granted

Table 44 (continued)

CONNECT_REQUEST of type ->	DEVICE_MGMT_CONNECTION		TUNNEL_CONNECTION	
PID_SECURED_SERVICE_FAMILIES is ->	non-secure (0) for DEV_MGMT	secure (1) for DEV_MGMT	non-secure (0) for TUNNELLING	secure (1) for TUN- NELING
Wrapped CONNECT_REQUEST in tunnelling user (>=2) secure session	Access granted	Access denied	Access granted	Access granted if allowed by PID_TUNNELLING_ USERS

c) KNXnet/IP device management:

Because KNXnet/IP device management v1 services (cEMI M_Prop) cannot be protected with KNX data security, access to a resource through cEMI M_Prop services will always be “anonymous” (without data security and role “unlisted”).

With KNXnet/IP device management v2 services (cEMI local transport layer) full management access to the device is possible even in device security mode if KNX data security is used together with the tool key. The lack of a source individual address in cEMI local transport layer prevents the use of other roles (R0 to R15).

d) KNXnet/IP tunnelling:

A detailed description how to verify access to a KNXnet/IP tunnelling interface in combination with different security scenarios and connection request options can be found in [5.4.3](#).

e) KNXnet/IP routing:

If the wrapped message is a “KNXnet/IP Routing Service Family” message (The KNXnet/IP service families are specified in [Annex A](#)) then it shall be ignored if it was not received on the routing endpoint or the session identifier is not 0 (multicast communication).

Secure routing frames (i.e. multicast secure wrapper frames containing a Routing.ind) shall be received if and only if KNXnet/IP routing service family is set to require secure communication [see [5.7.2.5.5](#) PID_SECURED_SERVICE_FAMILIES (PID = 94)].

Non-secure routing frames (i.e. plain multicast Routing.ind frames) shall be received if and only if the KNXnet/IP routing service family is set to not require secure communication (security version = 0).

If the KNXnet/IP routing service family is set to require secure communication (security version > 0), all sent Routing.ind frames shall be wrapped in a secure wrapper frame, regardless of their target group address, their target multicast address or any other aspect of the Routing.ind frame or the device configuration.

If the KNXnet/IP routing service family is set to not require secure communication, all sent Routing.ind frames shall not be wrapped in a secure wrapper frame and no timer notify frames may be sent or received.

f) Remote logging:

Access control resource to be defined when this connection type is specified.

g) Remote configuration:

It is not foreseen to support this service family if device security mode is enabled as these services would be able to set any configuration unauthorized via broadcast. If device security mode is enabled in the KNXnet/IP server device, it shall therefore ignore all incoming KNXnet/IP messages with service codes from the “KNXnet/IP remote configuration and diagnosis service family”.

h) Object server:

Access control resource to be defined when this connection type is specified.

5.7.2.2 Multicast communication

5.7.2.2.1 Key management

A single key is used for all devices in a KNXnet/IP routing multicast group. This backbone key is transferred once during device commissioning using a secure connection to the devices (via KNX data security possibly using secured KNXnet/IP device management). The key shall be transferred via the property `PID_BACKBONE_KEY` and shall have an unlimited lifetime.

All devices listening on the same multicast address using the same backbone key belong to one secured KNXnet/IP multicast group. Frames possibly received on other multicast addresses shall be discarded.

Everyone with knowledge of this secret backbone key is considered a legal member of the KNXnet/IP routing multicast group. If one device gets compromised, all keys used in this device shall be seen as compromised. Nevertheless, the backbone key can be changed for the remaining devices using secured connections to these devices to restore security for the KNXnet/IP routing multicast group communication.

5.7.2.2.2 Defending against replay attacks

a) Multicast sequence information:

For KNXnet/IP multicast communication it is not possible to use a simple sequence number like in the unicast case. In the multicast case there is an KNXnet/IP multicast group consisting of multiple nodes each of which can act as sender or as receiver. A sequence number would have to be maintained and persisted by every possible receiver for every sender. Without persistence, a receiver would be vulnerable by a replay attack from power-up until every sender has received a valid frame.

Apart from the resources required on the receiving side, at least one kind of vulnerability would remain: If an attacker can capture a valid frame and make sure that the frame doesn't reach the legal receiver, he can replay the frame later at any time. To get around this issue, sequence information is needed which changes even if no communication is taking place.

Therefore, a free running timer that is synchronized between all KNXnet/IP multicast group members shall be used for providing sequence information. The range of the timer values is large enough for the timers to be assumed to never overflow. Synchronization of timer values between devices are only allowed in the forward direction: The sender of a packet shall include its current timer value as time stamp in the sent packet. The receiver of a packet shall replace its own timer value with the received time stamp if the received time stamp is having a greater value than the internal timer.

The timer value of an incoming frame shall be compared to the current local timer value. The frame shall be discarded if the received frame is older than the time span given in `PID_MULTICAST_LATENCY_TOLERANCE` (typically a few seconds). This means that frames with slightly past timer values shall be accepted to account for network latency.

The use of KNX data security on application layer for critical applications includes an additional device specific counter that further minimizes the risk of replay attacks and reduces the need to tweak the latency tolerance on KNXnet/IP.

b) Device timer implementation:

The timer and the time stamp shall have a width of 48 bit. With timer ticks every millisecond, an overflow of the timer would theoretically occur after 9 thousand years. This value should

be sufficient to well exceed the lifetime of an average building. The timer shall be reset to zero whenever the backbone key is changed.

The device timer shall not be decreased in any case. This requirement especially includes the power-down case. See [5.7.2.2.3](#) for implementation details.

c) System clock accuracy:

The proposed timer-based replay attack protection depends on the device clocks being synchronized with an allowable tolerance of a few milliseconds. Given standard quartz micro controller oscillator circuits, a clock accuracy better than 50 ppm can easily be achieved. This implies that the clock deviation between any two devices is at most 100 ppm.

5.7.2.2.3 Timer synchronizing

a) Timer sync service:

1) General:

Whenever a KNXnet/IP service family using multicast communication is set to require secure communication, the multicast timer between devices in a secure KNXnet/IP multicast group shall be synchronized. For this purpose, devices shall send `TIMER_NOTIFY` messages to:

- synchronize the device timer after power up: If the device doesn't need to send data immediately after power up, a `TIMER_NOTIFY` frame shall be scheduled to be sent at a random time between 0 s (`minDelayInitialNotify`) and 10 s (`maxDelayInitialNotify`) if no recent-enough `TIMER_NOTIFY` or `SECURE_WRAPPER` frame has been received within this period of time. This shall prevent a flood of `TIMER_NOTIFY` frames sent by the same type of device after a power cycle. If the device needs to send data within this first period of time it can then shorten its delay time and immediately issue a `TIMER_NOTIFY` frame.
- periodically resynchronize the timers: A `TIMER_NOTIFY` shall be sent by any device if no recent-enough `TIMER_NOTIFY` or `SECURE_WRAPPER` frame has been received for about 10 s.
- resynchronize an outdated timer: A `TIMER_NOTIFY` shall be sent by any device after a random time of usually up to around 3 s (`maxDelayTimeFollowerUpdateNotify`) after the device received an outdated `TIMER_NOTIFY` or `SECURE_WRAPPER` frame.

Any device receiving a valid `TIMER_NOTIFY` or `SECURE_WRAPPER` frame shall compare the received timer value with its local timer value and update its local timer value if the received timer value is greater than the local timer value.

2) Reaction to outdated timer values:

Any device receiving a valid outdated `TIMER_NOTIFY` or `SECURE_WRAPPER` frame shall schedule sending a `TIMER_NOTIFY` frame after a random delay of usually up to around 3 s (`maxDelayTimeFollowerUpdateNotify`).

If a `TIMER_NOTIFY` or `SECURE_WRAPPER` frame with a timer value greater than the sync latency tolerance (fraction of the overall latency tolerance) behind the local timer value as time stamp field is received before the delay elapses, the schedule shall be cancelled.

If no recent-enough `TIMER_NOTIFY` or `SECURE_WRAPPER` frame is received before the delay elapses, the device shall send out its local timer value in a `TIMER_NOTIFY` frame and shall assume to be a time keeper for its secure KNXnet/IP multicast group.

3) Time keeper:

Devices that claim to be a time keeper for their secure KNXnet/IP multicast group continue scheduling `TIMER_NOTIFY` frames on outdated received `TIMER_NOTIFY` or `SECURE_WRAPPER` frames in a shorter and earlier time window than the other devices. Time keeper

devices use a reduced maxDelayUpdateNotify time and a reduced MaxDelayPeriodicNotify time to be able to more quickly correct outdated timers or tell devices that have newly joined the secure KNXnet/IP multicast group the correct current time. The exact definition of the time windows for time keeper and/or time follower devices is dependent on the configured network latency.

Whenever time keeper devices receive a valid TIMER_NOTIFY frame with the received timer value greater than their local timer value, they no longer claim to be a time keeper for their secure KNXnet/IP multicast group.

b) Timer sync state machine:

1) Goal:

The state machine in each device tries to keep all mc_timers in all devices synchronized with as little communication overhead as possible. It is completely transparent for all KNX/IP communication except for discarding outdated SECURE_WRAPPER frames. The underlying goal is to detect and discard replayed SECURE_WRAPPER frames.

2) Parameters: The parameters are given in [Table 45](#).

Table 45 — Parameters

Parameter	Unit	Value	Description
latencyTolerance	ms	configured default = 2 000 ms	SECURE_WRAPPER frames which are older than this tolerance when comparing their timer value to the receiving device's mc_timer are discarded. This parameter should therefore reflect the absolute worst-case latency of the network connecting all devices of the secure KNXnet/IP multicast group. This value can be configured through PID_MULTICAST_LATENCY_TOLERANCE.
syncLatencyTolerance	ms	configured default = 200 ms	Common case latency. 99,9 % of all observed network latencies should be within the syncLatencyTolerance. The default value is 10 % of the latencyTolerance. This value can be configured through PID_SYNC_LATENCY_FRACTION.
minDelayInitialNotify	s	0 s	Minimum delay for sending an initial TIMER_NOTIFY after power-up.
maxDelayInitialNotify	s	10 s	Maximum delay for sending an initial TIMER_NOTIFY after power-up.
minDelayTimeKeeperPeriodicNotify	s	10 s	Minimum delay of time keeper devices for sending a TIMER_NOTIFY after having received the last acceptable TIMER_NOTIFY or SECURE_WRAPPER.
maxDelayTimeKeeperPeriodicNotify	ms	calculated	Maximum delay of time keeper devices for sending a TIMER_NOTIFY after having received the last acceptable TIMER_NOTIFY or SECURE_WRAPPER.
minDelayTimeFollowerPeriodicNotify	ms	calculated	Minimum delay of all devices that are not time keepers for sending a TIMER_NOTIFY after having received the last acceptable TIMER_NOTIFY or SECURE_WRAPPER.
maxDelayTimeFollowerPeriodicNotify	ms	calculated	Maximum delay of all devices that are not time keepers for sending a TIMER_NOTIFY after having received the last acceptable TIMER_NOTIFY or SECURE_WRAPPER.

Table 45 (continued)

Parameter	Unit	Value	Description
minDelayTimeKeeperUpdateNotify	ms	100 ms	Minimum delay of time keeper devices for sending a TIMER_NOTIFY as an update after having received an outdated TIMER_NOTIFY or SECURE_WRAPPER.
maxDelayTimeKeeperUpdateNotify	ms	calculated	Maximum delay of time keeper devices for sending a TIMER_NOTIFY as an update after having received an outdated TIMER_NOTIFY or SECURE_WRAPPER.
minDelayTimeFollowerUpdateNotify	ms	calculated	Minimum delay of all devices that are not time keepers for sending a TIMER_NOTIFY as an update after having received an outdated TIMER_NOTIFY or SECURE_WRAPPER.
maxDelayTimeFollowerUpdateNotify	ms	calculated	Maximum delay of all devices that are not time keepers for sending a TIMER_NOTIFY as an update after having received an outdated TIMER_NOTIFY or SECURE_WRAPPER.

The calculated parameters are mainly depending on the network's latencyTolerance and syncLatencyTolerance, see [Table 46](#):

Table 46 — Calculated parameters, mainly depending on the network's latencyTolerance and syncLatencyTolerance

Parameter	Expression
maxDelayTimeKeeperPeriodicNotify	minDelayTimeKeeperPeriodicNotify + 3 × syncLatencyTolerance
minDelayTimeFollowerPeriodicNotify	maxDelayTimeKeeperPeriodicNotify + 1 × syncLatencyTolerance
maxDelayTimeFollowerPeriodicNotify	minDelayTimeFollowerPeriodicNotify + 10 × syncLatencyTolerance
maxDelayTimeKeeperUpdateNotify	minDelayTimeKeeperUpdateNotify + 1 × syncLatencyTolerance
minDelayTimeFollowerUpdateNotify	maxDelayTimeKeeperUpdateNotify + 1 × syncLatencyTolerance
maxDelayTimeFollowerUpdateNotify	minDelayTimeFollowerUpdateNotify + 10 × syncLatencyTolerance

3) Local variables: The local variables are described in [Table 47](#).

Table 47 — Description of local variables

Name	Description
mc_timer	48-bit wide timer, one tick per millisecond real-time. The value of this timer is used in SECURE_WRAPPER and TIMER_NOTIFY frames as sequence information. The state machine in each device tries to synchronize all mc_timers in all devices. mc_timer is always counting up and never stops. The mc_timer only ever increases except when IP_backbone_key is updated which implicitly resets mc_timer to 0.
notify_timer	Timer counting down, resolution ideally 10 ms or finer. Measured in seconds and fractions of seconds. When the notify_timer reaches 0,0 s it stops, event E10 is generated and a TIMER_NOTIFY frame is sent.
backbone_key	Symmetric 128 bit AES key used for multicast communication. Setting this to a new value implicitly sets the mc_timer back to 0 and generates event E11. Setting this to the same value as before generated event E12.
minDelayPeriodicNotify	Minimum delay for sending a TIMER_NOTIFY after having received the last acceptable TIMER_NOTIFY or SECURE_WRAPPER. This variable has different values depending on if the device is a time keeper or not.
maxDelayPeriodicNotify	Maximum delay for sending a TIMER_NOTIFY after having received the last acceptable TIMER_NOTIFY or SECURE_WRAPPER. This variable has different values depending on if the device is a time keeper or not.

Table 47 (continued)

Name	Description
minDelayUpdateNotify	Current minimum delay for sending a TIMER_NOTIFY as an update after having received an outdated TIMER_NOTIFY or SECURE_WRAPPER. This variable has different values depending on if the device is a time keeper or not.
maxDelayUpdateNotify	Current maximum delay for sending a TIMER_NOTIFY as an update after having received an outdated TIMER_NOTIFY or SECURE_WRAPPER. This variable has different values depending on if the device is a time keeper or not.

4) States: The states are described in [Table 48](#).

Table 48 — Description of state labels

State label	State description
SCHED_PERIODIC	A TIMER_NOTIFY is scheduled to be sent once notify_timer (delay between minDelayPeriodicNotify and maxDelayPeriodicNotify) expires (E10).
SCHED_UPDATE	A TIMER_NOTIFY is scheduled to be sent once notify_timer (delay between minDelayUpdateNotify and maxDelayUpdateNotify) expires (E10).

5) Events: The events are described in [Table 49](#).

Table 49 — Event description

Event label	Event description
E01	Received TIMER_NOTIFY frame (received_timer_value > mc_timer)
E02	Received TIMER_NOTIFY frame (received_timer_value <= mc_timer) and (received_timer_value > mc_timer - syncLatencyTolerance)
E03	Received TIMER_NOTIFY frame (received_timer_value <= mc_timer - syncLatencyTolerance) and (received_timer_value > mc_timer - latencyTolerance)
E04	Received TIMER_NOTIFY frame (received_timer_value <= mc_timer - latencyTolerance)
E05	Received multicast SECURE_WRAPPER frame (received_timer_value > mc_timer)
E06	Received multicast SECURE_WRAPPER frame (received_timer_value <= mc_timer) and (received_timer_value > mc_timer - syncLatencyTolerance)
E07	Received multicast SECURE_WRAPPER frame (received_timer_value <= mc_timer - syncLatencyTolerance) and (received_timer_value > mc_timer - latencyTolerance)
E08	Received multicast SECURE_WRAPPER frame (received_timer_value <= mc_timer - latencyTolerance)
E09	Transmitted multicast SECURE_WRAPPER frame
E10	The notify_timer expired (reached 0,0 s).
E11	Device joins new domain. PID_BACKBONE_KEY and/or PID_MULTICAST_ADDRESS is set to a new and different value (via property access or DomainAddressSerialNumber_Write).

The events E01 to E04 for the reception of TIMER_NOTIFY frames are symmetrical to the events E05 to E08 for the reception of SECURE_WRAPPER frames. These sets of events refer to different points in time when the frames are received. [Figure 86](#) — State machine events for different received timer values, illustrates these reception times on a time bar.

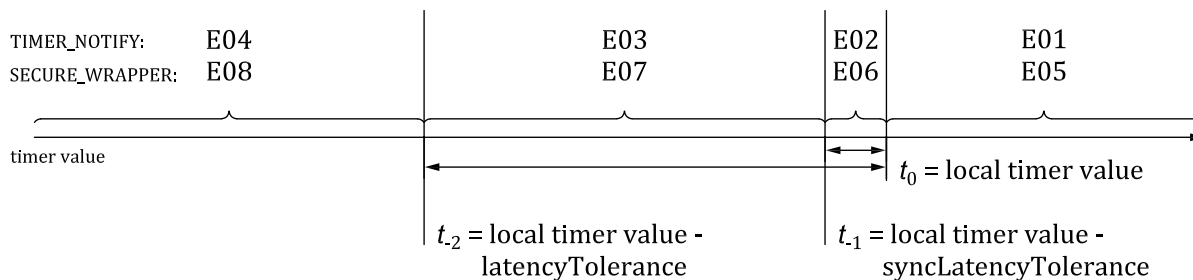


Figure 86 — State machine events for different received timer values

6) Actions: The action labels are described in [Table 50](#).

Table 50 — Description of action labels

Action label	Action description
A0	Do nothing.
A1	mc_timer = received_timer_value
A2	Accept SECURE_WRAPPER frame and pass to upper layers as usual.
A3	Reschedule notify_timer = random (minDelayPeriodicNotify, maxDelayPeriodicNotify).
A4	Remember received serial number and tag of outdated frame. notify_timer = random (minDelayUpdateNotify, maxDelayUpdateNotify)
A5	Send TIMER_NOTIFY with own mc_timer as sequence information, own serial number and random tag.
A6	Send TIMER_NOTIFY with own mc_timer as sequence information, but serial number and tag as remembered in A4.
A7	Restart the multicast timer synchronization, see 5.7.2.2.3 .
A8	Becoming time keeper: minDelayUpdateNotify = minDelayTimeKeeperUpdateNotify maxDelayUpdateNotify = maxDelayTimeKeeperUpdateNotify minDelayPeriodicNotify = minDelayTimeKeeperPeriodicNotify maxDelayPeriodicNotify = maxDelayTimeKeeperPeriodicNotify
A9	Becoming time follower: minDelayUpdateNotify = minDelayTimeFollowerUpdateNotify maxDelayUpdateNotify = maxDelayTimeFollowerUpdateNotify minDelayPeriodicNotify = minDelayTimeFollowerPeriodicNotify maxDelayPeriodicNotify = maxDelayTimeFollowerPeriodicNotify

NOTE 1 random(start, end) returns a uniformly distributed random number between start and end inclusive.

7) Transition table: A transition table is shown in [Table 51](#).

Table 51 — Transition table

Event	State	
	SCHED_PERIODIC	SCHED_UPDATE
E01	SCHED_PERIODIC A1 + A9 + A3	>> SCHED_PERIODIC A1 + A9 + A3
E02	SCHED_PERIODIC A9 + A3	>> SCHED_PERIODIC A9 + A3
E03	SCHED_PERIODIC A0	SCHED_UPDATE A0

Table 51 (continued)

Event	State	
	SCHED_PERIODIC	SCHED_UPDATE
E04	>> SCHED_UPDATE A4	SCHED_UPDATE A0
E05	SCHED_PERIODIC A1 + A2 + A3	SCHED_UPDATE A1 + A2
E06	SCHED_PERIODIC A2 + A3	SCHED_UPDATE A2
E07	SCHED_PERIODIC A2	SCHED_UPDATE A2
E08	>> SCHED_UPDATE A4	SCHED_UPDATE A0
E09	SCHED_PERIODIC A3	SCHED_UPDATE A0
E10	SCHED_PERIODIC A5 + A8 + A3	>> SCHED_PERIODIC A6 + A8 + A3
E11	SCHED_PERIODIC A7	>> SCHED_PERIODIC A7

NOTE 2 Although in E05 a time keeper might receive a newer timer value than its own, it does not get time follower, because this decision will only be taken based upon timer notify frames.

NOTE 3 Sending or receiving of secure wrapper frames (E09 or E05) does not cancel scheduled timer notify updates (although the recent timer value was already communicated to the multicast group) to allow direct (trusted) answers for outdated devices.

8) Starting the multicast timer synchronization:

The multicast timer synchronization shall only be active if at least one service family using multicast communication is set to require secure communication. The start of the multicast timer synchronization can therefore happen directly after device power-up or whenever the configuration of secured service families changes.

In these cases the start state of the timer sync state machine is:

- state = SCHED_PERIODIC.
- In case of device power-up, the notify_timer = random (0, maxDelayInitialNotify) to prevent a flood of notifications of devices booting up at the same time (e.g. after a power cycle), otherwise notify_timer = 0.
- mc_timer = Value read from persistent storage (only if valid) + worst case time offset (if applicable). If no such value is found (important: this is only acceptable for the first time ever that a device has started the multicast timer synchronization with the current backbone key) mc_timer = 0.

After the start of the multicast timer synchronization and before receiving the first authentic timer value a device is vulnerable to replay attacks. An attacker could replay any SECURE_WRAPPER and TIMER_NOTIFY traffic captured between the device's old, unsynchronized mc_timer and the current time. The device will not be able to tell that its own mc_timer and the received timer values are out of date and cannot detect the received traffic as replayed.

To work around this, a device can acquire an authentic timer value after the start of the multicast timer synchronization by using the following procedure:

- Set a flag mc_timer_authentic = false.

- Do not process the encapsulated data of SECURE_WRAPPER frames until mc_timer_authentic becomes true.
- Send or schedule a TIMER_NOTIFY. Remember the used tag.
- Wait for $\text{maxDelayTimeFollowerUpdateNotify} + 2 \times \text{latencyTolerance}$ after the first TIMER_NOTIFY or SECURE_WRAPPER (sent or received).
- If a TIMER_NOTIFY is received that repeats the own serial number and remembered tag value, then set mc_timer_authentic to true and stop waiting any further.
- Take the most recent timer value of all received TIMER_NOTIFYs and SECURE_WRAPPERs as the initial mc_timer and set mc_timer_authentic to true.

Initially an arbitrary number of replayed (and thus false) `TIMER_NOTIFY` or `SECURE_WRAPPER` frames may be received. But other sane devices will also have received these frames and the false (old) response timer values. At least one of these other sane devices will answer with an authentic timer value. The latest time until this authentic timer value could be received is `latencyTolerance + maxDelayTimeFollowerUpdateNotify + latencyTolerance`.

The disadvantage of this procedure is that a device may neither receive nor send any SECURE_WRAPPER frames before this authentic mc_timer acquisition is complete. In a typical Ethernet LAN this may take up to $10\text{ s} + 2\text{ s} + 3\text{ s} + 2\text{ s} = 17\text{ s}$.

If this kind of delay is not acceptable for a certain application a device may also send and receive SECURE_WRAPPER frames before the authenticity of the multicast group timer is established. In this case the application software shall be robust against early replay attacks.

5.7.2.2.4 TIMER_NOTIFY — Usage

This frame shall be sent during secure KNXnet/IP multicast group communication to keep the multicast group member's timer values synchronized. The frame shall be sent to the KNXnet/IP routing endpoint (port 3671 on the configured routing multicast address).

a) Binary format: The Binary format of the KNXnet/IP secure timer notify frame is shown in [Figure 87](#).

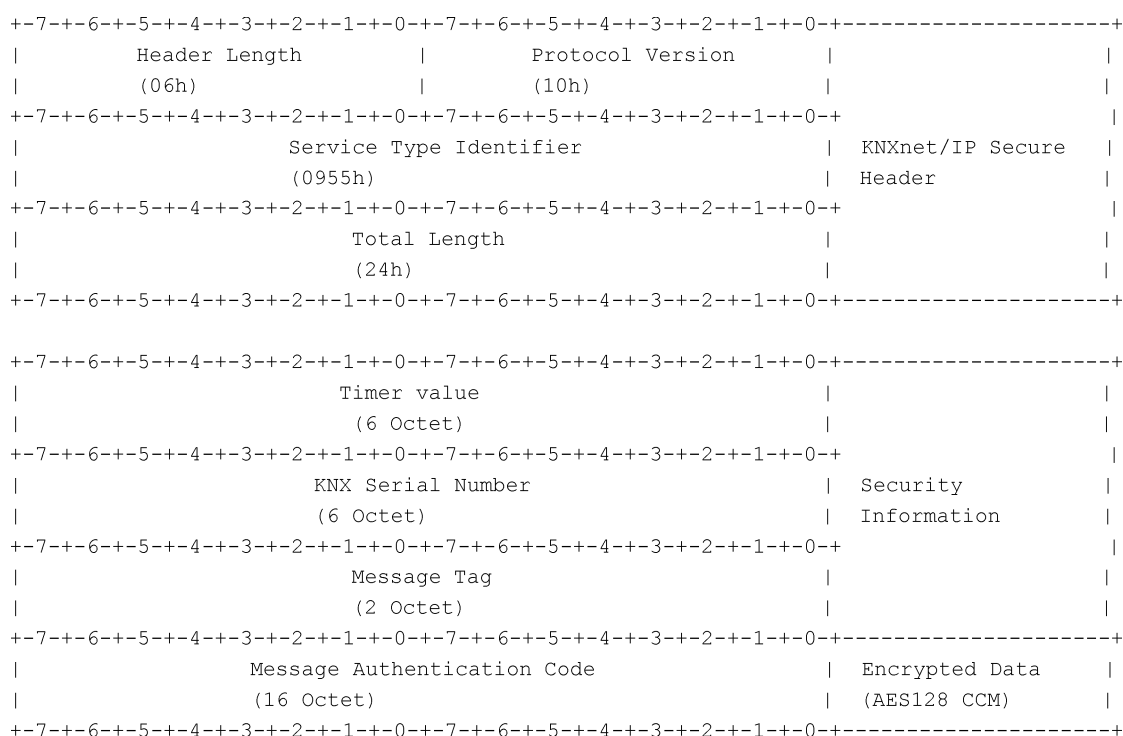


Figure 87 — Binary format of the KNXnet/IP secure timer notify frame

b) Fields:

— Timer value:

The current multicast timer value of the device sending the timer notification.

— KNX Serial Number:

In case of a periodic or initial notify this is the KNX serial number of the device sending the timer notification. In case of an update notify this is the serial number of the outdated frame triggering the update.

— Message Tag:

In case of a periodic or initial notify this field shall contain a random value. In case of an update notify this is the tag value of the outdated frame triggering the update. The tag field of periodic TIMER_NOTIFY frames shall be random to prevent the unencrypted data to only increase by one for subsequent timer values. This shall protect the backbone key used to generate the MAC against analytic attacks and serve as a challenge for update notify frames so they can be immediately trusted as valid timer source for the device having generated the original tag value.

— Message Authentication Code:

The full 128 bit CCM-MAC. The CCM algorithm ensures that the MAC itself is also encrypted.

c) Authentication:

TIMER_NOTIFY frames shall provide authentication using a message authentication code (MAC) that shall be appended to the TIMER_NOTIFY message for ensuring data integrity. For calculating this MAC, the CCM algorithm with AES128 as cipher shall be used. As AES128 key for the CCM algorithm the backbone key shall be used.

The KNX specific definitions of the CCM parameters for TIMER_NOTIFY messages shall be:

— A:

For KNXnet/IP timer notify frames A = KNXnet/IP secure wrapper header only. The length of the associated data is fixed to a = 6 octets.

— P:

As TIMER_NOTIFY frames only use authentication, the CCM payload P is empty. This means that the length of the payload Q for the generation of B_0 is fixed to 0.

— B_0 :

The composition of the first block B_0 for the CBC-MAC calculation in the CCM algorithm is specified in [Figure 88](#) — Format of B_0 for the CCM CBC-MAC in timer notify frames. These fields from the KNXnet/IP TIMER_NOTIFY frame are included here, so that they cannot be altered in the communication path between sender and receiver without being detected.

octet nr									
0	...	5	6	..	11	12	13	14	15
TimerValue			SerialNr			Tag		Q = 0000h	

Figure 88 — Format of B_0 for the CCM CBC-MAC in timer notify frames

- TimerValue contains the multicast timer value from the security information of the KNXnet/IP timer notify body.

- SerialNr contains the KNX serial number value from the security information of the KNXnet/IP timer notify body.
- Tag contains the message tag value from the security information of the KNXnet/IP timer notify body.
- Q shall be the length of the payload P in octets, which is fixed to 0 for KNXnet/IP timer notify frames.

— Ctr₀:

The format of the Block Counter Ctr₀ is also KNX specific (for timer notify frames only the calculation of one AES128 block for the encryption of the MAC is necessary). Ctr₀ shall be composed as specified in [Figure 89](#) — Format of Ctr₀ for the CCM MAC encryption in timer notify frames.

octet nr									
0	...	5	6	..	11	12	13	14	15
TimerValue			SerialNr			Tag		ffh	00h

Figure 89 — Format of Ctr₀ for the CCM MAC encryption in timer notify frames

- TimerValue contains the multicast timer value from the security information of the KNXnet/IP timer notify body.
- SerialNr contains the KNX serial number value from the security information of the KNXnet/IP timer notify body.
- Tag contains the message tag value from the security information of the KNXnet/IP timer notify body.

d) Reception and decoding:

Upon reception of a KNXnet/IP timer notify frame the received data is evaluated and decoded in the following steps:

— Frame size:

KNXnet/IP timer notify frames have a fixed length of 36 octets (6 octets for the secure header, 14 octets for the security information and 16 octets for the MAC). Received KNXnet/IP timer notify frames larger or smaller than 36 octets shall be discarded.

— Message Authentication Code (MAC):

The received timer notify frame is decrypted with the backbone key and the message authentication code Y1 over the received security information is created. This calculated MAC is then compared to the received and decrypted MAC TR from the timer notify frame. If they do not match, the KNXnet/IP timer notify frame shall be discarded.

— Sequence information:

For secure multicast communication even outdated, unexpected timer values are relevant (they trigger (re-)synchronization of the network). Therefore, the message authentication code has to be checked before the multicast timer value can be evaluated. The different possible reactions of the receiving communication partner depending on the received timer value are explained in detail in [5.7.2.2.3](#) Timer synchronizing.

5.7.2.3 Unicast connections

5.7.2.3.1 Key management

a) Session key:

When a KNXnet/IP client wants to securely communicate with a KNXnet/IP server using connectionless or connection oriented unicast KNXnet/IP services it shall use a temporary key for every communication session. The lifetime of this session key (how long this key remains valid) is called a KNXnet/IP secure session.

The session key shall be negotiated between the two communicating parties using the Elliptic-Curve Diffie-Hellman (ECDH) key agreement algorithm (Rescorla, Diffie-Hellman Key Agreement Method, 2009). As ECDH domain parameters the Curve25519 shall be used. During session setup both parties exchange the public part of a randomly generated ECDH key pair. Knowledge of each other's public key in combination with the own private key allows both peers to calculate the same random number from which the session key is derived.

Both communication parties shall generate a new random public/private key pair on the Curve25519 specification as ECDH parameters for every new session using a cryptographically safe source of entropy. It is not allowed to generate a random key pair once (e.g. per power-on or per device) and re-use that for more than one session.

b) Calculating the session key:

The session key is calculated as follows (for both the KNXnet/IP secure client and KNXnet/IP secure server):

- 1) `sharedSecret_in_little_endian = Curve25519(myPrivateKey, peersPublicKey)`
- 2) `hash_in_big_endian = SHA256(sharedSecret_in_little_endian)`
- 3) `sessionKey = get_first_16_bytes(hash_in_big_endian)`

5.7.2.3.2 Mutual authentication

a) Authentication by known secrets:

During session setup, a KNXnet/IP secure device shall authenticate itself to the connecting client by proving knowledge of a secret based on the device authentication code. The client shall authenticate itself to the device by proving knowledge of a secret based on a user password. Both secrets have previously been stored in the device.

b) Securing against dictionary attacks:

Password based authentication mechanisms always bear the risk of a dictionary attack being conducted. In a dictionary attack an attacker tries passwords from a dictionary in order of decreasing probability. For real-world passwords regularly a surprisingly small number of attempts are necessary to find out the correct password.

Dictionary attacks can be separated into online attacks and offline attacks. In an online attack, the attacker repeatedly connects to the system and on each connection tries a different password. In an offline attack, the attacker needs to be able to gain enough information by passively listening to the traffic of a connection to be able to check the passwords offline on his own (maybe parallelized) hardware.

Offline attacks are for example possible when the password itself or a simple (unsalted) hash of the password is transmitted as plain text. Password based challenge-response authentication schemes may also be prone to offline dictionary attacks if the attacker is able to get the plain text challenge and the plain text response.

KNXnet/IP secure unicast sessions cannot easily be compromised by an offline dictionary attack because the authentication data that an attacker would need to generate for each password to be tested is calculated using a time-consuming cryptographic algorithm reducing the amounts of possible tests in a reasonable time by several orders magnitude.

Conducting an online dictionary attack is always possible. However, compared to an offline attack the number of passwords that can be tried online in a given time span is rather small. Since an online attack takes considerable time and requires the attacker being actively connected to the installation for that time, his risk of being discovered is much higher compared to an offline attack.

To further decrease the feasibility of online or offline dictionary attacks, the tool software shall inform the installers about the strength of the selected passwords (increasing the probability that passwords have a minimal, acceptable strength).

5.7.2.3.3 Defending against replay attacks

To prevent replay attacks, every packet sent via a secure connection shall bear a number as sequence information. Every communication partner shall have its own independent sequence number counter. The sequence number shall have a width of 48 bit. Assuming one million (10^6) packets being sent per second for each direction, an overflow of the sequence number would theoretically occur after 9 years. Furthermore, the sequence numbers start at zero for every new connection further minimizing the risk of an overflow.

The sequence number shall be initialized to zero on connection setup and incremented by one after each packet sent. For every connection both peers shall store the sequence number of the last received frame. An incoming frame shall be discarded if the sequence number is less than or equal to the stored number.

An incoming frame shall be accepted by the receiver if the sequence number is greater than the sequence number of the previously successfully received frame on the same connection. There is no requirement that the sequence number of two consecutive frames differ by exactly one. Because of this, the unicast sequence information mechanism is inherently tolerant to frames getting lost.

This also means that for frames received in swapped order, those frames coming in too late will get lost. It is depending on the stability of the upper protocol layers that make use of the KNXnet/IP secure unicast session if this causes any problems. Because of this KNXnet/IP secure unicast sessions shall use TCP and shall not use UDP.

5.7.2.3.4 Securing the initial configuration

In a freshly installed device all passwords may be empty. Therefore, during initial configuration of the device (until the password has been set by the installer), no reliable authentication of the client to the device is possible. However, all other security objectives mentioned above are met. The device will authenticate itself to the client by using an initial device authentication code (value equal to FDSK). As knowledge of this code by the client requires at least some form of physical access to the device, some sort of client authentication is still guaranteed (not if device authentication is optional). Confidentiality, freshness and data integrity are assured using the standard measures described above.

5.7.2.3.5 Secure sessions

a) Associating session keys to secure sessions:

During session setup, the server shall assign a secure session identifier to the secure session being established (not to be confused with the KNXnet/IP connection identifier). The server shall ensure that this secure session identifier is unique to all of its currently open secure sessions. When transferring data over the session, the secure session identifier shall be included in every frame sent.

Selecting the right session key and sequence numbers for sending and receiving KNXnet/IP secure frames shall for the KNXnet/IP server be based on the secure session identifier. A KNXnet/IP client may additionally have to consider the IP information (e.g. TCP connection) or KNX serial number of the KNXnet/IP server if multiple concurrent sessions to different servers are maintained.

Secure sessions may not be nested, i.e. a secure session may not be wrapped in an existing secure session.

b) Session state machine:

- Goal: The goal of this state machine is to clearly specify the behaviour of KNXnet/IP servers during the phase of negotiating the temporary session keys as well as defining the lifetime of these keys (the lifetime of a session). The state machine refers to a single session on the KNXnet/IP secure server. Multiple sessions may exist in parallel.
- Parameters: Parameters of the session state machine is shown in [Table 52](#).

Table 52 — Parameters of the session state machine

Parameter	Unit	Value	Description
timeoutAuthentication	s	10 s	Maximum time the authentication process for newly created secure sessions may last until the unauthenticated session will be dropped.
timeoutSession	s	60 s	Maximum time an authenticated session may remain unused (without any communication over this session) until the session will be dropped.

- Local variables: Local variables of the session state machine are shown in [Table 53](#).

Table 53 — Local variables of the session state machine

Name	Description
session_timer	Timer counting down, resolution ideally 10 ms or finer. When the session_timer reaches 0,0 s it stops, the session will be aborted.

- States: The states of the session state machine are shown in [Table 54](#).

Table 54 — States of the session state machine

State label	State description
IDLE	The session is not open.
UNAUTHENTICATED	The session has been created, but is not yet authenticated.
AUTHENTICATED	The session has been successfully authenticated.

- Events: The events of the session state machine are shown in [Table 55](#).

Table 55 — Events of the session state machine

Event label	Event description
E00	Received SESSION_REQUEST
E01	Received valid SECURE_WRAPPER containing valid SESSION_AUTHENTICATE
E02	Received valid SECURE_WRAPPER containing invalid SESSION_AUTHENTICATE
E03	Received valid SECURE_WRAPPER containing SESSION_STATUS with status field STATUS_CLOSE
E04	Received valid SECURE_WRAPPER containing SESSION_STATUS with status field STATUS_KEEPALIVE
E05	Received valid SECURE_WRAPPER containing any frame except SESSION_AUTHENTICATE and SESSION_STATUS
E06	The session_timer expired (reaches 0,0 s)

In addition to the validation rules described at service level the terms “valid” and “invalid” also mean in this context:

- For SECURE_WRAPPER frames, valid means that the wrapper can be decrypted and MAC-validated with the session key associated with the session. Invalid SECURE_WRAPERS shall be ignored.
- For SESSION_AUTHENTICATE frames, valid means that the MAC can successfully be validated with the password hash of the indicated user ID as key.
- Actions: The actions of the session state machine are shown in [Table 56](#).

Table 56 — Actions of the session state machine

Action label	Action description
A0	i) Allocate session. ii) Send SESSION_RESPONSE. iii) session_timer = timeoutAuthentication
A1	i) Send SESSION_STATUS with status field STATUS_AUTHENTICATION_SUCCESS. ii) session_timer = timeoutSession
A2	i) Send SESSION_STATUS with status field STATUS_AUTHENTICATION_FAILED. ii) Deallocate session.
A3	i) Close all contained secure connections (if there are any), without explicit notification of the connection close to the client. ii) Send SESSION_STATUS with status field STATUS_CLOSE. iii) Deallocate session.
A4	session_timer = timeoutSession
A5	i) Send SESSION_STATUS with status field STATUS_TIMEOUT. ii) Close all contained secure connections (if there are any). iii) Deallocate session.
A6	i) Send SESSION_STATUS with status field STATUS_UNAUTHENTICATED. ii) Close all contained secure connections (if there are any). iii) Deallocate session.

- Transition table: The transition table of the session state machine is shown in [Table 57](#).

Table 57 — Transition table of the session state machine

Event	State		
	IDLE	UNAUTHENTICATED	AUTHENTICATED
E00	>> UNAUTHENTICATED A0	Cannot happen	Cannot happen
E01	Cannot happen	>> AUTHENTICATED A1	>> IDLE A2
E02	Cannot happen	>> IDLE A2	>> IDLE A2
E03	Cannot happen	>> IDLE A3	>> IDLE A3
E04	Cannot happen	>> IDLE A6	AUTHENTICATED A4

Table 57 (continued)

Event	State		
	IDLE	UNAUTHENTICATED	AUTHENTICATED
E05	Cannot happen	>> IDLE A6	AUTHENTICATED A4
E06	Cannot happen	>> IDLE A5	>> IDLE A5

5.7.2.3.6 SESSION_REQUEST

a) Usage:

This frame shall be sent by the KNXnet/IP secure client to the control endpoint of the KNXnet/IP secure server to initiate the secure session setup handshake for a new secure communication channel. The maximum time a KNXnet/IP secure client shall wait for a response of the KNXnet/IP secure server shall be 10 s.

b) Binary format:

The binary format of the KNXnet/IP secure session request frame is shown in [Figure 90](#).

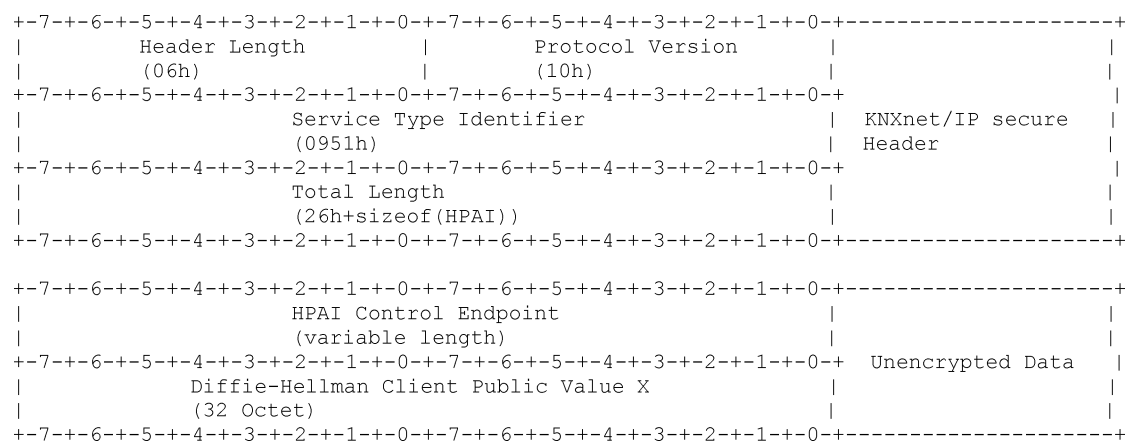


Figure 90 — Binary format of the KNXnet/IP secure session request frame

Fields:

— HPAI Control Endpoint:

This field shall contain the return address information of the KNXnet/IP secure client's control endpoint so the KNXnet/IP secure server knows where to direct possible secure channel responses and/or status information.

— Diffie-Hellman Client Public Value X:

ECDH public value X as calculated by the KNXnet/IP secure client from its chosen secret on the Curve25519 for ECDH key agreement.

c) Security:

The SESSION_REQUEST frame is sent unencrypted to the KNXnet/IP secure server and is therefore prone to replay attacks. Only if the server generates a new (cryptographically safe) random public/private ECDH parameter Y according to the Curve25519 specification for every new session it can be assured that replayed SESSION_REQUEST frames result in different session keys and a replay of SECURE_WRAPPER frames is not possible.

— Message Authentication Code:

The full 128 bit CCM-MAC. The CCM algorithm ensures that the MAC itself is also encrypted.

c) Authenticating the device to the client:

To defend against man-in-the-middle-attacks, the KNXnet/IP secure server shall authenticate itself against the KNXnet/IP secure client with its device authentication code as a shared secret. This authentication is integrated into the Elliptic-Curve Diffie-Hellman procedure when the KNXnet/IP secure server sends its ECDH public value to the client.

With the knowledge of the device authentication code as shared secret the client is able to optionally validate the MAC of the received session response frame and this way can trust the authenticity of both the message as well as the responding KNXnet/IP secure server.

d) Authentication:

SESSION_RESPONSE frames shall provide the KNXnet/IP secure server authentication using a message authentication code (MAC) that shall be appended to the session response frame for ensuring data integrity. For calculating this MAC, the CCM algorithm with AES128 as cipher shall be used. As AES128 key for the CCM algorithm the device authentication code shall be used.

The KNX specific definitions of the CCM parameters for SESSION_RESPONSE messages shall be:

— A:

For KNXnet/IP session response frames $A = \text{KNXnet/IP Secure Header} \mid \text{Secure Session Identifier} \mid (\text{Diffie-Hellman Client Public Value } X \wedge \text{Diffie-Hellman Server Public Value } Y)$. The length of the associated data is fixed to $a = 40$ octets.

— P:

As KNXnet/IP session response frames only use authentication, the CCM payload P is empty. This means that the length of the payload Q for the generation of B_0 is fixed to 0.

— B_0 :

The composition of the first block B_0 for the CBC-MAC calculation in the CCM algorithm is specified in [Figure 92](#) — Format of B_0 for the CCM CBC-MAC in session response frames.

octet nr				
0	...	13	14	15
0			Q = 0000h	

Figure 92 — Format of B_0 for the CCM CBC-MAC in session response frames

Q shall be the length of the payload P in octets, which is fixed to 0 for KNXnet/IP session response frames.

— Ctr_0 :

The format of the Block Counter Ctr_0 is also KNX specific (for session response frames only the calculation of one AES128 block for the encryption of the MAC is necessary). Ctr_0 shall be composed as specified in [Figure 93](#) — Format of Ctr_0 for the CCM MAC encryption in session response frames.

octet nr				
0	...	13	14	15
0			ffh	00h

Figure 93 — Format of Ctr_0 for the CCM MAC encryption in session response frames

e) Reception and decoding:

Upon reception of a KNXnet/IP session response frame the received data is evaluated and decoded in the following steps:

— Frame size:

KNXnet/IP session response frames have a fixed length of 56 octets (6 octets for the secure header, 2 octets for the session identifier, 32 octets for Diffie-Hellman Server Public Value Y and 16 octets for the Message Authentication Code). Received KNXnet/IP session response frames larger or smaller than 56 octets shall be discarded.

— Message Authentication Code:

Optionally, the received session response frame is decrypted with the device authentication code and the message authentication code Y_n over the received frame and the session's Diffie-Hellman public values X and Y is created. This calculated MAC is then compared to the received and decrypted MAC TR from the session response frame. If they match, the KNXnet/IP session response frame can be trusted.

f) Exception handling:

If the KNXnet/IP server device cannot process the secure session request (either because there are no more free secure session resources or the device is too busy to calculate another ECDH parameter set) then the SESSION_REQUEST shall be discarded and no SESSION_RESPONSE shall be sent.

5.7.2.3.8 SESSION_AUTHENTICATE

a) Usage:

This frame shall be sent by the KNXnet/IP secure client to the control endpoint of the KNXnet/IP secure server after the Diffie-Hellman handshake to authenticate the user against the server device. The maximum time a KNXnet/IP secure client shall wait for an authentication status response of the KNXnet/IP secure server shall be 10 s.

b) Binary format: The binary format of the KNXnet/IP session authenticate frame is shown in [Figure 94](#).

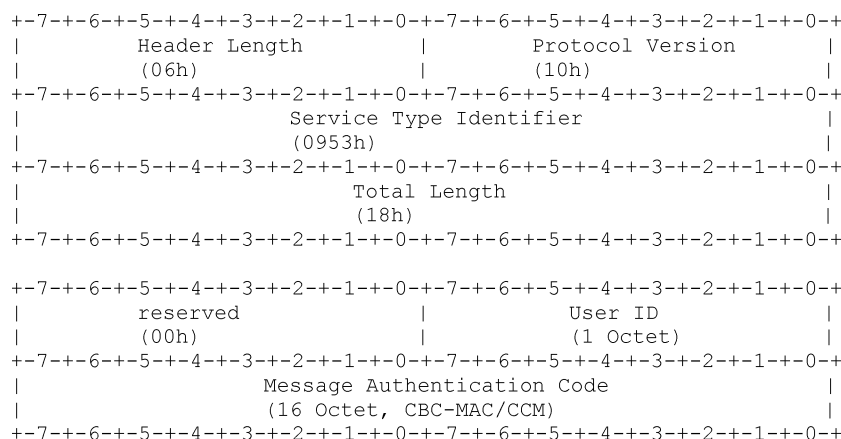


Figure 94 — Binary format of the KNXnet/IP session authenticate frame

Fields:

— Reserved:

This field shall be set to 00h by the client. If the server receives a non-zero value the frame shall be discarded.

— User ID:

This field shall indicate the user requested by the client for authentication of the secure session. The server shall use the user ID as index into the passwords array [see 5.7.2.5.4 PID_PASSWORD_HASHES (PID = 93) for details].

— 00h: Reserved, shall not be used

— 01h: Management level access

— 02h–7Fh: User level access

— 80h–FFh: Reserved, shall not be used

— The access level (management level access or user level access with possibly any device dependent role) shall also determine the set of services accepted by the server after authentication.

— Message Authentication Code:

The full 128 bit CCM-MAC. The CCM algorithm ensures that the MAC itself is also encrypted.

To protect the communication from possible attacks, the authentication request frame shall be encrypted and authenticated. Therefore, this frame shall only be sent wrapped in a SECURE_WRAPPER frame over the intended secure session.

c) Authenticating the client to the device:

For authenticating the client (typically the tool software or visualizations) and user of the secure session to the KNXnet/IP secure device, each device shall be secured by two types of passwords for two different levels of access.

— Management-level access

On the management level all services including device management shall be supported. To change KNXnet/IP secure parameters, KNX data security role “ETS” (usage of the tool key) shall be required.

— User-level access

On the user level, all services except device management may be supported. This means, on the user level everything shall be allowed that would also be possible using a physical connection to the KNX bus.

The passwords are specific to devices and therefore may be individual for each device. For practical reasons, a user may choose to opt for the same password set in all devices of his installation project. The client decides which of the two passwords to use for authentication. After successful authentication, the degree of access granted by the device to the client shall depend on the password used during authentication.

If authentication fails, the client has to restart with a new ECDH handshake; it is not allowed to send more than one authentication request with the same ECDH session key. This dramatically delays every attempt to online attack the passwords.

d) Authentication

SESSION_AUTHENTICATE frames shall provide their authentication information using a message authentication code (MAC) that shall be appended to the session authenticate frame for ensuring data integrity. For calculating this MAC, the CCM algorithm with AES128 as cipher shall be used. As AES128 key for the CCM algorithm the password hash of the user ID shall be used.

The KNX specific definitions of the CCM parameters for SESSION_AUTHENTICATE messages shall be:

— A:

For KNXnet/IP session authentication frames $A = \text{KNXnet/IP Header} \mid 00\text{h} \mid \text{User ID} \mid (\text{Diffie-Hellman Client Public Value } X \wedge \text{Diffie-Hellman server Public Value } Y)$. The length of the associated data is fixed to $a = 40$ octets.

— P:

As KNXnet/IP session authentication frames only use authentication, the CCM payload P is empty. This means that the length of the payload Q for the generation of B_0 is fixed to 0.

— B_0

The composition of the first block B_0 for the CBC-MAC calculation in the CCM algorithm is specified in [Figure 95](#) — Format of B_0 for the CCM CBC-MAC in session authenticate frames.

octet nr				
0	...	13	14	15
0			Q = 0000h	

Figure 95 — Format of B_0 for the CCM CBC-MAC in session authenticate frames

Q shall be the length of the payload P in octets, which is fixed to 0 for KNXnet/IP session response frames.

— Ctr_0 :

The format of the Block Counter Ctr_0 is also KNX specific (for session authenticate frames only the calculation of one AES128 block for the encryption of the MAC is necessary). Ctr_0 shall be composed as specified in [Figure 96](#) — Format of Ctr_0 for the CCM MAC encryption in session authenticate frames.

octet nr				
0	...	13	14	15
0			ffh	00h

Figure 96 — Format of Ctr₀ for the CCM MAC encryption in session authenticate frames

e) Reception and decoding:

Upon reception of a KNXnet/IP session authenticate frame the received data is evaluated and decoded in the following steps:

— Frame size:

KNXnet/IP session authenticate frames have a fixed length of 24 octets (6 octets for the secure header, 1 octet reserved, 1 octet for the user ID and 16 octets for the Message Authentication Code). Received KNXnet/IP session authenticate frames larger or smaller than 24 octets shall be discarded.

— Message Authentication Code:

The MAC of the received session authenticate frame is decrypted with the password hash for the user ID and the message authentication code Y_n over the received frame and the session's Diffie-Hellman public values X and Y is created. This calculated MAC is then compared to the received and decrypted MAC TR from the session authenticate frame. If they match [they can only match if the password (hash) used as key matches], the secure session is authenticated for the user ID (see [5.7.2.3.5 b\)](#) for details).

5.7.2.3.9 SESSION_STATUS

a) Usage:

This frame may be sent by the KNXnet/IP secure server to the KNXnet/IP secure client or by the KNXnet/IP secure client to the KNXnet/IP secure server in any stage of the secure session handshake to indicate an error condition or status information.

b) Binary format:

The binary format of the KNXnet/IP session status frame is shown in [Figure 97](#).

+-7--6--5--4--3--2--1--0--+-7--6--5--4--3--2--1--0--+															
Header Length								Protocol Version							
(06h)								(10h)							
+-7--6--5--4--3--2--1--0--+-7--6--5--4--3--2--1--0--+															
Service Type Identifier															
(0954h)															
+-7--6--5--4--3--2--1--0--+-7--6--5--4--3--2--1--0--+															
Total Length															
(08h)															
+-7--6--5--4--3--2--1--0--+-7--6--5--4--3--2--1--0--+															
+-7--6--5--4--3--2--1--0--+-7--6--5--4--3--2--1--0--+															
Status								Reserved							
(1 Octet)								(1 Octet)							
+-7--6--5--4--3--2--1--0--+-7--6--5--4--3--2--1--0--+															

Figure 97 — Binary format of the KNXnet/IP session status frame

Fields:

— Status

This field shall indicate an error condition or status information for the secure session. Usage by the KNXnet/IP secure server:

- responding to the authentication request of the client:
 - 00h → STATUS_AUTHENTICATION_SUCCESS
 - 01h → STATUS_AUTHENTICATION_FAILED
- signalling a timeout for or ending of the secure session:
 - 03h → STATUS_TIMEOUT
 - 05h → STATUS_CLOSE
- indicating an access control issue:
 - 02h → STATUS_UNAUTHENTICATED

Usage by the KNXnet/IP secure client:

- preventing the already authenticated session from expiring due to timeout:
 - 04h → STATUS_KEEPLIVE
- closing the secure session explicitly:
 - 05h → STATUS_CLOSE

To protect the communication from possible attacks, the session status frame shall be encrypted and authenticated. Therefore, this frame shall only be sent wrapped in a SECURE_WRAPPER frame over the intended secure session.

Whenever one of the above-mentioned events triggers the transmission of a session status frame, the communication partner shall expect to receive this status within a maximum wait time of 10 s.

Any of the status codes for ending the secure session (STATUS_CLOSE, STATUS_UNAUTHENTICATION_FAILED, STATUS_UNAUTHENTICATED and STATUS_TIMEOUT) shall be unconfirmed. This means that directly after sending any of these session status frames, the TCP connection considered unused may also be closed (start of the termination procedure for TCP connections by sending TCP FIN command) without having to wait for a reaction (confirmation) of the communication partner first.

5.7.2.4 Implementation requirements

5.7.2.4.1 Random number generation

The secure implementation of the KNXnet/IP secure protocol — like any other cryptographic protocol — needs a source of cryptographically secure random numbers at runtime. Because in a typical embedded KNX device there is hardly any entropy to derive random numbers from, a software implemented cryptographically secure pseudo random number generator (CSPRNG) should be used for calculating random numbers.

For computing a different sequence of random numbers in every device, the CSPRNG shall be seeded with a device-specific seed. This seed may be a random number individually programmed into every device during manufacturing. Alternatively, the device's individual private key might be used to seed the CSPRNG. To meet the security requirements of the cryptographic algorithms used, a random seed which should be at least 128 bits long is needed.

For computing a different sequence of random numbers in the same device after every power cycle, an additional seed shall be used that changes with every power cycle. A simple persistent counter which is increased for every power cycle may be used to implement this. Alternatively, the persistent secure KNXnet/IP multicast timer detailed in the following subclause may be used.

Immediately after power-up, the RAM is likely to contain random values which might be used as a source of entropy.

During runtime any available entropy should be fed into the CSPRNG. A suitable source of entropy for this is the system timer value read on the occurrence of certain external events like receiving a frame from the KNX TP line or on the Ethernet connection.

5.7.2.4.2 Persisting the multicast timer

The KNXnet/IP multicast timer shall be monotonically increasing at least every millisecond. It shall under no circumstances be decremented because this would weaken the resistance against replay attacks. To achieve this, the multicast timer shall be persisted during power-off conditions. Even better it should be increased during power-off conditions using an RTC.

Depending on the actual hardware implementation there may be enough time after detecting a low power condition to persist the multicast timer to non-volatile (flash) memory. If the hardware implementation cannot guarantee the above condition, the multicast timer can be written to flash by software in regular intervals of maximum one hour. After power-up when reading the stored multicast timer, the worst case is assumed that the value has been stored at the very beginning of an interval while the power was lost at the end of an interval. To ensure monotonic incrementing under this assumption, the timer difference corresponding to one interval is added to the timer value just read.

The maximum persistence interval shall be measured in mc_timer-time not real-time, because the mc_timer may increase by an arbitrary amount during one hour of real time. The fundamental requirement is that mc_timer may not run backwards even on power loss. From this it follows that when a device recently persisted an mc_timer value T and uses a persistence interval of D (all in milliseconds), then it may only send out timer notify frames and secure wrapper frames with timer values in the range T to T+D. Before it sends out a timer value > T+D it shall persist the current mc_timer value. Similarly, when receiving a frame with timer value > T+D it shall persist this new timer value immediately because otherwise it cannot guarantee not reusing timer values after a power loss. All this happens regardless of the real time elapsed since the last persisting operation.

5.7.2.5 Resource definition KNXnet/IP parameter object

5.7.2.5.1 General

The device management of a security enabled KNXnet/IP device shall expose an additional set of properties in the “KNXnet/IP Parameter Object” (Object Type = 11), see [Table 58](#). It shall hold the security related properties as listed in [Table 10](#). (This table only gives an overview. For the mandatory and optional properties, please refer to [Annex C](#)).

The security conditions for accessing these properties are specified in the detailed property descriptions in [5.7.2.5.2](#) and further. In general, the security related properties of the KNXnet/IP parameter object shall only be accessible via KNX data security. They shall not be accessible via any other standard- or non-standard means.

It is therefore recommended that the memory where the properties are stored should be encrypted as well, so that it cannot be interpreted by any other interface to this memory, bypassing the KNX S-AL, such as a JTAG-interface.

Table 58 — Security related properties in the KNXnet/IP parameter object

Property name	Property identifier	Property datatype
Secure Backbone Key	91 = PID_BACKBONE_KEY	PDT_GENERIC_16
Device Authentication Code	92 = PID_DEVICE_AUTHENTICATION_CODE	PDT_GENERIC_16
Password Hashes	93 = PID_PASSWORD_HASHES	PDT_GENERIC_16[]
Secured Service Families	94 = PID_SECURED_SERVICE_FAMILIES	PDT_FUNCTION
Multicast Latency Tolerance	95 = PID_MULTICAST_LATENCY_TOLERANCE	PDT_UNSIGNED_INT
Multicast Sync Latency Fraction	96 = PID_SYNC_LATENCY_FRACTION	PDT_SCALING
Tunnelling Users	97 = PID_TUNNELLING_USERS	PDT_GENERIC_02[]

5.7.2.5.2 PID_BACKBONE_KEY (PID = 91)

a) Abstract resource definition:

This parameter shall contain the key used for encryption, decryption and MAC calculation in secure multicast communication.

b) Security:

The backbone key shall not be readable and shall only be writable using application layer services secured with the tool key (authentication and confidentiality).

c) Usage by the management server:

In ex-factory state and after every factory reset the backbone key shall be zero.

The backbone key is part of the KNX/IP domain configuration. It can be changed via property data access or via A_DomainAddressSerialNumber_Write-PDU. The behaviour of the management server upon changing the backbone key shall not differ depending on the way this resource was accessed.

Changing the backbone key affects the generation of KNXnet/IP Routing.ind frames if the KNXnet/IP Routing Service Family is configured to be secure. KNXnet/IP secure wrapper frames already generated with the previous backbone key may still be sent from outgoing queues even after the backbone key has been changed (for maximum 1 s).

Nevertheless, the management server shall immediately after the change of the backbone key start the timer synchronization process as described in [5.7.2.2.3 b](#)) 8). Taking internal processing delays into account, the synchronization procedure shall start no later than 1 s after having received the backbone key change.

d) Usage by the Management client (ETS):

A Management client shall generate a different backbone key for each security domain. In particular, when just changing the routing multicast address of a security domain, it shall also generate and write a new and different backbone key for this new security domain.

A backbone network may not use the same backbone key on two different routing multicast addresses, because this introduces a security hole, because attackers can potentially stimulate desirable traffic on one multicast address and replay the traffic on the other multicast address. Different routing multicast address shall imply different backbone keys (but not vice versa).

5.7.2.5.3 PID_DEVICE_AUTHENTICATION_CODE (PID = 92)

a) Abstract resource definition:

This parameter shall contain the device authentication code used for authentication in secure unicast communication setup.

The device authentication code is a key derived from a user chosen shared secret. It will be calculated by the management client before writing it into this property and the KNXnet/IP secure client when verifying the KNXnet/IP secure server MAC in the session response.

$$\text{DeviceAuthenticationCode} = \text{PBKDF2}(\text{HMAC-SHA256}, \langle \text{SECRET} \rangle, \text{"device-authentication-code.1.secure.ip.knx.org"}, 65.536, 128)$$

It shall be set by the tool software when the device is configured for the first time. It is recommended that each device's authentication code should be individual (randomly chosen by the tool software), but the user can manually set this code to a user-specific value to simplify connection handling.

b) Security:

The device authentication code shall not be readable and shall only be writable using application layer services secured with the tool key (authentication and confidentiality).

c) Usage by the Management server:

In ex-factory state and after every factory reset the device authentication code shall be the factory default setup key printed on the device (this is also used for secured application layer services in ex-factory state).

Changes to the device authentication code shall not affect any existing secure sessions authenticated with past device authentication codes. After changing the device authentication code, all subsequent session response frames will be authenticated with the new device authentication code. Taking internal processing delays into account, the KNXnet/IP server shall start using the new device authentication code no later than 1 s after having received the request to change the device authentication code.

5.7.2.5.4 PID_PASSWORD_HASHES (PID = 93)

a) Abstract resource definition:

This array shall contain hashes of the passwords used for authenticating the client during secure unicast connection setup.

The password hashes are derived from the user chosen password texts. They will be calculated by the management client before writing it into this property and the KNXnet/IP secure client when authenticating a secure session for a given user. $\text{PasswordHash} = \text{PBKDF2}(\text{HMAC-SHA256}, \langle \text{PASSWORD} \rangle, \text{"user-password.1.secure.ip.knx.org"}, 65.536, 128)$

The index of the password in the array identifies a user (= User ID, see 5.7.2.3.8). Therefore, the minimum size is 1 (the first entry is always reserved for the management client granting full access) and the maximum size is 127 (management client + maximum 126 additional limited users).

b) Security:

The password hashes shall not be readable and shall only be writable using application layer services secured with the tool key (authentication and confidentiality).

c) Usage by the management server:

In ex-factory state and after every factory reset shall have the passwords set to the empty string (resulting in the fixed hash E9C304B914A35175FD7D1C673AB52FE1). The tool software shall remind the user if the password has not yet been changed for a device. For changing the passwords, the KNX restriction of at most one active device management connection for a given time applies. This ensures that an attacker is unable to interfere with the password change by opening a parallel connection.

A password change shall only apply to new connections. User connections authenticated using the old password shall remain authenticated after the password change. The password shall only

be used for authenticating the client during the secure session setup, see [5.7.3.1](#). The password (hashes) are not used during later communication.

Taking internal processing delays into account, the KNXnet/IP server shall start using the new password no later than 1 s after having received the request to change the current password.

d) Error handling:

If the management client requests to write one or more password hashes at property value array elements beyond the supported range, then the standard error handling for A_PropertyValue_Write shall apply, this is, the request shall be confirmed with an A_PropertyValue_Response-PDU with nr_of_elem = 0 and no data.

5.7.2.5.5 PID_SECURED_SERVICE_FAMILIES (PID = 94)

a) Abstract resource definition:

This function property shall control the service families for which security is enforced as well as the required version of the security services. Only dedicated service families can be secured. For example, the core discovery services are intentionally not secured at all to allow any client to search for devices on the network and discover their security requirements. The core connection services will follow the security requirements of the corresponding connection-oriented service families which make use of these services.

b) Security:

Please refer to the above access requirements. These requirements are exclusive: other roles, security features or services shall not have access to this property. Regardless of the device security mode, command access to this property shall only be possible using secure communication.

c) Initial state:

The initial state of all service families supported by the device shall be set to not require security.

d) Write (A_FunctionPropertyCommand-PDU) — WriteServiceID 00h: Write the IP service family security requirement

A basic format and common handling of A_FunctionProperty_Write-PDU is shown in [Figure 98](#).

octet 10	octet 11	octet 12 ... octet n
Reserved	ServiceID	ServiceInfo
	WriteServiceID	
00h	See below.	See below.

Figure 98 — Basic format and common handling of A_FunctionProperty_Write-PDU

An overview of PID_SECURED_SERVICE_FAMILIES WriteServiceIDs is given in [Table 59](#).

Table 59 — Overview PID_SECURED_SERVICE_FAMILIES WriteServiceIDs

WriteServiceID	Description
00h	Write the security requirements for a given KNXnet/IP service family (required security version)

A A_FunctionProperty_Write-PDU for Write ServiceID 00h for PID_SECURED_SERVICE_FAMILIES is shown in [Figure 99](#).

octet 10	octet 11	octet 12	octet 13
reserved	ServiceID	ServiceInfo	
	Write Security Requirement	Service Family ID	Security Version
00h	00h	See below.	See below.

Figure 99 — A_FunctionProperty_Write-PDU for Write ServiceID 00h for PID_SECURED_SERVICE_FAMILIES

The management client shall use this command to disable or enable the requirement to use a specific security version for a given KNXnet/IP service family. This means that the KNXnet/IP device shall generate KNXnet/IP frames for this KNXnet/IP service family only according to the configured security version. The KNXnet/IP device may accept frames for a specific KNXnet/IP service family encoded in higher security versions only if this is explicitly allowed by these higher security versions.

For the security version as specified in this document, only the following KNXnet/IP service family IDs are allowed as service info, see [Table 60](#):

Table 60 — KNXnet/IP service family IDs that are allowed as service info

ServiceID:		00h	Command “Write Service Family Security Requirement”
ServiceInfo:	Service Family ID	03h:	KNXnet/IP device management
		04h:	KNXnet/IP Tunnelling
		05h:	KNXnet/IP routing
	Security Version	00h:	Allow this KNXnet/IP service family to use plain communication without any security extension.
		01h:	Require that this KNXnet/IP service family shall use at least the security as specified in the document.
		>01h:	Reserved. For future use.

— Service family ID 05h: KNXnet/IP routing:

The security status of the KNXnet/IP routing service family is part of the KNX/IP domain configuration. It can therefore not only be changed via this function property but also via A_DomainAddressSerialNumber_Write-PDU. The behaviour of the management server upon enabling/disabling security for KNXnet/IP routing shall not differ depending on the way this resource was accessed.

Enabling or disabling security for the KNXnet/IP routing service family mainly affects the generation of KNXnet/IP routing.ind frames. KNXnet/IP routing.ind frames already generated with the previous security configuration (off or wrapped according to a configured security version) may still be sent from outgoing queues even after the security setting for the KNXnet/IP routing service family has been changed (for maximum 1 s).

The KNXnet/IP server shall immediately after enabling or disabling security for the KNXnet/IP routing service family start or stop the timer synchronization process as described in [5.7.2.2.3 b\) 8](#)). Taking internal processing delays into account, timer synchronization shall start or stop no later than 1 s after having received the request to change the security setting for the KNXnet/IP routing service family.

— Service family ID 03h: KNXnet/IP device management:

Changes to the security version of the KNXnet/IP device management service family shall not affect any existing device management connection. Especially when enabling security, existing plain (and unauthenticated) connections shall remain intact. The management client

may reset the management server (thus terminating all active sessions and connections) if the management client wants to prevent existing insecure device management connections remaining connected after having required security for the KNXnet/IP device management service family.

Enabling or disabling security for the KNXnet/IP device management service family affects the requirement for accepting device management connection requests only inside a sufficiently authenticated secure session. Taking internal processing delays into account, the KNXnet/IP server shall apply or ignore this requirement no later than 1 s after having received the request to change the security setting for the KNXnet/IP device management service family.

— Service family ID 04h: KNXnet/IP tunnelling:

Changes to the security version of the KNXnet/IP tunnelling service family shall not affect any existing tunnelling connection. Especially when enabling security, existing plain (and unauthenticated) connections shall remain intact. The management client may reset the management server (thus terminating all active sessions and connections) if the management client wants to prevent existing insecure tunnelling connections remaining connected after having required security for the KNXnet/IP tunnelling service family.

Enabling or disabling security for the KNXnet/IP tunnelling service family affects the requirement for accepting tunnelling connection requests only inside a sufficiently authenticated secure session. Taking internal processing delays into account, the KNXnet/IP server shall apply or ignore this requirement no later than 1 s after having received the request to change the security setting for the KNXnet/IP tunnelling service family.

— Response handling:

If the management server can successfully handle the command then it shall set the security version as requested and respond with an A_FunctionPropertyState_Response-PDU as in [Figure 100](#).

If the management server has any problem, then it shall respond as specified in [5.7.2.5.6](#).

The Management server shall respond with an A_FunctionPropertyState_Response-PDU with the appropriate positive or negative return code as listed in [5.7.2.5.6](#).

octet 10	octet 11
Return Code	ServiceID
	Write Security Requirement
	00h

Figure 100 — A_FunctionPropertyState_Response-PDU for WriteServiceID 00h for PID_SECURED_SERVICE_FAMILIES

- e) Read (A_FunctionPropertyState_Read-PDU) — ReadServiceID 00h: Read the IP service family security requirement:

If the management server can respond to the command then it shall respond with an A_FunctionPropertyState_Response-PDU with a positive return code, the ReadServiceID and the ServiceInfo as from the request (if available) and the command result as specified below in [Figure 101](#), [Figure 102](#) and [Table 61](#).

octet 10	octet 11	octet 12 ... octet n
Reserved	ServiceID	ServiceInfo
	ReadServiceID	
00h	See below.	See below.

Figure 101 — Basic format and common handling of A_FunctionPropertyState_Read-PDU

Table 61 — Overview PID_SECURED_SERVICE_FAMILIES ReadServiceIDs

ReadServiceID	Description
00h	Read the security requirements for a given KNXnet/IP service family (required security version)

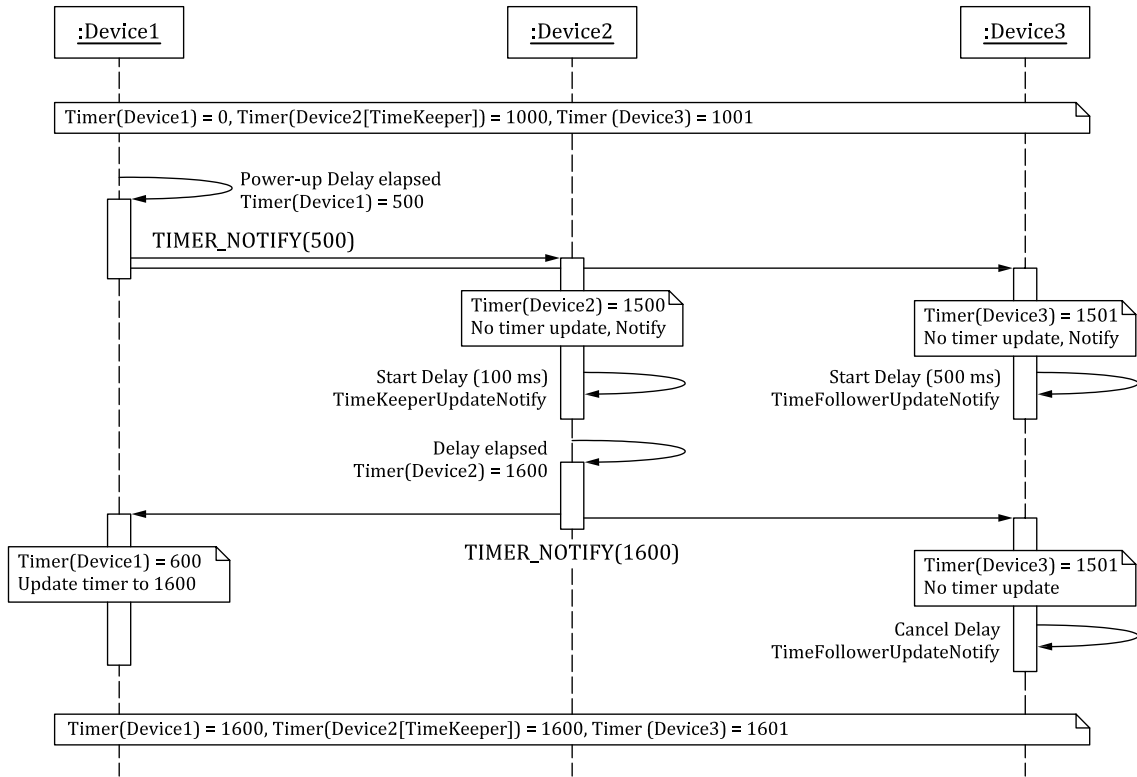


Figure 102 — A_FunctionPropertyState_Read-PDU for ReadServiceID 00h for PID_SECURED_SERVICE_FAMILIES

This command shall be used to read the currently configured security requirements for a given KNXnet/IP service family.

For the security version as specified in this document, only the following KNXnet/IP service family IDs are allowed as ServiceInfo in [Table 62](#):

Table 62 — KNXnet/IP service family IDs

ServiceID:		00h	Command "Read Service Family Security Requirement"
ServiceInfo:	Service Family ID	03h:	KNXnet/IP device management
		04h:	KNXnet/IP tunnelling
		05h:	KNXnet/IP routing

If the management server can successfully respond to the request then it shall respond with an A_FunctionPropertyState_Response-PDU as in [Figure 103](#) — A_FunctionPropertyState_Response-PDU for ReadServiceID 01h for PID_SECURED_SERVICE_FAMILIES.

If the management server has any problem then it shall respond as specified in [5.7.2.5.6](#).

The management server shall respond with an A_FunctionPropertyState_Response-PDU with the appropriate positive or negative return code as listed in [5.7.2.5.6](#).

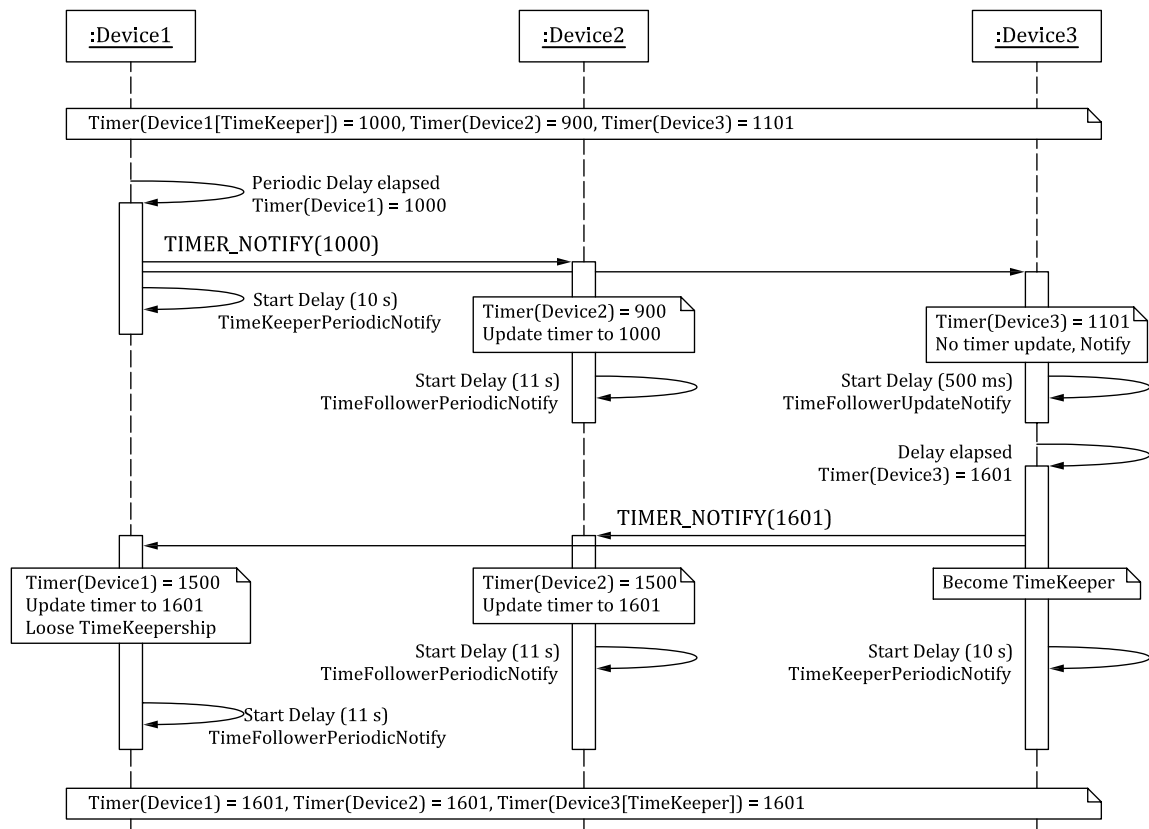


Figure 103 — A_FunctionPropertyState_Response-PDU for ReadServiceID 01h for PID_SECURED_SERVICE_FAMILIES

5.7.2.5.6 Common error — and exception handling for PID_SECURED_SERVICE_FAMILIES

In case of any error or exception, the management server shall return the specified A_FunctionPropertyState_Response-PDU (see [Figure 104](#)), but with the field return code as listed in [Table 63](#) and repeating the ServiceID — ReadServiceID or WriteServiceID as appropriate.

octet 10	octet 11
Return Code	ServiceID
> 7Fh	as in the request

Figure 104 — Common A_FunctionPropertyState_Read-PDU with error for PID_SECURED_SERVICE_FAMILIES

Table 63 — Return codes

Return code	ErrorName	ErrorType
FFh	E_ERROR	Error
	The service, function or command has failed without a closer indication of the problem. This return code shall only be used in case of an error for which no return code is defined or supported.	
A1h	E_COMMAND_INVALID	Invalid command
	This return code shall be used in case a not defined or not supported ReadServiceID or WriteServiceID is requested.	
F8h	E_DATA_VOID	Void data
	This return code shall be used in case any field in the request, except the ReadServiceID or WriteServiceID has an invalid or not supported value.	
	This return code shall be used if octet 10 in the request does not equal 00h.	
	This return code shall be used if the field ServiceInfo has an invalid or not supported value (e.g. not supported/allowed service family ID).	

5.7.2.5.7 PID_MULTICAST_LATENCY_TOLERANCE (PID = 95)

a) Abstract resource definition:

This parameter specifies the length of the acceptance window for accepting incoming multicast frames timestamped with a past multicast timer value.

This acceptance window is configurable to suit the needs of the specific installation (should be as small as possible, to be secure, but it can be bigger if there is more latency on the network). The maximum age of a telegram to be accepted as a valid telegram affects the reliability of the system. A greater time span will improve tolerance against network latencies while a shorter time span will improve the protection against replay attacks. A reasonable value for this time span is largely dependent on the expected network latencies.

b) Security:

The multicast latency tolerance can be read using plain application layer services, but shall only be writable using application layer services secured with the tool key (authentication and confidentiality).

c) Usage by the management server:

In ex-factory state and after every factory reset the multicast latency tolerance shall be 2 000 ms.

Changing the multicast latency tolerance affects the reception of multicast KNXnet/IP secure wrapper frames and decisions taken in the timer sync state machine as described in [5.7.2.2.3. b\)](#). Taking delays due to internal processing time into account, the acceptance of KNXnet/IP secure wrapper frames and the timer sync behaviour shall adapt to the new multicast latency tolerance value no later than 1 s after having received the request to change the multicast latency tolerance.

5.7.2.5.8 PID_SYNC_LATENCY_FRACTION (PID = 96)

a) Abstract resource definition:

This parameter specifies the fraction of the KNXnet/IP secure multicast latency tolerance in PID_MULTICAST_LATENCY_TOLERANCE (PID = 95) that shall be used as accepted latency of received secure wrapper or timer notify frames when deciding if the timer notify scheduled by the receiving device shall be cancelled/rescheduled or not.

b) Security:

The multicast latency tolerance fraction can be read using plain application layer services, but shall only be writable using application layer services secured with the tool key (authentication and confidentiality).

c) Usage by the management server:

In ex-factory state and after every factory reset the multicast latency tolerance fraction shall be 10,2 % (0x1a).

Changing the multicast latency tolerance fraction affects decisions taken in the timer sync state machine as described in [5.7.2.2.3](#). b). Taking delays due to internal processing time into account, the timer sync behaviour shall adapt to the new value no later than 1 s after having received the request to change the multicast latency tolerance fraction.

5.7.2.5.9 PID_TUNNELLING_USERS (PID = 97)

a) Abstract resource definition:

The tunnelling users table shall link the tunnelling slots with their configured individual addresses (see PID_TUNNELLING_ADDRESSES of KNXnet/IP Tunnelling v2) with the users (see [5.7.2.5.4](#)) having the rights to connect with these tunnelling addresses.

This authorization shall only be applied for authenticated sessions. If security is disabled for KNXnet/IP tunnelling, every unauthenticated management client has access to all of the available tunnelling addresses.

b) Security:

The tunnelling users table shall be readable and writable using application layer services secured with the tool key (authentication and confidentiality) only.

c) Format:

The tunnelling users table shall be an array property of which each element shall contain the relation between one user ID, which is the index into the Password Hashes table, and one tunnelling address, through its index into the tunnelling addresses table, see [Figure 105](#).

Array Index	User ID	Tunnelling Address Index
	1 octet	1 octet
1		
2		
...
N		

Figure 105 — Tunnelling users table

The tunnelling users table is sorted by user ID (first) and tunnelling address index (second). The same user ID may be granted access to more than one tunnelling address and one tunnelling address may be accessible by multiple users IDs.

The management user (= User ID 01h) has implicit access to all tunnelling addresses and shall not be included in the tunnelling users table.

It is not required that every tunnelling address has at least one entry in the tunnelling users table. Addresses without an entry in the tunnelling users table shall only be usable by the management user (= User ID 01h) if security is enabled for the tunnelling service in the secured services property.

d) Usage by the management server (device):

When an authenticated secure session is established and the management client requests a KNXnet/IP tunnelling connection, the management server shall through the user ID associated with the secure session search the tunnelling address(es) the user ID has access rights for.

If the tunnelling client does not request a specific tunnelling address to connect to, the tunnelling server shall connect with any of the authorized and free addresses (if any). Otherwise the tunnelling server shall respond with the appropriate error code.

If the user requests a specific tunnelling address and this address is in the tunnelling users table not assigned to the user, the management server shall respond with an appropriate error code.

The tunnelling user table shall only be used for authorization of the client during the connection setup. The tunnelling user table shall not be used during later communication.

A change to the tunnelling user table shall thus only affect new connections and existing connections already authorized to legitimate tunnelling users shall remain active. Taking internal processing delays into account, the KNXnet/IP server shall evaluate the new tunnelling user table no later than 1 s after having received the request to change the current tunnelling user table.

In ex-factory state and after every factory reset the tunnelling users table shall be empty.

— Error handling:

The tunnelling server (management server) may check the validity of the contents of the KNXnet/IP tunnelling users.

EXAMPLE 1 Presence of an invalid user ID; sorting error in the table, etc.

— The reaction of the tunnelling server on invalid entries is implementation specific.

EXAMPLE 2 The tunnelling server may not accept tunnelling connections at all, or it may only accept tunnelling connections up to where PID_TUNNELLING_USERS appears to be valid, or other.

e) Usage by the management client (ETS)

The management client shall write the entries of the tunnelling users table, from the lower indexes to the higher, sorted firstly according the field user ID and then according the field tunnelling address.

The management client may reset the management server (thus terminating all active sessions and connections) if the management client wants to prevent no longer existing tunnelling users to remain connected after having changed the tunnelling user table.

5.7.3 Management procedures

5.7.3.1 Secure session setup

See [Figure 106](#) — Secure setup handshake for a sequence diagram of the handshake taking place during secure session setup (the handling of authentication failures is omitted for better readability).

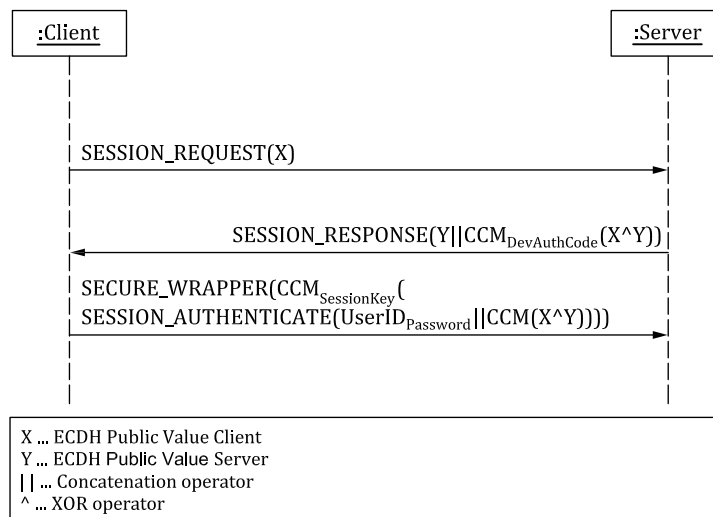


Figure 106 — Secure session setup handshake

During secure session setup the Elliptic-Curve Diffie-Hellman (ECDH) key agreement algorithm is used to agree on a common secret key. Since the original ECDH algorithm is vulnerable to a man-in-the-middle attack, a modified form of the algorithm is used which integrates mutual authentication of client and server. For authentication the ECDH public values sent between client and server are authenticated and encrypted in a way that protects the symmetrical keys used for authentication against offline dictionary attacks.

Establishing a secure session between a client and a server device is always initiated by the client sending a SESSION_REQUEST frame to the server. This frame contains the client's ECDH public value X.

For every received SESSION_REQUEST frame the server reserves a session identifier, generates the ECDH parameters and calculates the session key for this secure session. The server then answers with a SESSION_RESPONSE frame containing the secure session identifier, the server's ECDH public value Y and an authentication part.

When the server sends out a successful secure session response, the returned session identifier is marked as "connected" and shall not be reused until it is disconnected again.

Within 10 s after the secure session was established, the KNXnet/IP secure client has to start authentication of the user. Therefore, the KNXnet/IP secure client sends a SESSION_AUTHENTICATE request wrapped inside a SECURE_WRAPPER frame to the KNXnet/IP secure server. The client expects a SESSION_STATUS with the authentication result from the KNXnet/IP secure server within a timeout period of 10 s.

Error handling:

- The general KNXnet/IP error handling shall apply (see 5.2, this is, any frame received with an incorrect value of a wrong length, unknown protocol version, unknown service type or wrong value of a reserved field, shall be ignored).
- If the client sends any other SECURE_WRAPPER except a wrapped SESSION_AUTHENTICATE request before successful authentication, the server shall respond with a SESSION_STATUS with status field of STATUS_UNAUTHENTICATED.
- In case of the server detects any of the below listed authentication failures then it shall send a SESSION_STATUS frame with a status field of STATUS_AUTHENTICATION_FAILED to the client and the connection setup handshake shall be immediately aborted (the secure session disconnected). The following are authentication failures.
 - Use of a reserved user ID (this is, in the range 80h-FFh).

- Use of a user ID in the range 02h to 7Fh to which no password hash is assigned in PID_PASSWORD_HASHES.
- Use of a password to calculate the MAC of the SESSION_AUTHENTICATE which is not consistent with the password hash stored in PID_PASSWORD_HASHES.

Otherwise, if the authentication is successful, the status of the SESSION_STATUS shall be STATUS_AUTHENTICATION_SUCCESS.

- If no valid SECURE_WRAPPER frame is received by the server for a connected and authenticated session identifier within the session timeout after the last valid frame for this session, the server assumes a communication error, sends a final SESSION_STATUS frame with a status field of STATUS_TIMEOUT and closes the connection. The client may keep the session open by sending a SESSION_STATUS frame with a status field of STATUS_KEEPALIVE before the timeout elapses.

After the secure connection has been established and successfully authenticated, a regular KNXnet/IP connection will typically be created on top of it by exchanging CONNECT_REQUEST and CONNECT_RESPONSE frames encapsulated as payload inside SECURE_WRAPPER frames.

Within an authenticated secure session, the client may open any number of secure connections of any connection type. In addition, connection-less communication (e.g. DESCRIPTION_REQUEST) may be performed within the context of the session.

For all connections created within a secure session, the server shall accept only SECURE_WRAPPER frames associated with this session. SECURE_WRAPPER frames associated with other sessions and plain frames shall be ignored. For outgoing communication, the server shall use SECURE_WRAPPER frames associated with the owning session.

All other traffic belonging to the enclosed connections shall also always be securely encapsulated inside SECURE_WRAPPER frames for this session. The KNXnet/IP core services DISCONNECT_REQUEST and CONNECTIONSTATE_REQUEST shall return E_CONNECTION_ID if the referenced connection identifier is used outside the associated session (plain or within a different session).

Error handling:

- If the KNXnet/IP secure server receives a SECURE_WRAPPER frame containing a service code that requires an authenticated session for authorization, but the secure session has not successfully passed authentication, then the server shall send a SESSION_STATUS frame with a status field of STATUS_UNAUTHENTICATED and disconnect the secure session.
- If the KNXnet/IP secure server receives a SECURE_WRAPPER frame containing a disconnected session identifier, this frame shall be discarded.
- If the KNXnet/IP secure server receives a SECURE_WRAPPER frame containing a connected session identifier, but the payload could not be successfully decrypted and/or authenticated, this frame shall be discarded.

5.7.3.2 Closing a secure session

Both, the client and the server may close an established secure session at any time by sending a SESSION_STATUS frame with a status field of STATUS_CLOSE. In addition, the session is closed if the server sends a SESSION_STATUS with status codes STATUS_AUTHENTICATION_FAILED, STATUS_UNAUTHENTICATED or STATUS_TIMEOUT.

When a secure session is closed, all connections opened within the session shall be closed implicitly and all associated resources shall be released. This implicit connection close shall not be reported to the client.

Secure sessions shall be closed implicitly when the surrounding TCP connection closes.

5.7.3.3 Retransmission of lost frames

For multicast communication, UDP packets (and thus KNXnet/IP frames) may get lost, reordered and duplicated during transmission. The communication partners need to handle losing, reordering and duplicating KNXnet/IP frames on the application layer. KNXnet/IP multicast applications are directly exposed to the peculiarities of the UDP protocol implementation of the IP infrastructure used. The KNXnet/IP secure layer for multicast does not handle losing, reordering and duplication of KNXnet/IP frames in any special way.

For unicast connections (secure sessions) KNXnet/IP frames are never lost, reordered or duplicated because they use TCP as a reliable transport. Thus, the KNXnet/IP application is not exposed to the peculiarities of the TCP protocol implementation or the IP infrastructure.

5.7.4 Synchronizing timers

Figure 107 shows how a device synchronizes its local timer to the devices in an existing installation after power up. It can be seen how the random delays work together to minimize the number of frames actually sent dependent on the transmitted timer values.

If no TIMER_NOTIFY was received by device 1, it would continue sending as time keeper its periodic TIMER_NOTIFY frames about every 10 s.

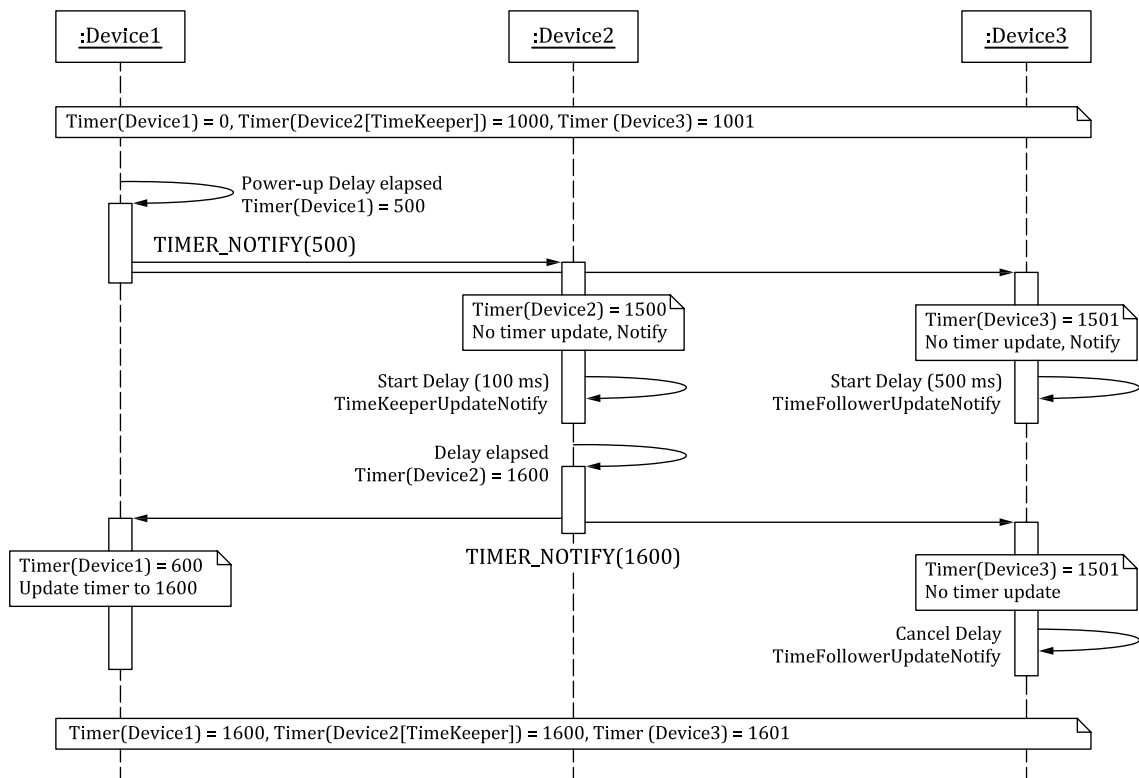


Figure 107 — Timer synchronization after power up

Figure 108 shows how devices are cyclically resynchronized to keep the lag between the individual device timers small. It also helps to achieve synchronization between former physically independent KNXnet/IP multicast groups being connected together.

Device 1 initiates the synchronization process. Each device has a cyclic synchronization timer expiring at a fixed time with a small random variation. In this example the Device 1 is initially time keeper and its periodic timer was the first to expire.

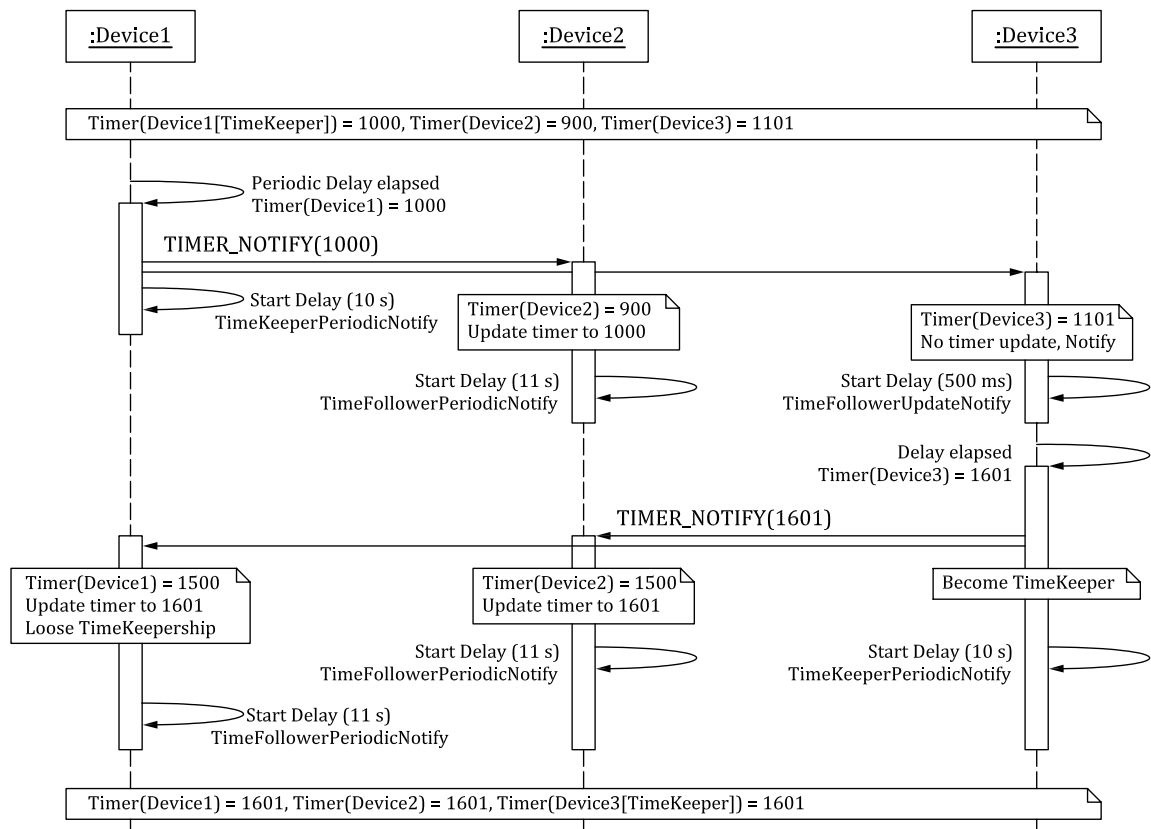


Figure 108 — Cyclic synchronization of timers

The figure shows how the devices interact sending and receiving timer notification frames and finally arrive with all device timer synchronized again.

Annex A (normative)

List of codes

A.1 General

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

Common KNXnet/IP constants are shown in [Table A.1](#).

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 can therefore be supported. As this is version 1 of the specification, 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
0900h	... 09FFh	KNXnet/IP secure

A.3.2 Core KNXnet/IP services

KNXnet/IP core service type identifiers are shown in [Table A.2](#).

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

KNXnet/IP device management service type identifiers are shown in [Table A.3](#).

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

Tunnelling KNXnet/IP service type identifiers are shown in [Table A.4](#).

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 TUNNELLING_REQUEST.
TUNNELLING_FEATURE_GET	0422h	2	Used by the KNXnet/IP tunnelling client to read out the value of a feature from the KNXnet/IP tunnelling server.

Table A.4 (continued)

Service name	Code	V.	Description
TUNNELLING_FEATURE_RESPONSE	0423h	2	Used by the KNXnet/IP tunnelling server to respond to a feature set or gotten by the KNXnet/IP tunnelling client.
TUNNELLING_FEATURE_SET	0424h	2	Used by the KNXnet/IP tunnelling client to set the value of a feature of the KNXnet/IP tunnelling server.
TUNNELLING_FEATURE_INFO	0425h	2	Used by the KNXnet/IP tunnelling server to inform the KNXnet/IP tunnelling client on a value of an interface feature.

A.3.5 Routing services

KNXnet/IP routing service type identifiers are shown in [Table A.5](#).

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.
ROUTING_SYSTEM_BROADCAST	0533h	1	Used for configuration of devices to be added to a KNX/IP domain.

A.3.6 Remote diagnosis and configuration

KNXnet/IP remote diagnosis and configuration service type identifiers are shown in [Table A.6](#).

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.4 Connection types

A.4.1 General

Connection types are shown in [Table A.7](#).

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.
REMCNFG_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.4.2 KNXnet/IP secure services

KNXnet/IP secure service type identifiers are shown in [Table A.8](#).

Table A.8 — KNXnet/IP secure service type identifier

Service name	Code	V.	Description
SECURE_WRAPPER	0950h	1	Contains an encrypted frame plus data for ensuring data integrity and freshness.
SESSION_REQUEST	0951h	1	Used to initiate the secure connection setup handshake for a new secure communication session.
SESSION_RESPONSE	0952h	1	Sent in response to a received secure session request frame.
SESSION_AUTHENTICATE	0953h	1	Sent after the Diffie-Hellman handshake to authenticate the user against the server.
SESSION_STATUS	0954h	1	Sent in any stage of the secure session handshake to indicate an error condition or status information.
TIMER_NOTIFY	0955h	1	Used to ensure synchronization of the multicast group member's timer values.

A.5 Error codes

A.5.1 Common error codes

Common KNXnet/IP error codes are shown in [Table A.9](#).

Table A.9 — 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

Common CONNECT_RESPONSE status codes are shown in [Table A.10](#).

Table A.10 — Common CONNECT_RESPONSE status codes

Error constant	Value	V.	Description
E_NO_ERROR	00h	1	The connection was established successfully.
E_ERROR	0Fh	2	Any further undefined, possibly implementation specific error has occurred.
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.
E_CONNECTION_IN_USE	2Eh	2	The IA requested for this connection is in use.
E_AUTHORISATION_ERROR	28h	2	The client is not authorised to use the requested IA in the extended CRI.
E_NO_TUNNELLING_ADDRESS	2Dh	2	The IA requested in the extended CRI is not a tunnelling IA.

A.5.3 CONNECTIONSTATE_RESPONSE status codes

CONNECTIONSTATE_RESPONSE status codes are shown in [Table A.11](#).

Table A.11 — 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

Tunnelling CONNECT_ACK error codes are shown in [Table A.12](#).

Table A.12 — 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

Device management DEVICE_CONFIGURATION_ACK status codes are shown in [Table A.13](#).

Table A.13 — Device management DEVICE_CONFIGURATION_ACK status codes

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

A.5.6 Secured communication status codes

Secure session status codes are shown in [Table A.14](#).

Table A.14 — Secure session status codes

Service name	Value	V.	Description
STATUS_AUTHENTICATION_SUCCESS	00h	1	The user could successfully be authenticated.
STATUS_AUTHENTICATION_FAILED	01h	1	An error occurred during secure session handshake.
STATUS_UNAUTHENTICATED	02h	1	The session is not (yet) authenticated.
STATUS_TIMEOUT	03h	1	A timeout occurred during secure session handshake.
STATUS_CLOSE	04h	1	The secure session shall be closed.
STATUS_KEEPALIVE	05h	1	Prevent inactivity on the secure session closing it with timeout error.

A.6 Description information block (DIB)

[Table A.15](#) lists the valid description type codes.

[Table A.16](#) lists the valid KNX medium codes.

Table A.15 — 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.
KNX_ADRESSES	05h	1	KNX addresses used by/assigned to the device.
SECURED_SERVICES	06h	1	Secure configuration
Reserved	07h–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.16 — 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.17](#) lists the valid host protocol codes.

Table A.17 — 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.18](#) lists the valid timeout constants.

Table A.18 — Timeout constants

Constant name	Value	V.	Description
CONNECT_REQUEST_TIMEOUT	10 s	1	KNXnet/IP client shall wait for 10 s for a CONNECT_RESPONSE frame from KNXnet/IP server.
CONNECTIONSTATE_REQUEST_TIMEOUT	10 s	1	KNXnet/IP client shall wait for 10 s for a CONNECTIONSTATE_RESPONSE frame from KNXnet/IP server.
DEVICE_CONFIGURATION_REQUEST_TIMEOUT	10 s	1	KNXnet/IP client shall wait for 10 s for a DEVICE_CONFIGURATION_RESPONSE frame from KNXnet/IP server.
TUNNELLING_REQUEST_TIMEOUT	1 s	1	KNXnet/IP client shall wait for 1 s 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 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.

A.9 Internet protocol constants

[Table A.19](#) lists the Internet protocol relevant values.

Table A.19 — 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 frames

B.1 SEARCH_REQUEST

The SEARCH_REQUEST frame binary format: IP example is shown in [Figure B.1](#).

•		+-----+	
•	1	06h	header size
•		+-----+	
•	2	10h	protocol version
•		+-----+	
•	3	02h \	
•		+ - - - - - +	> service type identifier 0201h
•	4	01h /	
•		+-----+	
•	5	00h \	
•		+ - - - - - +	> total length, 14 octets
•	6	0Eh /	
•		+-----+	
•	7	08h	structure length
•		+-----+	
•	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	0Eh \	
•		+ - - - - - +	> port number of control endpoint, 3671
•	14	57h /	
•		+-----+	

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

B.2 SEARCH_RESPONSE

The SEARCH_RESPONSE frame binary format: IP example is shown in [Figure B.2](#).

•		+-----+		
•	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

The KNXnet/IP DESCRIPTION_REQUEST frame binary format: example is shown in [Figure B.3](#).

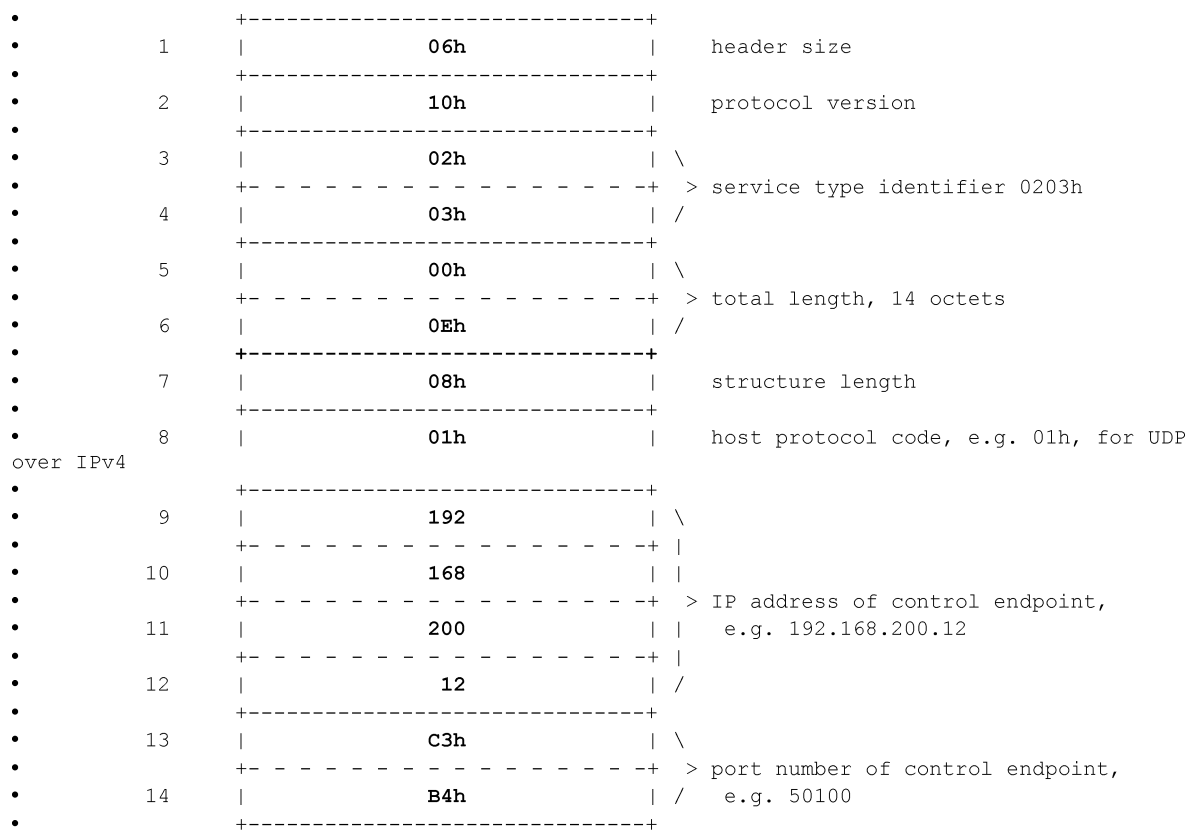


Figure B.3 — KNXnet/IP DESCRIPTION_REQUEST frame binary format: example

B.4 DESCRIPTION_RESPONSE

The DESCRIPTION_RESPONSE frame binary format: IP example is shown in [Figure B.4](#).

•		+-----+		
•	1	06h	header size	
•		+-----+		
•	2	10h	protocol version	
•		+-----+		
•	3	02h \		
•		+-----+	> service type identifier 0204h	
•	4	04h /		
•		+-----+		
•	5	00h \		
•		+-----+	> total length, 86 octets	
•	6	56h /		
•		+-----+		
•	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	
•		+-----+		

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)

•		+-----+		
•	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
•		+-----+		
•	79		08h	structure length (DIB other device
information)				
•		+-----+		
•	80		FEh	description type code, FEh = MFR_DATA
•		+-----+		
•	81		00h	\
•		+-----+		
•	82		01h	/
•		+-----+		
•	83		'N'	\
•		+-----+		
•	84		'1'	
•		+-----+		
•	85		'4'	
•		+-----+		
•	86		'6'	/
•		+-----+		

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

B.5 CONNECT_REQUEST

The KNXnet/IP CONNECT_REQUEST frame binary format: example is shown in [Figure B.5](#).

•		+-----+	
•	1	06h	header size
•		+-----+	
•	2	10h	protocol version
•		+-----+	
•	3	02h	\
•		+-----+	> service type identifier 0205h
•	4	05h	/
•		+-----+	
•	5	00h	\
•		+-----+	> total length, 24 octets
•	6	18h	/
•		+-----+	
•	7	08h	structure length
•		+-----+	
•	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	08h	structure length
•		+-----+	
•	16	01h	host protocol code, e.g. 01h, for UDP
•	over IPv4		
•		+-----+	
•	17	192	\
•		+-----+	
•	18	168	
•		+-----+	> IP address of data endpoint,
•	19	200	e.g. 192.168.200.20
•		+-----+	
•	20	20	/
•		+-----+	
•	21	C3h	\
•		+-----+	> port number of data endpoint,
•	22	B4h	e.g. 50100
•		+-----+	/
•	23	02h	structure length
•		+-----+	
•	24	04h	connection type code, e.g. 04h,
•	TUNNEL_CONNECTION		
•		+-----+	

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

B.6 CONNECT_RESPONSE

The CONNECT_RESPONSE frame binary format: IP example is shown in [Figure B.6](#).

•		+-----+ 		
•	1		06h	header size
•		+-----+ 		
•	2		10h	protocol version
•		+-----+ 		
•	3		02h	\
•		+-----+ 		
•	4		06h	/
•		+-----+ 		
•	5		00h	\
•		+-----+ 		
•	6		14h	/
•		+-----+ 		
•	7		15h	communication channel ID, e.g. 21
•		+-----+ 		
•	8		00h	status code (NO_ERROR)
•		+-----+ 		
•	9		08h	structure length
•		+-----+ 		
•	10		01h	host protocol code, e.g. 01h, for UDP
•	over IPv4			
•		+-----+ 		
•	11		192	\
•		+-----+ 		
•	12		168	
•		+-----+ 		
•	13		200	
•		+-----+ 		
•	14		20	/
•		+-----+ 		
•	15		C3h	\
•		+-----+ 		
•	16		B4h	/
•		+-----+ 		
•	17		04h	structure length of CRD for
•	TUNNELING_CONNECTION			
•		+-----+ 		
•	18		04h	connection type code, e.g. 04h,
•	TUNNEL_CONNECTION			
•		+-----+ 		
•	19		11h	\
•		+-----+ 		
•	20		0Ah	/
•		+-----+ 		

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

B.7 CONNECTIONSTATE_REQUEST

The CONNECTIONSTATE_REQUEST frame binary format: IP example is shown in [Figure B.7](#).

•		+-----+	
•	1	06h	header size
•		+-----+	
•	2	10h	protocol version
•		+-----+	
•	3	02h \	
•		+-----+	> service type identifier 0207h
•	4	07h /	
•		+-----+	
•	5	00h \	
•		+-----+	> total length, 16 octets
•	6	10h /	
•		+-----+	
•	7	15h	communication channel ID, e.g. 21
•		+-----+	
•	8	00h	reserved
•		+-----+	
•	9	08h	structure length
•		+-----+	
•	10	01h	host protocol code, e.g. 01h, for UDP
•	over IPv4		
•		+-----+	
•	11	192 \	
•		+-----+	
•	12	168	
•		+-----+	> IP address of control endpoint,
•	13	200	e.g. 192.168.200.12
•		+-----+	
•	14	12 /	
•		+-----+	
•	15	C3h \	
•		+-----+	> port number of control endpoint,
•	16	B4h /	e.g. 50100
•		+-----+	

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

B.8 CONNECTIONSTATE_RESPONSE

The CONNECTIONSTATE_RESPONSE frame binary format: IP example is shown in [Figure B.8](#).

•		+-----+	
•	1	06h	header size
•		+-----+	
•	2	10h	protocol version
•		+-----+	
•	3	02h \	
•		+-----+	> service type identifier 0208h
•	4	08h /	
•		+-----+	
•	5	00h \	
•		+-----+	> total length, 8 octets
•	6	08h /	
•		+-----+	
•	7	15h	communication channel ID, e.g. 21
•		+-----+	
•	8	00h	status code (NO_ERROR)
•		+-----+	

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

B.9 DISCONNECT_REQUEST

The DISCONNECT_REQUEST frame binary format: IP example is shown in [Figure B.9](#).

•		+-----+		
•	1	06h	header size	
•		+-----+		
•	2	10h	protocol version	
•		+-----+		
•	3	02h \		
•		+-----+		
•	4	09h /	> service type identifier 0209h	
•		+-----+		
•	5	00h \		
•		+-----+		
•	6	10h /	> total length, 16 octets	
•		+-----+		
•	7	15h	communication channel ID, e.g. 21	
•		+-----+		
•	8	00h	reserved	
•		+-----+		
•	9	08h	structure length	
•		+-----+		
•	10	01h	host protocol code, e.g. 01h, for UDP	
•	over IPv4			
•		+-----+		
•	11	192 \		
•		+-----+		
•	12	168		
•		+-----+		
•	13	200	> IP address of control endpoint, e.g. 192.168.200.12	
•		+-----+		
•	14	12 /		
•		+-----+		
•	15	C3h \		
•		+-----+		
•	16	B4h /	> port number of control endpoint, e.g. 50100	
•		+-----+		

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

B.10 DISCONNECT_RESPONSE

The DISCONNECT_RESPONSE frame binary format: IP example is shown in [Figure B.10](#).

•		+-----+		
•	1	06h	header size	
•		+-----+		
•	2	10h	protocol version	
•		+-----+		
•	3	02h \		
•		+-----+		
•	4	0Ah /	> service type identifier 020Ah	
•		+-----+		
•	5	00h \		
•		+-----+		
•	6	08h /	> total length, 8 octets	
•		+-----+		
•	7	15h	communication channel ID, e.g. 21	
•		+-----+		
•	8	00h	status code (NO_ERROR)	
•		+-----+		

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

B.11 DEVICE_CONFIGURATION_REQUEST

The DEVICE_CONFIGURATION_REQUEST frame binary format: example is shown in [Figure B.11](#).

```

.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          06h          | header size
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          10h          | protocol version
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          03h          | \
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          10h          | > service type identifier 0310h
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          00h          | \
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          nnh          | > total length, nn octets
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          04h          | - - - - connection header - - - -
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          15h          | structure length of connection header
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          00h          | communication channel ID, e.g. 21
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          00h          | sequence counter
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          00h          | reserved
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          fch          | - - - - CEMI frame - - - -
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          ...          | Message Code, e.g. M_Read.req
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          ...          | \
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          ...          | > Service Information
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          ...          |
.      +-----+-----+-----+-----+-----+-----+-----+-----+
.      |          ...          | /
.      +-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure B.11 — DEVICE_CONFIGURATION_REQUEST frame binary format: Example

B.12 DEVICE_CONFIGURATION_ACK

The DEVICE_CONFIGURATION_ACK frame binary format: example is shown in [Figure B.12](#).

		+-----+-----+	- - - KNXnet/IP header - - -
.	1	06h	header size
.		+-----+-----+	
.	2	10h	protocol version
.		+-----+-----+	
.	3	03h \	
.		+-----+-----+	> service type identifier 0311h
.	4	11h /	
.		+-----+-----+	
.	5	00h \	
.		+-----+-----+	> total length, 10 octets
.	6	0ah /	
.		+-----+-----+	- - - connection header - - -
.	7	04h	structure length of connection header
.		+-----+-----+	
.	8	15h	communication channel ID, e.g. 21
.		+-----+-----+	
.	9	00h	sequence counter
.		+-----+-----+	
.	10	00h	status
.		+-----+-----+	

Figure B.12 — DEVICE_CONFIGURATION_ACK frame binary format: example

B.13 TUNNELLING_REQUEST

The TUNNELLING_REQUEST frame binary format: example is shown in [Figure B.13](#).

•		+-----+ - - - KNXnet/IP header - - -
•	1	06h header size
•		+-----+
•	2	10h protocol version
•		+-----+
•	3	04h \
•		+-----+ > service type identifier 0420h
•	4	20h /
•		+-----+
•	5	00h \
•		+-----+ > total length, L+12 octets
•	6	L+0Ch /
•		+-----+ - - - 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

The TUNNELLING_ACK frame binary format: example is shown in [Figure B.14](#).

•		+-----+ - - - KNXnet/IP header - - -
•	1	06h header size
•		+-----+
•	2	10h protocol version
•		+-----+
•	3	04h \
•		+-----+ > service type identifier 0421h
•	4	21h /
•		+-----+
•	5	00h \
•		+-----+ > total length, 10 octets
•	6	0Ah /
•		+-----+ - - - 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

The ROUTING_INDICATION frame binary format: example is shown in [Figure B.15](#).

•		+-----+	
•	1	06h	header size
•		+-----+	
•	2	10h	protocol version
•		+-----+	
•	3	05h \	
•		+-----+	
•	4	30h /	> service type identifier 0530h
•		+-----+	
•	5	00h \	
•		+-----+	
•	6	L+08h /	> total length, L+08h octets
•		+-----+	
•	7	11h	- - - cEMI frame - - -
•		+-----+	
•	8	00h	message code (e.g. L_Data.req message)
•		+-----+	
•	9	... \	
•		+-----+	
•	10	...	> Service Information (L bytes)
•		+-----+	
•	L+8	... /	
•		+-----+	

Figure B.15 — ROUTING_INDICATION frame binary format: example

B.16 ROUTING_LOST_MESSAGE

The ROUTING_LOST_MESSAGE frame binary format: example is shown in [Figure B.16](#).

•		+-----+	
•	1	06h	header size
•		+-----+	
•	2	10h	protocol version
•		+-----+	
•	3	05h \	
•		+-----+	
•	4	31h /	> service type identifier 0531h
•		+-----+	
•	5	00h \	
•		+-----+	
•	6	0Ah /	> total length, 0Ah octets
•		+-----+	
•	7	04h	structure length
•		+-----+	
•	8	00h	device state
•		+-----+	
•	9	00h \	
•		+-----+	
•	10	05h /	> number of lost messages, e.g. 5
•		+-----+	

Figure B.16 — ROUTING_LOST_MESSAGE frame binary format: example

B.17 ROUTING_BUSY

The ROUTING_BUSY frame binary format: example is shown in [Figure B.17](#).

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, e.g. 100 ms
10		64h	/	
11		00h	\	> routing busy control value (default value: 0000h)
12		00h	/	

Figure B.17 — ROUTING_BUSY frame binary format: example

B.18 REMOTE_DIAGNOSTIC_REQUEST

The REMOTE_DIAGNOSTIC_REQUEST frame binary format: example is shown in [Figure B.18](#).

•		+-----+ - - - 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
•		+-----+
•	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

The REMOTE_DIAGNOSTIC_RESPONSE frame binary format: example is shown in [Figure B.19](#).

•		+-----+ - - - KNXnet/IP header - - -
•	1	06h header size
•		+-----+
•	2	10h protocol version
•		+-----+
•	3	07h \
•		+-----+ > service type identifier 0741h
•	4	41h /
•		+-----+
•	5	00h \
•		+-----+ > total length, 58 octets
•	6	3Ah /
•		+-----+ - - - 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
•		+-----+ > IP address
•	13	02h e.g. 192.168.2.12
•		+-----+
•	14	0Ch /
•		+-----+
•	15	FFh \
•		+-----+
•	16	FFh
•		+-----+ > subnet mask
•	17	FFh e.g. 255.255.255.0
•		+-----+
•	18	00h /
•		+-----+
•	19	C0h \
•		+-----+
•	20	A8h
•		+-----+ > default gateway IP address
•	21	02h 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
•		+-----+ > IP address
•	29	02h e.g. 192.168.2.12
•		+-----+
•	30	0Ch /
•		+-----+
•	31	FFh \
•		+-----+
•	32	FFh
•		+-----+ > subnet mask
•	33	FFh 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

The REMOTE_BASIC_CONFIGURATION_REQUEST frame binary format: example is shown in [Figure B.20](#).

			- - - 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
•		+-----+	
over IPv4	8	01h	host protocol code, e.g. 01h, for UDP
•		+-----+	
•	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

REMOTE_RESET_REQUEST frame binary format: example is shown in [Figure B.21](#).

•		+-----+ - - - KNXnet/IP header - - -
•	1	06h header size
•		+-----+
•	2	10h protocol version
•		+-----+
•	3	07h \
•		+-----+ > service type identifier 0743h
•	4	43h /
•		+-----+
•	5	00h \
•		+-----+ > total length, 10 octets
•	6	0Ah /
•		+-----+ - - - SELECTOR - - -
•	7	02h structure length of SELECTOR
•		+-----+
•	8	01h ProgrammProgramming Mode Selector
•		+-----+ - - - RESET_COMMAND - - -
•	9	01h restart
•		+-----+
•	10	00h reserved
•		+-----+

Figure B.21 — REMOTE_RESET_REQUEST frame binary format: example

Annex C (normative)

KNXnet/IP parameter object

[Table C.1](#) shows the property identifiers.

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
79	PID_TUNNELLING_ADDRESSES	—/r
91	PID_BACKBONE_KEY	—/r
92	PID_DEVICE_AUTHENTICATION_CODE	—/r
93	PID_PASSWORD_HASHES	—/r
94	PID_SECURED_SERVICE_FAMILIES	—/r
95	PID_MULTICAST_LATENCY_TOLERANCE	—/r
96	PID_SYNC_LATENCY_FRACTION	—/r
97	PID_TUNNELLING_USERS	—/r

Table C.2 shows the KNXnet/IP parameter object properties.

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.
<p>The following properties shall be implemented by KNXnet/IP devices providing KNXnet/IP routing:</p> <ul style="list-style-type: none"> —KNXnet/IP routing capabilities; —queue overflow to IP; —queue overflow to KNX; —telegrams transmitted to IP; —telegrams transmitted to KNX 			

Table C.2 (continued)

Property Name	Property ID	M/O	Comment
Time to live (TTL)	PID_TTL	M	TTL for IP routing
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	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	Number of telegrams lost because queue to IP overflowed.
Queue overflow to KNX	PID_QUEUE_OVERFLOW_TO_KNX	O	Number of telegrams lost because queue to KNX overflowed.
Telegrams transmitted to IP	PID_MSG_TRANSMIT_TO_IP	O	Number of telegrams successfully transmitted to IP.
Telegrams transmitted to KNX	PID_MSG_TRANSMIT_TO_KNX	O	Number of telegrams successfully transmitted to KNX.
Friendly name	PID_FRIENDLY_NAME	M	Mandatory for devices implementing KNXnet/IP
Secure backbone key	PID_BACKBONE_KEY	M	Mandatory for devices implementing KNXnet/IP secure
Device authentication code	PID_DEVICE_AUTHENTICATION_CODE	M	Mandatory for devices implementing KNXnet/IP secure
Password hashes	PID_PASSWORD_HASHES	M	Mandatory for devices implementing KNXnet/IP secure
Secured service families	PID_SECURED_SERVICE_FAMILIES	M	Mandatory for devices implementing KNXnet/IP secure
Multicast latency tolerance	PID_MULTICAST_LATENCY_TOLERANCE	M	Mandatory for devices implementing KNXnet/IP secure
Multicast sync latency fraction	PID_SYNC_LATENCY_FRACTION	M	Mandatory for devices implementing KNXnet/IP secure
Tunnelling users	PID_TUNNELLING_USERS	M	Mandatory for devices implementing KNXnet/IP secure
<p>The following properties shall be implemented by KNXnet/IP devices providing KNXnet/IP routing:</p> <ul style="list-style-type: none"> —KNXnet/IP routing capabilities; —queue overflow to IP; —queue overflow to KNX; —telegrams transmitted to IP; —telegrams transmitted to KNX 			

Annex D
(normative)

Common external messaging interface (cEMI)

D.1 cEMI

D.1.1 cEMI: Message format and services

D.1.1.1 General

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 (see [Figure D.1](#)).

D.1.1.2 Message flow — Overview

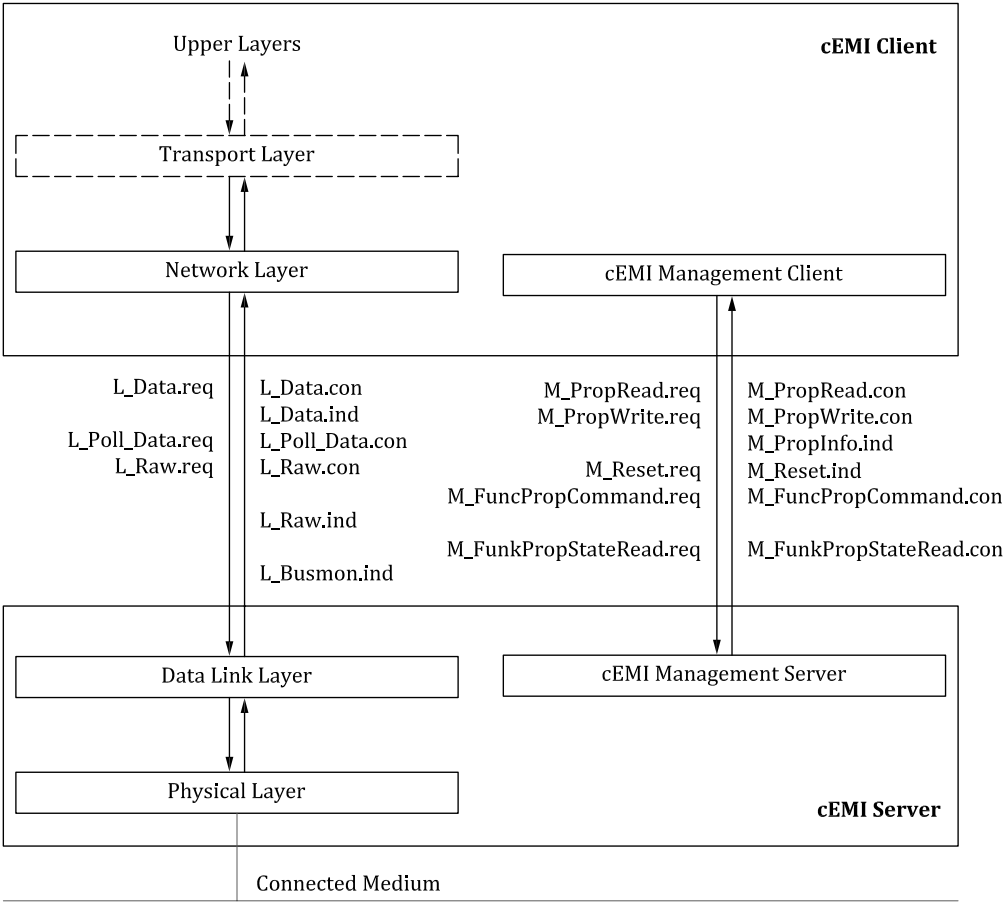


Figure D.1 — Message flow between cEMI client and cEMI server

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 and 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, see [Table D.1](#):

Table D.1 — cEMI message generic message structure

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 information length (AddIL) field are not present in local device management services.

A cEMI management message shall have the following generic structure, see [Table D.2](#):

Table D.2 — cEMI management message generic message structure

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:

- a) medium dependent information, and
- b) other information: e.g. (relative) timestamp and error flags busmonitor function.

The additional information field shall contain tagged information, identified by the type ID in [Table D.3](#):

Table D.3 — Additional information

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.

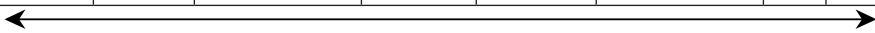
Combinations (concatenation) of additional information types can be used in the additional information field.

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. A message structure including additional information is shown in [Table D.4](#).

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. Windows™).

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 in [Table D.5](#):

Table D.5 — Message structure including additional information

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 = 1 000 (dec)
— Overflow after 232 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

The L_Data.req message without any additional information is shown in [Table D.6](#).

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

Message code	Additional info length	Service information
MC	AddIL
11h	0	...

L_Data.con message, from a powerline medium frame, with domain address is shown in [Table D.7](#)

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	...

L_Data.ind message, RF frame with RF control information and KNX serial number is shown in [Table D.8](#).

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	SN ₆	SN ₅	SN ₄	SN ₃	SN ₂	SN ₁	...

L_Data.ind message, RF frame with RF control information and RF domain address is shown in [Table D.9](#).

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 subclauses 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, for example 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 than 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

Basic frame structure for L_Data messages is shown in [Table D.10](#).

Table D.10 — Basic frame structure for L_Data messages

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

- a) MC: message code
- b) AddIL: length of additional information
- c) Ctrl1:

Control field 1

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

1) Frame type (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

2) Repetition (r) (bit 5):

description: Repeat, not valid for all media.

encoding: 0: Repeat frame on medium if error
1: Do not repeat

3) System broadcast (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

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

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

encoding: Please refer to "Usage of priority".

5) Acknowledge request (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

6) Confirm (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

d) Ctrl2:

Control field 2

1 octet							
Ctrl2							
7	6	5	4	3	2	1	0
AT	HC			EFF			

1) Destination address type (AT) (msb):

description: This defines the type of telegram sent.

encoding: 0: Individual
1: Group

2) Hop count (HC) (bit 6 to bit 4):

description: This defines the default number of hop counts before the message expires.

encoding: Value binary encoded

3) Extended frame format (EFF) [bit 3 to bit 0 (lsb)]:

description: This field contains the frame type.

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

- e) SAH: Source address high (source subnetwork address)
- f) SAL: Source address low (source device address)
- g) DAH: Destination address high (destination subnetwork address)
- h) DAL: Destination address low (destination device address)
- i) 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 — L_Data.req

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 (see [Table D.11](#)) with the 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 specification 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 a proper 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:

a) Frame type (FT):

description: This field shall specify whether the frame is a standard frame or an extended frame, see Table D.12.

encoding:

Table D.12 — Frame type

Value of FT-flag	Meaning/behaviour on bus media		
	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.			

b) 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), see Table D.13;
"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 — Repetition

Value of r-flag	Meaning/behaviour on bus media		
	TP1	PL110	RF
0: No repetitions	No repetitions	No repetitions	No repetitions
1: Don't care	Repetitions are allowed	Repetitions are allowed	No repetitions

c) 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 medium shall ignore the SB-flag.
- encoding: 0: System broadcast
1: Broadcast

d) 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 specifications of the communication medium. See Table D.14.
- encoding: 0: No acknowledge is requested
 1: Acknowledge requested

Table D.14 — Acknowledge request

Value of a-flag	Meaning/behaviour on bus media		
	TP1	PL110	RF
0: Don't care	Requested	Requested	Not applicable
1: Ack requested	Requested	Requested	Not applicable

e) Confirm (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 — L-Data.con

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	x0rxp-pxC

NOTE C: Confirm Flag: 1=Error; 0=No Error

The L_Data.con (see [Table D.15](#)) 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

a) 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

b) Repetition (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.

c) System broadcast (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.

d) Acknowledge request (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:

e) Confirm (C):

encoding: 0 = No Error
1 = Error

D.1.1.5.3.5 L_Data.ind

Table D.16 — L_Data.ind

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. L-Data.ind is shown in [Table D.16](#).

Use of flags in Ctrl1-field:

a) 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

b) Repetition (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

c) 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

d) Acknowledge request (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:

e) Confirm (C):

description: The C-flag in the L_Data.ind shall 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. For L_Poll_Data.req and L_Poll_Data.con see [Table D.17](#) and [Table D.18](#).

D.1.1.5.4.2 L_Poll_Data.req

Table D.17 — L_Poll_Data.req

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 — L_Poll_Data.con

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.1.1.5.5 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

The basic message structure for L_Raw messages is shown in [Table D.19](#).

Table D.19 — Basic message structure for L_Raw messages

Message code	Additional info length	Additional information	Raw frame on medium
MC	AddIL	...	Data
1 octet	1 octet	var. length	

NOTE Raw frame on medium: starting with the Ctrl octet & ending with the CRC octet(s).

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).

D.1.1.5.5.3 L_Raw.req

Table D.20 — L_Raw.req

Message code	Additional info length	Additional information	Raw frame on medium
MC	AddIL	...	Data
10h

NOTE 1 In EMI 2, the equivalent service is specified as L_Plain_Data.req.

NOTE 2 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.

The L_Raw.req service primitive shall be used for sending a frame in raw format onto any of the given KNX media, see [Table D.20](#). This message can be used, for example 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 — L_Raw.con

Message code	Additional info length	Additional information	Raw frame on medium
MC	AddIL	...	Data
2Fh

The L_Raw.con (see [Table D.21](#)) 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 — 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 — L_Raw.ind

Message code	Additional info length	Additional information	Raw frame on medium
MC	AddIL	...	Data
2Dh

The L_Raw.ind service (see [Table D.22](#)) shall be used for receiving a frame in raw format from any of the given KNX media. This message can be used, for example 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 — L_Busmon.ind

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, see [Table D.23](#). 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.

- a) T_Data_Individual
- b) T_Data_Connected

The basic frame structure for both services shall be as specified in [Figure D.2](#) 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

- | | | |
|---------------------------------------|--|---|
| a) MC: | message code | |
| b) AddIL: | length of additional information | |
| c) A d d i t i o n a l
information | This shall contain one or more additional information fields according the value of the field additional information length. | |
| d) Reserved | This field is reserved and shall be filled with 000000000000h.
NOTE 1 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. | |
| e) 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. | |
| f) 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. | |
| g) 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

The message flow for T_Data_Individual is shown in [Figure D.3](#).

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

The message flow for T_Data_Connected is shown in [Figure D.6](#).

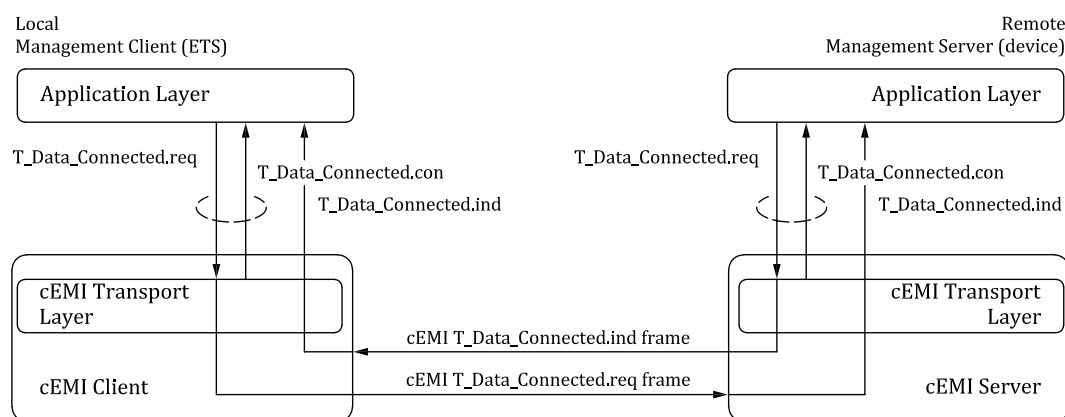


Figure D.6 — Message flow for T_Data_Connected

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, see [Figure D.7](#).

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, see [Figure D.8](#).

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 General

This subclause 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 the 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 service primitive containing information about the success of the request.

Messages generated by the cEMI server, for example 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) a cEMI client sends a new request only when the confirmation of the preceding request is received, or
- b) 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, for example 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

The basic message structure for data properties is shown in [Table D.24](#).

Table D.24 — Basic message structure for data properties

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 (see [Table D.25](#)) 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, see [Table D.26](#). This response shall contain the requested number of elements beginning at the requested start index within the requested property of the addressed interface object, see [Table D.27](#).

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, for example 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. The M_PropRead.con message, negative response is shown in [Table D.28](#).

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 (see [Table D.29](#)) 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, see [Table D.30](#). 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, for example 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. The M_PropWrite.con message, negative response is shown in [Table D.31](#).

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, for example 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, see [Table D.32](#).

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

— 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.

— Error code set:

The error code set uses an 8 bit enumeration data type (N_8), see [Table D.33](#).

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 a non existing property	R/W

Table D.33 *(continued)*

Error code	Error type	Description	Service
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 at this moment cannot 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.

— **cEMI server behaviour after reception of a management service request**

- a) If only the minimum requirements concerning are supported, the following applies.
 - i) If any problem is detected, the access shall be confirmed by the negative response 'Unspecified Error', else continue with ii).
 - ii) Send a positive service confirmation.
- b) If a more sophisticated error handling is supported, providing more differentiating error identification, the following applies.
 - i) 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 ii).
 - ii) If the array field(s) within the accessed (read or write) property causes a problem, for example too many elements are accessed, the access shall be confirmed with the negative response "Property Index/Range Error"; otherwise continue with iii) for write, respectively viii) for read access.
 - iii) 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 iv).
 - iv) 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 v).
 - v) If the data (Point) type of the received property value does not conform to 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 vi).
 - vi) 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 vii).
 - vii) 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).
 - viii) 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 (see [Figure D.9](#)) 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, see [Figure D.10](#). 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 (see [Figure D.11](#)) 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 Type and an object instance number, the property shall be addressed with the Property Identifier (PID). The contents of the data part depend 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, see [Figure D.12](#).

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 (see [Table D.34](#)) shall be used to restart the cEMI server device on initiation by the cEMI client. The cEMI server device shall execute a basic restart as on the reception of an A_Restart. To perform a basic restart the management server shall:

- close all KNX transport layer connections;
- close all KNXnet/IP connections (device management, tunnelling or other);
- close all KNX secure session;
- close all KNX TCP connections;
- apply changed configuration parameters at the latest 30 s after completing the restart.

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 (see [Table D.35](#)) 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;
- b) a bus power down-/up cycle, only in case the cEMI server device is powered from the bus;
- c) 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:

- d) M_Reset.ind is not mandatory for a USB powered device due to reasons on USB protocol/management level.
- e) 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 specification does not include automatic reconnecting to the KNXnet/IP client by the KNXnet/IP server.

NOTE 1 An M_Reset.con service is not applicable as the response/confirmation after an 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 General

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, for example 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.

- a) Obtaining information from the connected device, for example individual address, domain address, maximum APDU-Length, connected medium, state of the device, possible communication modes of the device.
- b) Setting of the individual address.
- c) Setting of the domain address.
- d) Setting of the device's communication layer mode; for example raw data (Busmonitor mode), data link layer, transport layer.

D.1.2.2 Generic management based on interface objects

D.1.2.2.1 General

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 listed in the [Table D.36](#) below.

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 A tool needs to be able 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, see [Table D.37](#).

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
PID_MAX_INTERFACE_APDU_LENGTH	68	APDU length for bus access

D.1.2.2.3.2 Communication mode (data link layer/raw/transport layer) (PID_COMM_MODE) (PID = 52)

a) 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 controls whether a telegram received from the bus is 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.

The interpretation of DPT_CommMode of PID_COMM_MODE is shown in [Table D.38](#).

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

b) 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.

c) 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 General

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 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

E.2.1 General

A general overview of the device object Property Identifiers (PID) is shown in [Table E.1](#).

Table E.1 — device object Property Identifiers (PID)

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.2.2 PID_SERVICE_CONTROL (PID = 8)

The property PID_SERVICE_CONTROL shall be a permanent control field of the device, see [Table E.2](#).

Table E.2 — PID_SERVICE_CONTROL

Bit	Name	Description	Coding	Default
0	USER_STOPPED_INFO_EN	User stopped serviceInfo enable	not used	—
1	OWNINDIVIDUAL_ADDRESS_RECEIVED_EN	Enables 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.2.3 PID_DEVICE_CONTROL (PID = 14)

The property PID_DEVICE_CONTROL shall be a temporary control field of the device, see [Table E.3](#).

Table E.3 — PID_DEVICE_CONTROL

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.2.4 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.2.5 PID_ERROR_FLAGS (PID = 53)

PID_ERROR_FLAGS are shown in [Table E.4](#).

Table E.4 — PID_ERROR_FLAGS

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		shall be 1

Table E.4 (continued)

Bit	Name	Description	Coding	Default
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.2.6 PID_PROGMode (PID = 54)

The PID_PROGMode is shown in [Table E.5](#).

Table E.5 — PID_PROGMode

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.

A general overview of the router object Property Identifiers (PID) is shown in [Table E.6](#).

Table E.6 — Router object Property Identifiers (PID)

Property ID	Access R/W	Access via NP services	Req.	Value
1 = PID_OBJECT_TYPE	3/-	X	M	ROUTER_OBJECT: 0006h
5 = PID_LOAD_STATE_CONTROL	3/0		M	See E.3.2 "PID_LOAD_STATE_CONTROL (PID = 5)"
51 = PID_LINE_STATUS	3/-	X	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
Access shall also be possible via property services for PID = 1 and PID = 51.				

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, see [Table E.7](#):

Table E.7 — Supported load controls

Load control	Coded
EV_NOP	00h
EV_START_LOAD	01h
EV_LOAD_COMPLETE	02h
EV_UNLOAD	04h

The following load states may be returned, see [Table E.8](#):

Table E.8 — Returned load states

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, see [Table E.9](#).

Table E.9 — Status of the subline

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 = GroupAddressFilterObject 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, see [Table E.10](#).

Table E.10 — Properties affecting the handling of frames

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), see [Table E.11](#).

Table E.11 — Properties affecting the handling of group addressed frames

Bit	Name	Description	Coding	Default
0–1	GROUP_6FFF	Specify the handling of group addressed frames $\leq 6FFFh$.	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 $\geq 7000h$.	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. For the structure of the property function for reading or writing, see [Table E.12](#).

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)

The SRVID_CLEAR_ROUTINGTABLE is shown in [Table E.13](#).

Table E.13 — SRVID_CLEAR_ROUTINGTABLE

Octet:	10	11
	00h	01h

No additional information are used.

- Used with A_FunctionPropertyCommand-PDU: function clears all addresses in the filter table.
- [Table E.14](#) – 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)

The SRVID_SET_ROUTINGTABLE is shown in [Table E.15](#).

Table E.15 — SRVID_SET_ROUTINGTABLE

Octet:	10	11
	00h	02h

No additional information 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.

The response A_PropertyValue_Response PDU is shown in [Table E.16](#).

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)

The SRVID_CLEAR_GROUPADDRESS is shown in [Table E.17](#).

Table E.17 — SRVID_CLEAR_GROUPADDRESS

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 are 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 are equal.
 - If START ADDRESS is greater than END ADDRESS no address will be checked.

The response A_PropertyValue_Response PDU is shown in Table E.18.

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)

The SRVID_SET_GROUPADDRESS is shown in [Table E.19](#).

Table E.19 — SRVID_SET_GROUPADDRESS

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 are 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 are equal.
 - If START ADDRESS is greater than END ADDRESS no address will be checked.

The response A_PropertyValue_Response PDU is shown in [Table E.20](#).

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 set;
 - return_code 00h means all tested addresses are successfully set.

E.3.6.6 Error handling

The reaction to an unknown service ID is defined as follows in [Table E.21](#):

Table E.21 — Reaction to an unknown service ID

Octet:	10	11	...
	00h	undefined	...

- used within A_FunctionPropertyCommand-PDU or A_PropertyValue_Read-PDU.

The response A_PropertyValue_Response PDU is shown in [Table E.22](#).

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)

The PID_COUPL_SERV_CONTROL is shown in Table E.23.

Table E.23 — PID_COUPL_SERV_CONTROL

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 if 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;
 - a) 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;
 - b) 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, <https://www.ietf.org/rfc.html>
- [2] POSTEL J., “Transmission Control Protocol”, STD 7, RFC 793, USC Information Sciences Institute, September 1981, <https://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, <https://www.ietf.org/rfc.html>
- [5] DROMS R., “Dynamic Host Configuration Protocol”, RFC 2131, Bucknell University, March 1997 , <https://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, *Information technology — 8-bit single-byte coded graphic character sets — Part 1: Latin alphabet No. 1*
- [8] ISO 16484-5, *Building automation and control systems (BACS) — Part 5: Data communication protocol*

