



BSI Standards Publication

Home and Building Electronic Systems (HBES)

Part 4-4: HBES IoT Point API

National foreword

This British Standard is the UK implementation of EN 50090-4-4:2025.

The UK participation in its preparation was entrusted to Technical Committee IST/6/-/12, Home Electronic Systems.

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

Contractual and legal considerations

This publication has been prepared in good faith, however no representation, warranty, assurance or undertaking (express or implied) is or will be made, and no responsibility or liability is or will be accepted by BSI in relation to the adequacy, accuracy, completeness or reasonableness of this publication. All and any such responsibility and liability is expressly disclaimed to the full extent permitted by the law.

This publication is provided as is, and is to be used at the recipient's own risk.

The recipient is advised to consider seeking professional guidance with respect to its use of this publication.

This publication is not intended to constitute a contract. Users are responsible for its correct application.

© The British Standards Institution 2025
Published by BSI Standards Limited 2025

ISBN 978 0 539 25948 3

ICS 35.240.67; 97.120

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

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 June 2025.

Amendments/corrigenda issued since publication

Date	Text affected
------	---------------

English Version

Home and Building Electronic Systems (HBES) - Part 4-4: HBES IoT Point API

Systèmes électroniques pour les foyers domestiques et les
bâtiments (HBES) - Partie 4-4: API de Point IoT HBES

Elektrische Systemtechnik für Heim und Gebäude (ESHG) -
Teil 4-4: ESHG IoT Point API

This European Standard was approved by CENELEC on 2025-04-16. CENELEC 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 CENELEC 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 CENELEC member into its own language and notified to the CEN-CENELEC Management Centre has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Croatia, Cyprus, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Republic of North Macedonia, Romania, Serbia, Slovakia, Slovenia, Spain, Sweden, Switzerland, Türkiye and the United Kingdom.



European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

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

Contents

Page

European foreword.....	4
1 Scope	5
2 Normative references	5
3 Terms, definitions and abbreviations.....	6
3.1 Terms and definitions	6
3.2 Abbreviations.....	11
4 HBES IoT Point API	13
4.1 Introduction	13
4.2 System entities	14
4.3 Device Model.....	15
4.4 Conventions used in this document.....	17
5 Point API Standard	17
5.1 Application Protocol.....	17
5.2 Overview	17
5.3 System Design	20
5.4 Device Bootstrapping and Configuration	23
5.5 Resource Model	27
5.6 Runtime Interworking.....	75
6 Security.....	100
6.1 Introduction	100
6.2 Device Identity Enrollment.....	100
6.3 Device Identity Certificates.....	105
6.4 Certificate Validation	107
6.5 Device Access Control.....	108
6.6 OSCORE Application Layer Security.....	115
7 Software Update	136
7.1 Introduction	136
7.2 Software Update Client Resource (swu)	137
7.3 Software Update Modes	141
8 Profiles.....	151
8.1 HBES IoT Point API Device.....	151
8.2 CBOR Encoding.....	157
9 Examples	158
9.1 Device point list examples.....	158
9.2 Device configuration example.....	162
9.3 Data encryption/decryption example.....	170
10 HBES IoT Router	173
10.1 Introduction	173
10.2 Conformance.....	174
10.3 Number Format	174
10.4 Uniform Resource Identifiers	174
10.5 Uniform Resource Name	174
10.6 HBES IoT Router Specification	174
10.7 Runtime Interworking.....	182
10.8 Profiles.....	184

10.9 Security 187

10.10 Examples 187

Bibliography.....190

European foreword

This document (EN 50090-4-4:2025) has been prepared by CLC/TC 205 “Home and Building Electronic Systems (HBES)”.

The following dates are fixed:

- latest date by which this document has to be (dop) 2026-05-31
implemented at national level by publication of
an identical national standard or by
endorsement
- latest date by which the national standards (dow) 2028-05-31
conflicting with this document have to be
withdrawn

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

Any feedback and questions on this document should be directed to the users' national committee. A complete listing of these bodies can be found on the CENELEC website.

1 Scope

This document lays down the requirements for the HBES Point API extension to the EN 50090 series, allowing vendor independent communication between smart home and building devices on IPv6 networks.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

EN 50090-1:2011, *Home and Building Electronic Systems (HBES) - Part 1: Standardization structure*

EN 50090-3-3, *Home and Building Electronic Systems (HBES) - Part 3-3: Aspects of application - HBES Interworking model and common HBES data types*

EN 50090-4-1, *Home and Building Electronic Systems (HBES) - Part 4-1: Media independent layers - Application layer for HBES Class 1*

EN 50090-4-2, *Home and Building Electronic Systems (HBES) - Part 4-2: Media independent layers - Transport layer, network layer and general parts of data link layer for HBES Class 1*

EN 50090-7-1, *Home and Building Electronic Systems (HBES) - Part 7-1: System management - Management procedures*

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

RFC 7252, *The Constrained Application Protocol (CoAP)*

RFC 8949, *Concise Binary Object Representation (CBOR)*

RFC 6838, *Media Type Specifications and Registration Procedures*

RFC 6690, *Constrained RESTful Environments (CoRE) Link Format*

RFC 1035, *Domain names – Implementation and specification*

RFC 8323, *CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets*

RFC 4291, *IP Version 6 Addressing Architecture*

RFC 6763, *DNS-Based Service Discovery*

RFC 8766, *Discovery Proxy for Multicast DNS-Based Service Discovery*

RFC 6762, *Multicast DNS*

RFC 3596, *DNS Extensions to Support IP Version 6*

RFC 8613, *Object Security for Constrained RESTful Environments (OSCORE)*

RFC 7959, *Block-Wise Transfers in the Constrained Application Protocol (CoAP)*

RFC 9175, *Constrained Application Protocol (CoAP): Echo, Request-Tag, and Token Processing*

RFC 8516, *“Too Many Requests” Response Code for the Constrained Application Protocol*

RFC 3306, *Unicast-Prefix-based IPv6 Multicast Addresses*

RFC 3307, *Allocation Guidelines for IPv6 Multicast Addresses*

RFC 6282, *Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*

RFC 9148, *EST-coaps: Enrollment over Secure Transport with the Secure Constrained Application Protocol*

RFC 8995, *Bootstrapping Remote Secure Key Infrastructure (BRSKI)*

RFC 5967, *The application/pkcs10 Media Type*

RFC 5273, *Certificate Management over CMS (CMC): Transport Protocols*

RFC 5280, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*

RFC 2818, *HTTP Over TLS*

RFC 7251, *AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS*

RFC 8392, *CBOR Web Token (CWT)*

RFC 3986, *Uniform Resource Identifier (URI): Generic Syntax*

RFC 8747, *Proof-of-Possession Key Semantics for CBOR Web Tokens (CWTs)*

RFC 8152, *CBOR Object Signing and Encryption (COSE)*

RFC 3339, *Date and Time on the Internet: Timestamps*

RFC 6335, *Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry*

RFC 4492, *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)*

RFC 5869, *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in EN 50090-1:2011 and the following apply.

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

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

3.1.1

Actuator

Point performing an actuation in HBES IoT (executed by a specific procedure, with an expected result) that changes an Installation state during Runtime

3.1.2**Advanced Message Queuing Protocol**

open standard application layer protocol for message-oriented middleware with defining features such as message orientation, queuing, routing (including point-to-point and publish-and-subscribe), reliability and security

3.1.3**Application Function**

set of Functions used to achieve the desired behavior of a technical system, typically using a combination of devices exchanging information via their input and output Datapoints

Note 1 to entry: An Application Function may be split into several Functional Blocks (located in one or more devices) with their input and output Datapoints that are logically connected to each other.

EXAMPLE “direct electrical heating”, “electrical heating with accumulators”, “warm water heating”, “fan coil air-conditioning” ...

3.1.4**Authorization and Group Manager**

entity service that supports an authorized access so that a device can join to a specific communication channel

3.1.5**Channel**

collection of Datapoints of a device that are logically related to each other typically by association with a hardware feature or a specific function of that device

Note 1 to entry: These Datapoints may be derived from one or more defined Functional Blocks (as defined by KNXA) or may be an expansion above and beyond defined Functional Blocks or may be independent of a NX Functional Block if none is defined for the function associated with the channel.

3.1.6**Datapoint**

representation of a logical input entity of a device acting as recipient of Installation state data, whereas a logical output of a device acts as source of Installation state data

3.1.7**Device**

physical element that is part of the network and an object a customer can buy

3.1.8**Domain CA**

entity that issues operational certificates for the domain

3.1.9**Endpoint**

interface to a service, a process, or a queue or topic destination in service-oriented architecture

3.1.10**Function**

part of the intended behavior of a Functional Block in a building context

3.1.11**Functional Block**

one or more Functions that belong together, that cannot be separated across two devices but is big enough that a device with only one such entity could be marketed and that has a well-defined black box behavior

3.1.12

Function Point

runtime system state information of a specific Application Function shared by at least two datapoints and having a unique identifier that addresses a group of controlled objects called a Group Address

EXAMPLE < Light Switch > in room living on/off, whereas the < ... > is the Function Point name

3.1.13

Group Address

numerical identifier of a Function Point

3.1.14

Group Communication

communication model in which one sender communicates information to one and typically more receivers

Note 1 to entry: In HBES IoT, this can be realized by simple UDP communication or by using a message broker system or other.

Note 2 to entry: In non-IoT HBES this is referred to as multicast or P2P based Group Communication, either with Group Addresses or Interface Objects (Points) exchanging state data

3.1.15

Group Message

message exchanged between Group Objects using a specific group identifier, not necessarily expressing any type of IP unicast or IP multicast communication pattern

3.1.16

Group Object

preferably foreseen for Group Communication using Group Address(es) - becoming a member of a Function Point represented by the assigned Group Address – or accessible via P2P communication, if no Group Address is assigned

3.1.17

HBES Installation ID

ID used to separate HBES Installations from each other, specified as a random, unique ID

3.1.18

HBES IoT

protocol suite/framework for transport of HBES data on the Internet of Things with IPv6

3.1.19

HBES IoT 3rd Party API

the set of requirements and regulations through which partial access to an Installation can be gained by offering a collection of Endpoints

Note 1 to entry: It offers an access at the level of the Installation and supports more sophisticated queries to (history) values of installation state data or specific elements of the Installation, such as location, Application Function and Datapoints.

3.1.20

HBES IoT Router

gateway between HBES IoT and non-IoT HBES

3.1.21

HBES IoT Point API

set of requirements and regulations through which devices directly exchange information with each other based on HBES IoT

3.1.22**HBES System**

system which encompasses HBES standards and definitions, allowing the creation of an Installation, and including aspects such as topology constraints for devices, device configuration procedures and runtime interworking principles, Functional Blocks, with Application behavior specified in FBs and more

3.1.23**Installation**

assembly of materials and components (devices) placed in position to provide a service, a deployed system consisting of equipment and Functions that are used for a particular purpose

EXAMPLE A deployed Installation may be a HVAC system or a fire protection system.

3.1.24**Management Client**

means to configure and commission Devices, as well as to plan, design and diagnose an entire Installation

Note 1 to entry: Is also responsible to write specific configuration data such as Device parameters or group tables to the Devices.

3.1.25**MaC Project**

Project created by a Management Client documenting the Configuration of an Installation

3.1.26**Message Broker**

entity that is receiving messages from publishers and providing it to interested subscribers, the defining characteristic is that the broker itself is a discrete service

3.1.27**MQTT**

Message Queuing Telemetry Transport publish-subscribe-based messaging Protocol, standardized as ISO/IEC 20922

3.1.28**Non-IoT HBES**

HBES TP, RF, PL and KNXnet/IP protocol for transport of HBES data, in this document colloquially also used as synonym for an HBES System without any HBES IoT Devices

3.1.29**Ontology**

conceptual descriptions of things that have a real-world commonality sharing the knowledge of a domain, mainly expressed with OWL

Note 1 to entry: Are a structured way to describe the meaning of data in ontology classes and should not be mixed up with common data model structures.

3.1.30**OWL**

Web Ontology Language, informally OWL 2, specified by the World Wide Web Consortium (W3C), mainly serialized with XML syntax for RDF (RDF/XML)

Note 1 to entry: In this specification the abbreviation OWL is always an explicit reference to OWL 2.

3.1.31**Point**

represents an interface to data in the system

Note 1 to entry: In this document the term Point is used as an umbrella for data that can be accessed from outside the Device, for instance to interact with other Points from other Devices, hence the term is a generic superset of the term Datapoint (describing more precisely the technics how the “data” in the system are structured and/or coded).

3.1.32

Point API

simple RESTful (CoAP or HTTP) application programming interface designed for, but not limited to, constrained class 2 devices [RFC 7228] supporting device individualization, device linking and accessing device runtime data (e.g., Functional Block or Channel Datapoints)

3.1.33

Publisher

entity that is sending messages to a Message Broker

3.1.34

Recipient

entity that is receiving messages from a Publisher

Note 1 to entry: If the Recipient is not Subscriber at the same time, then the Recipient endpoint needs to be a fixed configuration in the Publisher group table.

3.1.35

RDF

framework to represent information in the web by using triples, which can be serialized and stored in many formats

Note 1 to entry: Formats such as the TURTLE or JSON(-LD) are described under <https://www.w3.org/TR/rdf11-concepts/>

3.1.36

Registrar

entity that is a service representative of a certain domain, configured to decide whether a new device is allowed to join the domain

3.1.37

Runtime

process-to-process communication of data between Devices, opposing to Configuration

Note 1 to entry: Concerns mainly the communication of Datapoint values (control and status information).

3.1.38

Security Zone

group of devices that all make use of the same Trust Anchor

3.1.39

Sensor

Point in HBES IoT, performing an observation (executed by a specific procedure, triggered by a stimulus), responding a result as an Installation state during Runtime

3.1.40

Subscriber

HBES IoT device receiving messages from a Message Broker

3.1.41

Tag

kind of annotation term used to extend available data with (in most cases) well known standardized information from a dictionary (in contrast to user defined, arbitrary term)

3.1.42**Thing Description**

semantic metadata model to describe (abstract or physical) things, as specified by the thing description <https://www.w3.org/TR/wot-thing-description/> and thing Ontology <https://www.w3.org/2019/wot/td>

3.1.43**Trust Anchor**

authoritative entity for which trust is assumed and not derived (e.g. an X.509 root certificate)

3.1.44**X.509**

certificate format

3.2 Abbreviations

For the purposes of this document, the following abbreviations apply.

AEAD	Authenticated Encryption with Additional Data
AL	Application Layer
AMQP	Advanced Message Queuing Protocol
AP	Access Point
API	Application Programming Interface
BLE	Bluetooth Low Energy
BRSKI	Bootstrapping Remote Secure Key Infrastructure
C	Conditional
CBOR	Concise Binary Object Representation
CoRE	Constraint RESTful Environments
COSE	CBOR Object Signing and Encryption
CRL	Certificate Revocation List
CSR	Certificate Signing Request
CWT	CBOR Web Token
DAD	Duplicate Address Detection
DNS	Domain Name System
DNS-SD	Domain Name System Service Discovery
DB	Database
DER	Distinguished Encoding Rules
DTLS	Datagram Transport Layer Security
EST	Enrollment over Secure Transport
FQDN	Fully Qualified Domain Name
GA	Group Address
GO	Group Object
FB	Functional Block
FP	Function Point
HBES	Home and Building Electronic Systems
HKDF	Hash-based Key Derivation Function

HMAC	Hash-Based Message Authentication Code
IANA	Internet Assigned Numbers Authority
IOO	Info On off
IO	Input Output
IP	Internet Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
KDC	Key Distribution Center
KDF	Key Derivation Function
KNXA	KNX Association
KDC	Key Distribution Center
LLA	Link Local Address
LRI	Logical Resource Identifier
LSM	Load State Machine
LTE	Logical Tag Extended
M	Mandatory
MaC	Management Client
MAC	media access control address
MQTT	Message Queuing Telemetry Transport
NB	Narrow Band
NFC	Near Field Communication
O	Optional
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OSCORE	Object Security for Constrained RESTful Environments
OSV	Out of Service
OT	Operational Technology
PAKE	Password Authenticated Key Exchange
PASE	Password Authenticated Session Establishment
PBKDF	Password-based Key Derivation Function
PSK	Pre-shared Key
PTR	Pointer
QR	Quick Response
RD	Resource Directory
RDF	Resource Description Framework
RT	Resource Type
SAN	Subject Alternative Name
S-Mode	System Mode
SP	Sleep Period

SRP	Service Registration Protocol
SSM	Source-Specific Multicast
TCP	Transport Control Protocol
TD	Thing Description
TOFU	Trust on First Use
UDP	User Datagram Protocol
ULA	Unique Local Address
URN	Uniform Resource Name
URI	Uniform Resource Identifier
REST	Representational State Transfer
W3C	World Wide Web Consortium
WoT	Web of Things
WPAN	Wireless Personal Area Network

4 HBES IoT Point API

4.1 Introduction

The HBES System Architecture is given in Figure 1.

HBES IoT uses Internet Protocol (IP) suite standards for the transmission of HBES IoT application layer data across IP networks.

Physical media like Ethernet (IEEE 802.3), Wi-Fi (802.11), or WPAN (802.15.4) carry HBES IoT packets. These may contain unicast TCP or UDP frames or multicast UDP frame transmission of HBES IoT application data. HBES IoT application data is agnostic to the underlying communication layers. Hence, it is possible to send HBES IoT messages over non-IP transport bindings such as NB IoT, or BLE. However, this is out-of-scope of this document.

A typical (IP-based) interworking infrastructure allows heterogeneous data link media to work seamlessly with each other. The Point API maps HBES AL data to a RESTful resource model, and CBOR/JSON-based data representation is used to communicate over IP. Using this API, a client can read/write values or subscribe to Point events (e.g., switch on/off). The HBES IoT Point API is based on the following building blocks:

- **Discovery**

Discovery (of resources, including devices) can be made with unicast or multicast. Resource discovery in CoAP (CoRE) is accomplished using a `"/.well-known/core"` resource URI that returns a list of links about resources (e.g., *Functional Block* Properties) hosted by that server that matches filter attributes.

- **S-Mode Messaging**

The S-Mode messaging uses a secure message-oriented communication pattern for group communication where a producer sends a message to notify consumers of a change in the domain. A tool, or rather a Management Client (MaC), configures group communication events via group tables.

- **Point Read, Write, and Publish/Subscribe**

Parameter and diagnostic Properties are used for sensor, actuator, parameter, and diagnostic values, such as getting the current sensor value or setting a setpoint. They are addressed by URIs, can be directly accessed with the corresponding standard CoAP access method GET (read values), and can be manipulated with PUT/POST (write values). Additionally, also subscribing to Property values is possible.

• Security

Group communication and access to the parameter and diagnostic Properties are secured by OSCORE or (D)TLS. OSCORE (application layer security) is used for S-Mode messaging, Point read, write, and publish/subscribe. (D)TLS (transport layer security) is mainly used for mutually authenticated pre-shared key distribution. The initial bootstrapping of pre-shared OSCORE keys and operational device certificates bases on an out-of-band (QR code, NFC, or BLE, etc.) authentication code as a proof of possession.

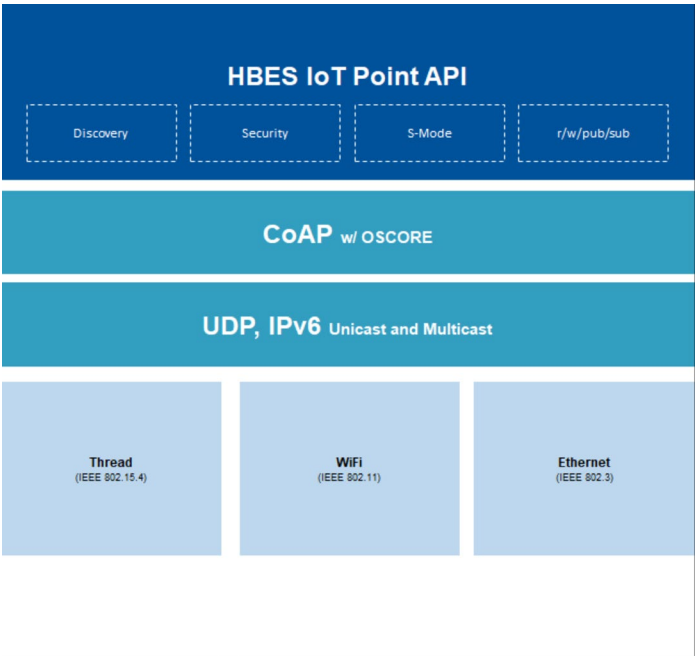


Figure 1 — HBES IoT Point API

4.2 System entities

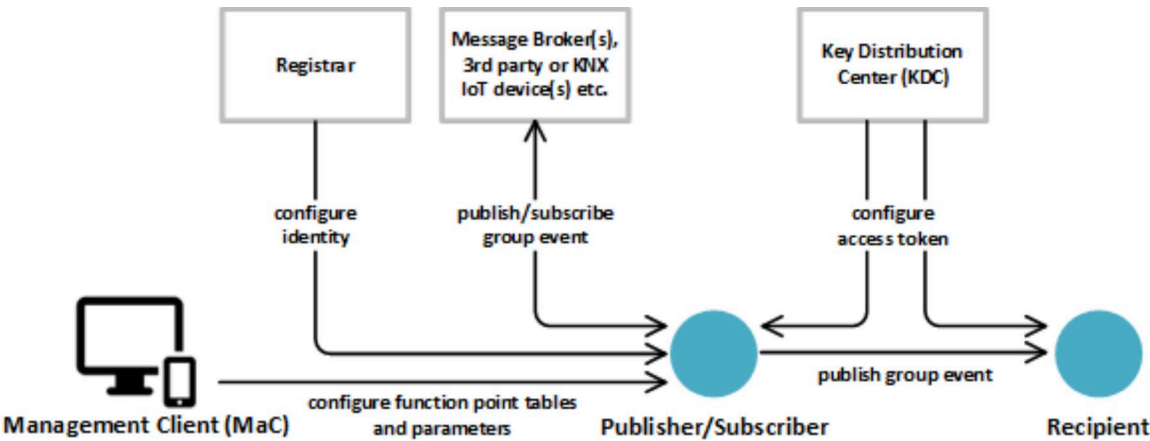


Figure 2 — System entities

The **Key Distribution Center (KDC)** enables and enforces the authorized access of joining HBES IoT devices to access the related KNX Group Communication. Multiple installations can be associated with the same Authorization and Group Manager, which might be a service entity part of a MaC (Management Client) or an independent service.

The **Message Broker** is an intermediary that can provide data marshaling, routing, message translation, and persistence. It receives messages from a Publisher and delivers them to all related Recipients. An HBES Classic to HBES IoT Router (see also Clause 10) is an example of a Message Broker.

Publishers and **Subscribers** are HBES IoT devices that are loosely coupled and often do not know each other. Their primary relationship is that they are configured with the same *Group Address*. In HBES IoT:

- the Publisher sends Group Messages to Subscribers (e.g., via Message Broker). It represents a particular information source, for example, a light switch status;
- the Subscriber is the consumer of Group Messages from a Publisher. They are CoAP Clients but are Group Message Recipients;
- the Publishers and Subscribers use CoAP to communicate;
- A Recipient is a device that receives Group Messages from a Publisher. If the Recipient is not a Subscriber simultaneously, then the Recipient endpoint is a static configuration in the Publisher Function Point table.

The **Registrar** decides whether a new HBES IoT device can join the domain; it:

- might be a part of the MaC (Management Client), containing a mixture of Backlist rules, Whitelists (for known installed devices), and stateful tracking to protect the HBES System;
- subsequently configures domain CA certificates followed by the operational device certificate on HBES IoT devices.

An overview of the system entities is given in Figure 2.

4.3 Device Model

Figure 3 depicts the HBES IoT Device Model, which is, from a high-level perspective, the same as the classic HBES Device Model.

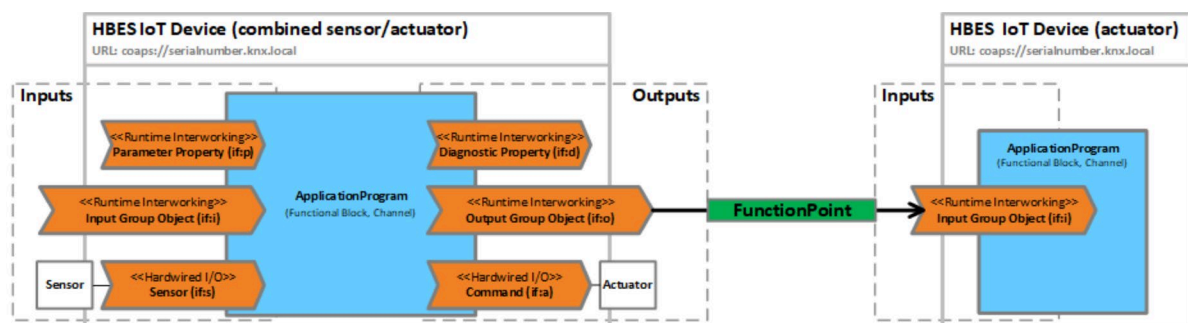


Figure 3 — HBES IoT Device Model

An HBES IoT device may have physical inputs and/or outputs.

- A physical input may be internal to the device or have an external sensor connected via a terminal block.
- A physical output may be internal to the device or have an external actuator connected via a terminal.

An HBES IoT device has at least one logical input and/or output. Such a logical input or output is called *Group Object (GO)*. The *Group Object* has a unique identifier with reference to the device. When a *Group Address* is assigned to a *Group Object*, the *Group Object* becomes a member of that *Function Point* identified by the *Group Address*. A *Group Address* is the instantiation of a *Function Point* and is unique in an installation.

An input *Group Object* can be assigned to one or more *Function Points*.

An output *Group Object* can be assigned to one or more *Function Points* but can only send information to one *Function Point*.

For configuration purposes, an HBES IoT device may have parameters to determine specific behavior concerning the whole device or a device Channel. Each Parameter Property has a unique identifier with reference to the device.

For diagnostic purposes, an HBES IoT device may provide diagnostic information concerning the whole device or a device Channel. Each Diagnostic Property has a unique identifier with reference to the device.

Parameter and diagnostic Properties use the same communication mechanisms and are not differentiated, and both are named Property in the following.

A Group Object is a specialization of a Property. A Group Object is designed for runtime communication in a Brokerless or Broker-based system (see 5.3). A simple property is only designed and intended for configuration or diagnostic purposes, and it supports only simple point-to-point communication methods (read-, write- and subscription-commands).

An HBES IoT device may have *Group Objects* and parameter and diagnostic Properties that belong together. Where *Group Objects* and Properties belong together, these may be declared as a Channel.

A device may have Channels with different sets of Group Objects and Properties and/or Channels that have identical sets of Group Objects and Properties. Devices may have one or more Channels that are functionally identical and have the same type of Group Objects and Properties.

Apart from Channels, an HBES IoT device may have *Group Objects*, parameter, and diagnostic Properties that apply to the whole device. These are associated with the control function of the whole device.

If and how *Group Objects* and *Properties* are declared as a Channel is up to the manufacturer designing the device.

A Channel may be declared to include at least one *Functional Block* describing the function of that Channel (see EN 50090-6-2).

Functional Blocks for different applications are defined by KNXA. *Functional Blocks* encompass inputs and outputs as well as parameter properties with generic names and identifiers.

The *Functional Blocks* determine the Interworking between products of different manufacturers. More than one *Functional Block* type may be required to describe the functionality of a Channel.

An HBES IoT device has a unique identifier called *Individual Address*. The *Individual Address* is assigned and is unique with respect to the project.

An HBES IoT device shall have a globally unique identifier assigned by the manufacturer called *Serial Number*.

The hardware (MAC-) or network (IP-) address of the device shall not be used as *Individual Address*.

Table 1 summarizes the above-mentioned core elements and their associated unique identifiers.

Table 1 — Core elements and identifiers

Core element	Unique Identifier	
	Name	Scope
Device	Serial Number	globally unique
	Individual Address	unique within a project
Functional Block	Functional Block ID	unique per functionality
Function Point	Group Address	unique within a project
Group Object (GO)	GO identifier	unique per device
Parameter / Diagnostic Property	Property identifier	unique per device

4.4 Conventions used in this document

4.4.1 Conformance

For all resources, their expected request/ response document formats and content are not described to the full extent in this document. Whether an element is mandatory or optional can be found in the JSON-schema description as an electronic document. The URL <https://schema.knx.org> allows retrieval of the most recent electronic documents of the HBES IoT Point API. Older versions are also available.

4.4.2 Number Format

All numbers in this document are assumed to be decimal unless indicated with a prefix. Hexadecimal numbers are prefixed with the designation "0x", and binary numbers are prefixed with the designation "0b". Binary numbers are defined as successive groups of 4 bits, separated by a space character from the most significant bit to the least significant bit (0b1111 1111). Byte strings are notated in one of the base encodings, without padding, enclosed in single quotes, prefixed by >h< for base16, >b32< for base32, >h32< for base32hex, >b64< for base64. For example, the byte string 0x12345678 could be written as h'12345678'.

4.4.3 Uniform Resource Identifiers

A Uniform Resource Identifier (URI) identifies a logical resource. This document uses the CoAP URL for link formats which comprises scheme, authority, path, and query values: scheme ":" ["/" authority] path ["?" query]. The authority part shall contain either an FQDN or an IP address.

The following link formats are used in this document:

- Absolute Link with schema and authority: coap://{ip-address or fqdn}:{port}/base-path/path1?query
- Absolute path that starts at the root path ("/"): /base-path/path?query
- Relative path that starts after a base path (without "/" at the beginning): path?query

4.4.4 Uniform Resource Name

A Uniform Resource Name (URN) identifies a logical resource in a permanent way, even after that resource does not exist anymore. This document uses the URN format as defined in RFC 2141, which comprises the namespace identifier "knx" and namespace-specific identifiers: "urn:knx" [":" namespace-specific-identifier].

The following URN formats are used in this document:

- URN with the namespace identifier: urn:knx:namespace-specific-identifier;
- Truncated URN without namespace identifier (with ":" at the beginning): :namespace-specific-identifier.

5 Point API Standard

5.1 Application Protocol

This version of the HBES IoT Point API standard solely considers the use of CoAP as an application protocol. The below standards and examples, therefore, focus on CoAP. Indications for other application protocols may be added in future versions.

5.2 Overview

5.2.1 Common Data Model

The following clauses describe a standard data model and API for HBES IoT devices that represent a contract for the interworking between devices and, in addition, between devices and infrastructure (e.g., Management Client, Message Broker, etc.). For a semantic description of instances of the API (project export), HBES IoT adopts the W3C WoT interactions model for Points that exchange data via IP interworking infrastructure. The

Thing Description [TD] is a central building block in the W3C Web of Things (WoT). The TD provides a framework for semantic metadata for the device itself, an interaction model based on WoT's PropertyAffordances, ActionAffordances, and EventAffordances paradigm, a semantic schema to make data models machine-processable and features for Web Linking to express relations among devices and Points. For example, the "/.well-known/core" URI is an ActionAffordance with a default entry point for discovery purposes, or EventAffordances and PropertyAffordances are usually mapped to a Parameter and Diagnostic Property. The TD is the glue between HBES IoT device data and the domain model (see EN 50090-6-2).

An overview of the Point Interactions is given in Figure 4.

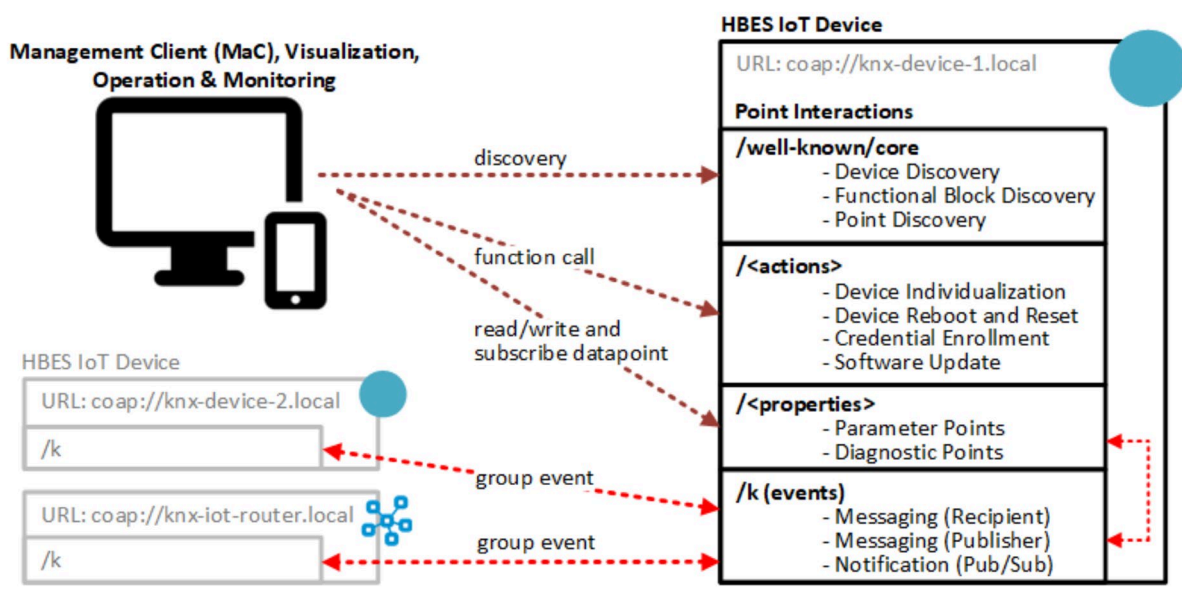


Figure 4 — Point Interactions

5.2.2 Application Layer Service Mapping

The Application layer provides many application services to the application process. Application processes in different devices interoperate by using services from the application layer. Depending on the communication mode, various application layer services are available. HBES IoT only takes over the core functionality of a subset of existing Application layer services. These service functionalities are mapped to REST calls which can be described as PropertyAffordances, ActionAffordances, or EventAffordances, for example, for a semantic project export or device descriptions (see EN 50090-6-2). The following clause explains how existing Application layer services are mapped to service discovery, PropertyAffordances, ActionAffordances, and EventAffordances in HBES IoT.

The *A_IndividualAddress_Read*- and *A_IndividualAddress_Write*-services, for example, are used to find a device within a system and to change the device address. These services are used with broadcast communication, and the communication partner is not identified in this service, i.e., the partner device is enabling its *Programming Mode*, e.g., by pressing a programming button on that device. In HBES IoT, the service is split into a discovery ActionAffordance (discovery phase: DNS or GET `coap://{ipv6-multicast}/.well-known/core?if=urn:knx:if.pm`) and a second subsequent authorized ActionAffordance call to change the device address (POST `coap://{ipv6-unicast}/.well-known/knx/ia`).

PropertyAffordances expose the internal state of a device that can be directly accessed (read) and optionally manipulated (write) on parameter and diagnostic Properties. Devices can also choose to make parameter and diagnostic Properties subscribable (observe) by pushing the new state after a change. PUT and GET on Properties are used instead of the Property services, such as *A_PropertyValue_Read* or *A_PropertyValue_Write* as described in EN 50090-4-1.

ActionAffordances offer device functions, and these functions may manipulate the internal state of a device in a way that is impossible through setting Properties. Examples are changing internal state that is not exposed as PropertyAffordances (e.g., *A_Restart-PDU*), changing multiple Properties, or changing Properties over time.

An EventAffordance describes event sources that asynchronously push messages. Hence, event-oriented runtime communication is mapped to EventAffordances in the sending device and ActionAffordances in the receiving device. Here not state, but state transitions (events) are communicated (e.g., "light switch ON"). Events may be triggered by an internal state change that is not exposed as PropertyAffordance, e.g., a push button event to switch from Off to On. POST is used for Group Objects instead of A_GroupValue services as described in EN 50090-4-1.

5.2.3 Application Protocol

This document uses CoAP [RFC 7252] as a mandatory application protocol. CoAP is a reliable transport that can preserve packet ordering and handles message duplication. However, the Point API standard is not limited to CoAP. Some device profiles may implement a CoAP and an HTTP interface. This enables an HTTP client to access data on an HBES IoT device without implementing a CoAP communication stack. However, how an HTTP request is mapped to a CoAP request and how a CoAP response is mapped back to an HTTP response is out-of-scope of this document, but an HTTP implementation should follow the guidelines described in RFC 8075.

5.2.4 Content-Format

HBES IoT device resources shall support the application/link format, the application/octet-stream format, or the Concise Binary Object Representation (CBOR) format RFC 8949. The CoAP Content-Format (RFC 8949, 9.4), used to interact with Points via a RESTful method, shall use the Internet media type RFC 6838 application/cbor (RFC 8949, 9.4). The application/link format is used to list resources such as Points, and octet/stream is used for file transfer.

HBES IoT resources may also support the JavaScript Object Notation (JSON) format RFC 8259.

The examples given in this document are shown in JSON or CBOR diagnostic notation for human readability (RFC 8949, Clause 8).

In the absence of the CoAP Accept and Content-Format options in a CoAP request or a successful CoAP response, devices shall assume the default format specified for the corresponding device resource for each of those options.

This document uses the following CBOR notation (for major type 7, the minor types used in this document are a subset only of the specified minor types RFC 8949, 3.3), see Table 2:

Table 2 — Used notations

short notation	CBOR type	major type	minor type	Notes
unsigned	unsigned integer	0		The length depends on the value.
negative	negative integer	1		The length depends on the value.
byte string	byte string	2		The length is dynamic.
text	text string	3		The length is dynamic.
[]	Array	4		The length is dynamic, as needed by the Point, i.e., a B8 with only 2 bits used will be encoded as an array of bools with length 2. In case of a homogeneous array of a specific type, this is indicated by adding the <type>*.
{}	Map	5		The length is dynamic.
<tag>(CBOR type)	Tagging	6		<tag> is a number.
<1>(unsigned)	date/time	0	1	Numerical representation of seconds relative to 1970-01-01T00:00Z in UTC time.

short notation	CBOR type	major type	minor type	Notes
bool	false/true	7	20 / 21	
float	single precision float	7	26	IEEE 754 32-bit float.

5.3 System Design

5.3.1 Events and Group Communication

Group communication uses a message-oriented communication pattern for point-to-multipoint communication where a producer sends a message to notify consumers of a change in the domain. An event is a message that informs various listeners about something that has happened. A key element of an event is that the producer potentially does not care much about the response. Often it does not expect any answer at all, or if there is a response that the producer does care about, it is indirect.

There are two concepts for distributing events in a message-oriented system: Message Broker-based Publish–Subscribe (pub/sub) and brokerless transport layer multicast (or unicast) messaging pattern. These concepts are described in the following clauses.

5.3.2 Brokerless System

The HBES IoT point-to-multipoint communication relies on the IP network infrastructure to deliver messages to one or more Recipients. IP network devices – like network routers, switches, or bridges – are used for this purpose. A transport layer multicast messaging pattern is much more efficient than sending multiple point-to-point transport layer unicast messages. Multicast allows near-simultaneous message delivery and, thereby, near-simultaneous modification of actuators on all devices in a group, providing users with a perceived effect of synchronism or simultaneity (lights ON). Multicast is only possible on top of UDP in IP networks while using a sequence number supports recognizing duplication of messages. However, IP multicasting is unreliable (unconfirmed transmission) and requires proper configuration of suitable network equipment like (managed) switches and routers. Routers need knowledge about listeners in their subsystem for efficient message filtering. Multiple transport layer unicast may be used instead of multicast for communication where simultaneity is of lower importance and/or reliability is of higher priority.

The advantages of a point-to-multipoint system include the following:

- Near-simultaneous message delivery to multiple Recipients with UDP protocol using multicast addressing.
- Only standard network infrastructure is needed.
- Messages are delivered directly without software intermediaries and, therefore, have less latency and overhead.

The Group Notification Message Keys used in Figure 5 are defined in 5.5.9.

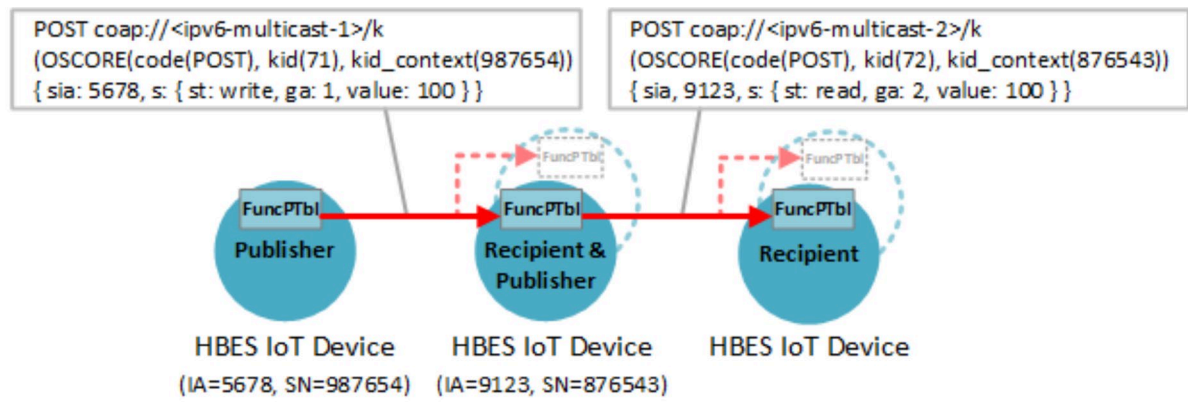


Figure 5 — Brokerless system

NOTE JSON is used to denote contents in this paper. In practice, CBOR encoding is used.

Figure 5 shows a typical brokerless communication scenario with two different group connections, where a sender (Publisher) uses multicast addressing to reach one or multiple Recipients. The device shown in the middle acts as Recipient for the 1st group and as the sender (Publisher) for another 2nd group.

The multicast IPv6 address format for point-to-multipoint communication is defined in 5.6.5.

The unicast IPv6 address format for point-to-point communication is defined in 5.6.2.

Application-level security, based on OSCORE (see 6.6), ensures message confidentiality and integrity for end-to-end security between Publisher and Recipient.

5.3.3 Message Broker-based System

Point-to-point messaging with a fixed (configured) setup or Broker-based Publish–Subscribe (pub/sub) setup is a messaging pattern for facilitating communication in loosely coupled distributed systems. It has gained increasing popularity in the context of Internet of Things (IoT) applications.

With a fixed point-to-point configuration, the serving device sends the message directly to the Recipient using unicast transport.

In most pub/sub systems, a reliable transport layer unicast communication is used. Hence, these systems rely on unicast application layer messaging (i.e., point-to-point connections from a central broker to each Subscriber) to disseminate messages. This option assumes a Message Broker in the middle. All the communication is passed through the Message Broker, and the Message Broker also routes the group notification to all subscribed applications based on a mapping table.

The advantages of this model include the following:

- Publisher and Subscriber do not have to be directly addressable. They can be anywhere if they have access to the Message Broker.
- Publisher and Subscriber can use different messaging protocols to communicate with the Message Broker (e.g., EN 50090, WebSocket, HTTP, or CoAP).
- Reliable Group Notifications (acknowledged unicast messages) can be handled to a large list of Subscribers, multiple networks, or even chained Message Brokers or scalable Message Brokers.

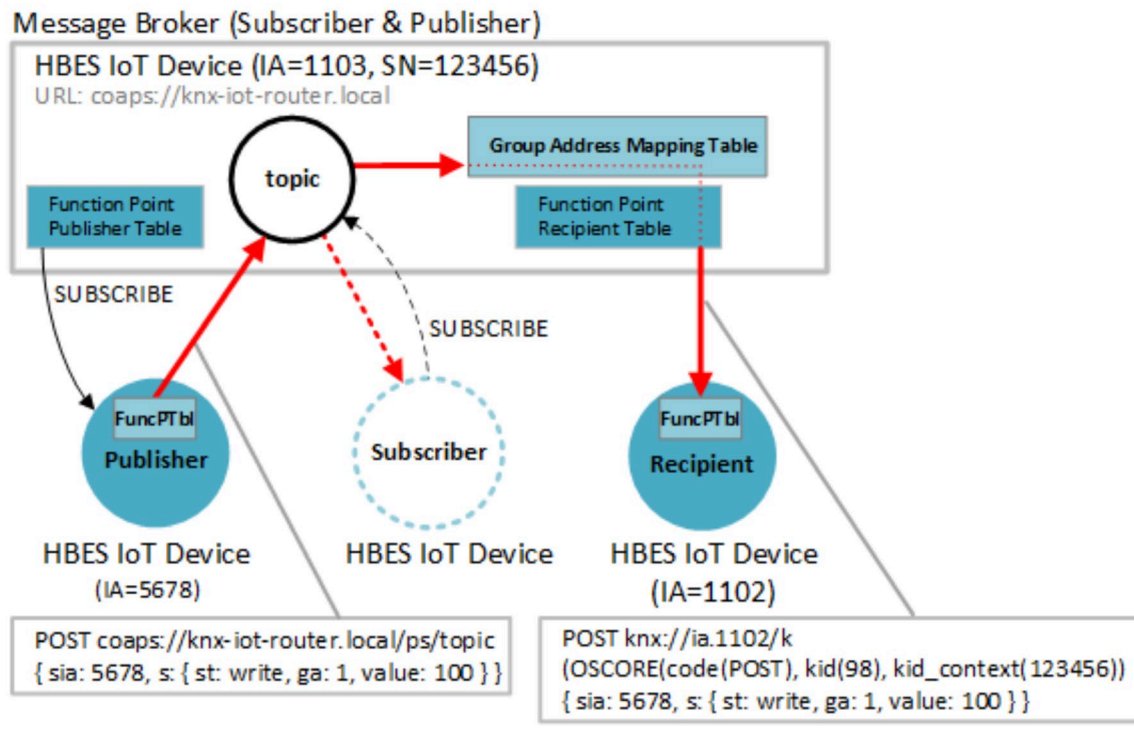


Figure 6 — Broker-based System

Figure 6 shows a typical communication scenario with a Message Broker, a (D)TLS Publisher, and an OSCORE Recipient. Note that the "kid" in the OSCORE message always contains the OSCORE input material identifier (osc-id) from the corresponding access token, and the "kid_context" contains the configured context in the OSCORE access token from the device that sends the message. It requires a Message Broker using technologies like AMQP, MQTT, etc., to bridge between different transport layers and protocols. In this example, the Message Broker relays messages from (D)TLS Publishers to Recipients (or Subscribers). A Message Broker can also queue and send the same message to multiple Recipients. Note that the Message Broker functionality and the protocol proxy (e.g., from CoAP to MQTT) are outside the scope of this document. In the context of HBES IoT, the Message Broker is just a CoAP server and an intermediate that delivers messages to HBES IoT Recipients. Hence, the Message Broker is a standard HBES IoT device, and HBES IoT defines how to connect to a Message Broker in a reliable and secure manner.

The unicast IPv6 address format is defined in 5.6.2.

Message confidentiality and integrity with the Message Broker can be ensured at two levels.

1. Application-level security as end-to-end security between Publisher and Recipient based on OSCORE, see 6.6 and/or
2. Transport security between Publishers/Subscribers and the Broker with (D)TLS.

The application-level security requires all Publishers, Subscribers, and Recipients to have pre-shared credentials that grant them access to the necessary group. The configuration of credentials and access scopes is described in 6.5.

5.3.4 Device Linking

5.3.4.1 Introduction

HBES IoT devices send and receive group messages on configured unicast or multicast IP addresses (messaging endpoint). A MaC (Management Client) configures the system communication interworking. For this purpose, the MaC configures the Function Point Tables of all devices in an installation. An HBES IoT device shall support Group Communication via

- Transport layer unicast and multicast (see 5.3.2),
- Publish-Subscribe (see 5.3.3).

5.3.4.2 Function Point Tables

The Function Point Tables of a device are *Group Address* mapping tables. The Function Point Tables have references from *Group Addresses* ("ga") of an HBES IoT device to a protocol-specific (e.g., CoAP) destination address for sending or source address for receiving messages (unicast or multicast URL).

The MaC configures Function Point Tables on all HBES IoT Point Devices in both a brokerless and a broker-based system. The Function Point Tables consist of the following tables:

- Point mapping configuration: The Group Object Table maps *Group Addresses* to a Point to read a value, store a value, or further processing of incoming values (see 5.5.7.3).
- Outgoing group event configuration: The Function Point Recipient Table (destination unicast/multicast address) defines to which devices or with which multicast address group events are sent (see 5.5.7.4).
- Incoming group event configuration: The Function Point Publisher Table (unicast subscription or multicast listening address) contains configurations from which devices group events need to be received (subscription address) or from which multicast address group events are received (see 5.5.7.5).

The Group Object Table of a device contains all *Group Object* Points ("href": {point-path}) that can be activated; the *Group Object* Points are defined by the manufacturer.

The HBES IoT *Group Address* is used for S-Mode communication. The *Group Address* is an unstructured ID ("ga"). The MaC assigns the "ga" with predefined *Group Objects* (Group Object Table) and writes the configuration to the HBES IoT device (see 5.4). In addition, the *Group Address* is also used to discover configured S-Mode *Group Objects* (see 5.6.1).

Figure 7 illustrates the general linkage concept between *Group Objects*, Points, and *Group Addresses* but does not prescribe how to implement the table structures.

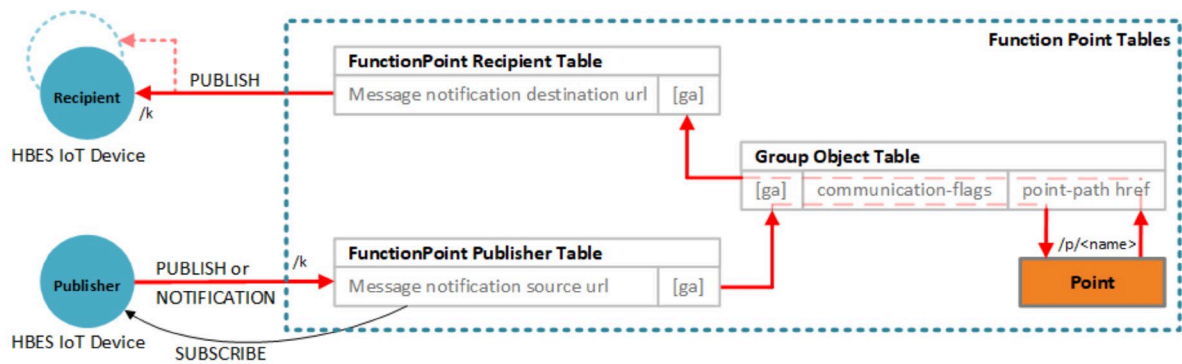


Figure 7 — Function Point Tables

5.4 Device Bootstrapping and Configuration

5.4.1 Introduction

Device Bootstrapping is the process of joining a device to an installation and configuring an application. The application configuration of an HBES IoT Point API device aligns with the existing MaC workflows and contains the following steps.

STEP 0 - Project creation in MaC.

This is as it is done for non-IoT HBES EN 50090 compatible devices. This can be done before, in parallel, or after step 1. This may include at this place as well offline configuration, i.e., setting parameters and creating links between the devices for installation parts engineered entirely offline.

STEP 1 - Enter HBES IoT Point API device *Serial number* in MaC, e.g., by QR code scan.

This step brings the serial number to MaC, needed for step 2, and, e.g., an authentication code required for step 3.

STEP 2 - Device Discovery.

Discovery of an HBES IoT Point API device follows the method described in 5.6.1.2.

Note With this version of the HBES IoT Point API standard, it is assumed that a device is reachable already on the IP network at this step, e.g., a wireless (Thread or Wi-Fi) HBES IoT device is already onboarded into the IP network and thus discoverable.

STEP 3 - Device Assignment, i.e., matching the discovered devices with the instances foreseen in the MaC project.

This step includes the security setup described under 6.2 and 6.5.

Device Individualization, i.e., assigning an *Individual Address*, is the same as described in EN 50090-4-1. This is either by using the *Programming Mode Method* or the *Serial number Method* in case the *Serial number* is already known to MaC, e.g., by QR code scan.

Device Individualization is described in more detail in 5.4.2.

STEP 4 - Offline configuration.

This step configures parameters and creates links between the devices (Function Point Table). At this place, this may also include configuration of a before online discovered and uploaded installation or part of an installation.

STEP 5 - Download device configuration (Application Download with MaC).

With this step, MaC configures the credentials at application level, pre-shared keys. This is, once downloaded, the devices can trust each other.

Device configuration is described in more detail in 5.4.3.

The above numbering does not imply a fixed order for executing these steps, and the order may differ as far as possible and meaningful. Some may even be repeated several times during setup/commissioning of an installation, as it is done for non-IoT EN 50090 compliant HBES devices.

Note This version of the HBES IoT Point API standard does not consider how to join, replace or delete a device from the network and how to avoid the replaced/removed device can still communicate.

5.4.2 Device Individualization Procedure

Every device in a KNX installation shall have a unique, unambiguous *Individual address*, and this clause defines how a MaC configures the HBES IoT device's *Individual address*.

The *Individual address* configuration workflow is like the existing procedure described in EN 50090-7-1. The device can either be identified by its *Serial number* or by activating the Programming Mode locally on the device, for example, by pressing a button. The MaC can choose one of the following discovery options:

- DNS-SD: see 5.6.1.2 with _knx service type.
- "/.well-known/core": see 5.6.1.3.
 - with *Serial number*, see 5.6.1.3.6.1.
 - with *Programming Mode*, see 5.6.1.3.6.2.

Before writing the *Individual address* to the discovered device, a MaC shall check that no other device has the same *Individual address*. The MaC has the following options to check *Individual addresses*:

- DNS-SD: see 5.6.1.2 with *_ia* subtype.
- *"/.well-known/core"*: see 5.6.1.3 or the example below.

Verify whether the <i>Individual address</i> is already occupied on the network.
REQ: GET coap://[FF03::FD]/.well-known/core?ep=knx://ia.33a3.20a
RES: 2.05 CONTENT (Content-Format: application/link-format (40))
Payload: <>;ep="knx://sn.001caffe1234 knx://ia.33a3.20a"

If the MaC receives a response to a multicast request, it shall conclude that the *Individual address* is occupied. If no response is received after a timeout, then the *Individual address* is not occupied, and subsequently, the MaC can configure the *Individual address*. For this configuration request, the MaC uses the sender IP address out of the IP response to the before-discovery request as the destination device IP unicast address.

The procedure shall deactivate the *Programming Mode* by restarting the device (*"/.well-known/knx"*).

At this point, the MaC knows the IP address of the uncommissioned HBES IoT device. However, the MaC may configure the security credentials first before it configures the *Individual address* (see 6.6.2)

In the last step, the MaC configures the device's *Individual address* (see 5.5.5.5). An HBES IoT device shall accept this configuration request and set its *Individual address* accordingly.

Write device <i>Individual address</i> .
REQ: POST coap://{ipv6-unicast}/.well-known/knx/ia (Content-Format: application/cbor (60), OSCORE(code(POST), kid(osc-id)))
Payload: { 12: < <i>Individual address</i> >, 25: <fid>, 26: <iid> }

5.4.3 Device Configuration Procedure

5.4.3.1 General

After the device individualization procedure, the MaC writes the device's new Function Point configuration (fp/g, fp/r, fp/p) and application configuration parameters (p) to the device. A Device Load State Machine controls these resources (see 5.5.8).

5.4.3.2 Full download

Table 3 specifies the configuration procedure for a full download. An example is given in 9.2.1.

Table 3 — Full download

Step	Procedure	Description
1	Check MaC DB Entry compatibility with HBES IoT device by comparing "dev/mid" and "dev/hwt".	See 5.5.6 Continue with the next steps only if the compatibility check is successful.

Step	Procedure	Description
2	Set State Machine to "unload".	See 5.5.8. If the intermediate state "unloading" is supported by the HBES IoT device (Management Server), it may answer with state "unloading". The MaC shall not continue before the HBES IoT device has reached the state "unloaded".
3	Set State Machine to "loading".	See 5.5.8.
4	Write parameter configuration.	See 5.5.11.
5	Write Group Object Table.	See 5.5.7.3.
6	Write Function Point Publisher Table.	See 5.5.7.5.
7	Write Function Point Recipient Table.	See 5.5.7.4.
8	Write Application Program Version "ap/pv".	See 5.5.8.
9	Set State Machine to "loaded".	See 5.5.8. If the intermediate state "loadcompleting" is supported by the HBES IoT device, then it may answer with the state "loadcompleting". The MaC shall not continue before the HBES IoT device has reached the state "loadcomplete".
10	Read the device configuration fingerprint of the written resources.	See 5.5.5.4. The retrieved checksum may be stored by the MaC for a future decision about a partial/differential download.
11	Restart device (optional).	See 5.5.5.3.3.

5.4.3.3 Partial/Differential download

Table 4 specifies the configuration procedure for a partial/differential download. The partial/differential download is used if the device is in a known state. In this case, the MaC can write only the differences to the device. An example is given in 9.2.2.

Table 4 — Partial/differential download

Step	Procedure	Description
1	Check MaC DB Entry compatibility with HBES IoT device by comparing "dev/mid" and "dev/hwt".	See 5.5.6. Continue with the next steps only if the compatibility check is successful.
2	Check if the HBES IoT device is in a known state.	See 5.5.5.4. If the device configuration fingerprint value is known by the MaC, then the MaC may continue with a differential download. Else, the MaC shall fall back to the full download procedure acc. 5.4.3.2, and continue there with Step 2.
3	Set State Machine to "loading".	See 5.5.8.
4	Write parameter configuration (conditional: only if changes there).	See 5.5.11. The MaC may write only changed parameters (identified by href).

Step	Procedure	Description
5	Write Group Object Table (conditional: only if changes there).	See 5.5.7.3. The MaC may write only changed group objects (identified by "id").
6	Write Function Point Publisher Table (conditional: only if changes there).	See 5.5.7.5. The MaC may write only changed table entries (identified by "id"). Entries with a new "id" will create a new entry.
7	Write Function Point Recipient Table (conditional: only if changes there).	See 5.7.4. The MaC may write only changed table entries (identified by "id"). Entries with a new "id" will create a new entry.
8	Write Application Program Version "ap/pv"	See 5.5.8.
9	Set State Machine to "loaded".	See 5.5.8. If the intermediate state "loadcompleting" is supported by the HBES IoT device, then it may answer with the state "loadcompleting". The MaC shall not continue before the HBES IoT device has reached the state "loadcomplete".
10	Read the checksum of the written resources.	See 5.5.5.4. The retrieved checksum may be stored by the MaC for a future decision about a partial/differential download.
11	Restart device (optional).	See 5.5.5.3.3.

5.5 Resource Model

5.5.1 Introduction

Resources are the basis for representing the state and functionality or, rather, an application. Resources provide access to standard application services like device and resource discovery, device-specific properties, and function calls. The resource model provides means for presenting, describing, and addressing data and metadata. Data is accessible by RESTful methods for creating, reading, setting, and deleting data. This clause gives rules for device interactions that need to be exposed for configuration and discovery and provides guidelines for using predefined attributes.

5.5.2 Resources (Points)

5.5.2.1 Definition

Points are application-specific API resources representing the application domain model. With a URI, a resource is uniquely defined, and a URL includes the schema to access the resource. Using HBES-specific nomenclature for resource names helps developers understand the functionality and basic semantics of resources, and it also reduces the need for further documentation of the API items. This clause defines mandatory and optional resource paths and names.

5.5.2.2 Base Path

An HBES IoT device may have a base path to which the Point API resource paths are appended. However, it is recommended to avoid a base path if possible.

5.5.2.3 Resource Names

To simplify the encoding of resource IDs in URLs, their representations shall only consist of ASCII strings of letters, numbers, underscore, minus, colon, and period.

URLs are often treated as case sensitive; therefore, lowercase separated words with hyphens "-" should be used.

This document does not define resource names for *Functional Blocks* (f) or properties (p). However, it is recommended to reuse either IDs or names from the Application Descriptions as defined by KNXA:

- Functional Block path: "{base-path}/f/{fb-id-instance}"
- Property path: "{base-path}/p/{fb-id-instance}/{property-id}"

5.5.2.4 Resource Path

The resource path definitions (see Table 5) allow access to items of an HBES IoT device, such as discovery, device, authorization, *Functional Blocks*, and Property items, based on entities as specified in EN 50090-6-2.

To keep the API and authorization simple, a resource path should contain only one collection or item resource with an {id} and may contain a sub-resource path for collections:

- collection path: "/collection"
- item path: "/items/{id}"
- collection path of a particular item: "/items/{id}/collection"

The resources follow "functional segmentation" and "separation of concern" design principles and do not mix different functionalities in the same API resource. This keeps maintenance and service evolution manageable.

The length of an HBES IoT device resource path shall not exceed the maximum of 30 bytes.

An HBES IoT device server has the following mandatory ("M") or optional ("O") resource path definitions with their corresponding access methods.

Table 5 — Resource path definitions

Category	Method	Resource Path	Support	Reference
Discovery	GET	/.well-known/core	M	5.5.4
Device	POST	/.well-known/knx	M	5.5.5
	POST	/.well-known/knx/ia	M	5.5.5.5
	GET	/.well-known/knx/f	M	5.5.5.4
	GET	{base-path}/dev	M	5.5.6
	GET	{base-path}/dev/{property-path}	M	
	PUT	{base-path}/dev/{property-path}	M/O	
	GET	{base-path}/swu	M	7.2
	GET	{base-path}/swu/{property-path}	M	
	PUT	{base-path}/swu/{property-path}	M/O	
Application Program	GET	{base-path}/ap	M	5.5.8
	GET	{base-path}/ap/{property-path}	M	
	PUT	{base-path}/ap/{property-path}	M/O	

Category	Method	Resource Path	Support	Reference
Security	POST	/well-known/knx/spake	M	6.6.3.3
	PUT	/well-known/knx/spake	M	6.6.3.5
	GET	/well-known/knx/idevid	O	6.3.1
	GET	/well-known/knx/ldevvid	O	6.3.2
	GET	/base-path/auth	M	6.2.5
	POST	/base-path/auth/crts	O	6.5.2
	GET	/base-path/auth/crts	O	
	DELETE	/base-path/auth/crts/{cert-id}	O	
	POST	/base-path/auth/at	M	6.5.4
	GET	/base-path/auth/at	M	
	GET	/base-path/auth/at/{token-id}	M	
	DELETE	/base-path/auth/at/{token-id}	M	
	GET	/base-path/auth/o/{property-path}	M	6.6.4
Messaging	POST	/k	M	5.5.9
	GET	/k	M	
Function Point Tables	POST	/base-path/fp/g	M	5.5.7
	GET	/base-path/fp/g	M	
	GET	/base-path/fp/g/{group-object-id}	M	
	DELETE	/base-path/fp/g/{group-object-id}	M	
	POST	/base-path/fp/r	M	
	GET	/base-path/fp/r	M	
	GET	/base-path/fp/r/{recipient-id}	M	
	DELETE	/base-path/fp/r/{recipient-id}	M	
	POST	/base-path/fp/p	O	
	GET	/base-path/fp/p	O	
	GET	/base-path/fp/p/{publisher-id}	O	
	DELETE	/base-path/fp/p/{publisher-id}	O	
Actions	POST	/base-path/a/lsm	M	5.5.8
	GET	/base-path/a/lsm	M	
	POST	/base-path/a/swu	O	7.2
	POST	/base-path/a/sen	O	6.2.4.2
Functional blocks	GET	/base-path/f/{fb-id-instance}	M	5.5.10
Properties	POST	/base-path/p	M	5.5.11
	GET	/base-path/p	O	
	GET	/base-path/p/{property-path}	M	
	PUT	/base-path/p/{property-path}	M	

Category	Method	Resource Path	Support	Reference
Subscriptions	DELETE	/{{base-path}}/sub	M	5.5.12

5.5.3 Interface Types (if)

5.5.3.1 Introduction and overview

This clause specifies a set of default interface types and defines how manufacturers can add additional manufacturer-specific interface types.

The interface describes a generic context or view to interact with a resource or a set of resources. The interface description "if" attribute is an opaque string that provides a name indicating a specific interface definition used to interact with the HBES IoT device. The usage of the interface description "if" attribute is described in 5.5.3.

The interface types are derived from the HBES Device Model (see 4.3), and may contain a set of additional mandatory or optional metadata.

Table 6 defines HBES IoT interfaces and which CoAP Methods are allowed for a particular interface on an HBES IoT device. Note that interface types can be combined:

Table 6 — Interface definitions and methods

Type	Interface	Method	Description
Link List	if.ll	GET, (OBSERVE)	Read a linked list and, in combination with if.o, subscribe to all Points of the list.
Parameter	if.p	GET, PUT, (OBSERVE)	Adjust parameter Point (see EN 50090-6-2).
Diagnostic	if.d	GET, (OBSERVE)	Read diagnostic Point (see EN 50090-6-2).
Configuration	if.c	GET, PUT, POST, DELETE	Configuration and programming of a device.
Logical Input	if.i	PUT, POST, GET	Write and command runtime input Group Object Point (see EN 50090-6-2 and 4.3).
Logical Output	if.o	GET, POST, OBSERVE	Read and subscribe runtime output Group Object Point (see EN 50090-6-2 and 4.3).
Group Communication	if.g.s	POST, GET, (OBSERVE)	Group communication (S-Mode) runtime interworking (input and output) address.
Batch	if.b	GET, PUT, POST	Read or write a collection (e.g., Point list).
Actuator	if.a	GET, PUT, POST	Hardwired actuator (see EN 50090-6-2 and 4.3).
Sensor	if.s	GET, PUT	Hardwired sensor (see EN 50090-6-2 and 4.3).
Security	if.sec	GET, PUT, POST, DELETE	Configuration (read and write) of security, incl. authorization-related data (see Clause 6).
Software Update	if.swu	GET, PUT, POST, DELETE	Software update (push and pull) related data (see Clause 7).
Programming Mode	if.pm	GET	Data that can be read in Programming Mode.
Manufacturer	if.m.{name}	Manufacturer-specific definition	Manufacturer-specific interface types.

5.5.3.2 Link List Interface Type (if.ll)

The Link List interface gradually reveals resources on an HBES IoT device. It is used to retrieve (GET) a list of resources on an HBES IoT device. The GET request should contain an Accept option with the application/link-format content format. The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format RFC 6690. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on an origin server.

CoAP client requests a link list (datapoint list) from a Room Temperature Sensor (Functional Block 321).
REQ: GET coap://{ip unicast}/f/rts
RES: CONTENT (Content-Format: application/link-format (40)) Payload: </p/rts/temproom>;rt=":dpa.321.51";ct=50, </p/rts/tempcorrvalue>;rt=":dpa.321.111";ct=50, </p/rts/tempalarmlimitupper>;rt=":dpa.321.113";ct=50

5.5.3.3 Parameter and Diagnostic Interface Type (if.p, if.d)

Parameter and diagnostic Properties (see 4.3) are the basic interfaces of a Point API and define whether a client can only read or read and update values and metadata, such as getting the current sensor value or updating a setpoint. The following mandatory and optional metadata members from 5.5.11.3 are assigned to the "if.p" (parameter) and "if.d" (diagnostic) interface:

- id
- rt
- if
- unit
- min
- max

A Point may implement a Parameter ("if.p") and Diagnostic Property ("if.d") in combination with a communication interworking interface type ("if.i", "if.o"). In such cases, it is recommended to combine only "if.i" with "if.p" (inputs) or "if.o" with "if.d" (outputs).

5.5.3.4 Configuration Interface Type (if.c)

The Configuration interface is used to configure a basic functionality, Function Point Tables, or for programming an HBES IoT device.

5.5.3.5 Input Interface Type (if.i)

A communication interworking logical input Point (see 4.3) receives events from Publishers. An input interface Point receives either status events (e.g., light switch ON/OFFF) or periodic updates from Publishers (if.o interface). A Point with an interface type "if.i" shall not be of interface type "if.o" at once.

The Recipient device may have a default behavior if no update is received within a given period (e.g., Max-Age timeout). For example, a controller gets a periodic synchronization from an output temperature sensor and uses

a default value if the sensor value is not present (e.g., broken or not configured). It is recommended to use the "hbt" metadata member for timeout configurations in combination with the following timeout calculation: $\text{timeout} = (\text{"hbt"} * 2) + 1$.

Example: $\text{timeout} = (15\text{min} * 2) + 1 = 31\text{min}$

5.5.3.6 Output Interface Type (if.o)

A communication interworking logical output Point (see 4.3) sends status events (e.g., light switch ON/OFF) or periodic updates (e.g., temperature sensor) from a Publisher to a Receiver ("if.i" interface). A Point with an interface type "if.o" shall not be of interface type "if.i" at once.

An output interface Point shall support S-Mode bindings for event notifications and should support subscriptions (CoAP Observe). By enabling additional subscriptions on output interface Points, a monitoring system can dynamically monitor values depending on the situation, eliminating the need to configure group objects in advance. Examples are operating modes, setpoints, status updates, sensor values, etc. Those values, such as an outside temperature value, may have additional metadata used to synchronize values throughout the system. The following metadata members from 5.5.11.3, may be assigned to the "if.o" interface:

- mrt
- cov
- hbt

5.5.3.7 Group Object Interface (if.g.s)

Points with a Groups Object interface support S-Mode event notifications, and a MaC can link *Group Addresses* to this object. The following metadata members from 5.5.11.3, are assigned to the "if.g.s" interface:

- ga

5.5.3.8 Batch Interface Type (if.b)

The Batch interface ("if.b") manipulates a collection of sub-resources or an array of values with a single request. A resource with an "if.b" interface shall support pagination as defined in 5.6.8. Single items can be accessed with pagination query parameters starting with pn=0 if no query parameter is present.

If a resource contains an array of datapoint values, then the resource shall be of type Batch interface ("if.b").

Datapoint values shall be modeled as key/value pairs in the GET response (see rules in 5.5.11.1), but only if the client requests more than one item ($ps > 1$). The default list size shall be $ps=1$ if the query parameter is absent.

The HBES IoT device GET response shall list items in the same order as they were written (first array item = "po=0"). An HBES IoT device shall support write (PUT) of a single value, for example, on "if.p" interfaces. However, updating a list of values on the same resource is optional.

The CoAP client reads an array of Ipv6 configurations with a single request.

REQ:

GET coap://{ipv6-unicast}/dev/ipv6?pn=0&ps=3

RES:

CONTENT (Content-Format: application/cbor (60))

Payload:

```
[
  { 1: "20010db8000300006cd98ad28e881a47" }, //value
  { 1: "fe80000000000006cd98ad28e881a47" }, //value
  { 1: "fd0000ff1ce0ba56cd98ad28e881a47" } //value
]
```

The client omits pagination parameters and gets only the first list item.

REQ:

GET coap://{ipv6-unicast}/dev/ipv6

RES:

(Content-Format: application/cbor (60))

Payload:

```
{ 1: 20010db8000300006cd98ad28e881a47 } //value
```

The interface type of a resource representing a collection of sub-resources shall be of type Batch interface. The Batch interface resource shall support the same methods as its sub-resources and can be used to read (GET), update (PUT), or apply (POST) the values of that sub-resource. The method used on Batch only applies to sub-resources that support it. For example, sensor interfaces do not support PUT; thus, a PUT request to a Sensor member of that Batch would be ignored.

The next example shows how a MaC configures multiple values of one or many Properties with a single request.

CoAP client writes a collection of datapoints to a device with a single request to a Room Temperature Sensor (Functional Block 321).

REQ:

POST coap://{ipv6-unicast}/p (Content-Format: application/cbor (60))

Payload:

```
[
  { 11: "/p/rts/tempcorrvalue", 1: 0.5 },
  { 11: "/p/rts/tempalarmlimitupper", 1: 27.2 }
]
```

5.5.3.9 Programming Mode Interface Type (if.pm)

A MaC may want to find devices in *Programming Mode* during the device bootstrapping and configuration step. Hence, an HBES IoT device shall support the *Programming Mode*.

With the *Programming Mode*, the destination HBES IoT device can be identified by selecting the device manually. This can be done by pressing a button that brings this device into *Programming Mode*, i.e., only the

device where the button is pressed. The way an HBES IoT device is set to *Programming Mode* is manufacturer specific.

5.5.3.10 Manufacturer Specific Interface Type (if.m.{name})

The manufacturer interface allows manufacturer-specific interface types on resources, such as manufacturer-specific methods (GET, PUT, etc.). Manufacturer-specific interface types shall start either with "urn:" or ":if.m." followed by a manufacturer-specific string {name}.

5.5.4 Device Discovery Resource (.well-known/core)

For CoAP-based resource discovery purposes, an HBES IoT device shall support the /.well-known/core resource [RFC6690], and the device shall accept unicast and multicast discovery requests on this resource. More details are described in 5.6.1. In the case of multicast discovery queries, an HBES IoT device shall follow the rules for response suppression described in 5.6.4.3. In the case of unicast discovery queries, an HBES IoT device should respond as described in 5.6.3.3.2.

The unicast response shall contain all *Functional Block* resources of an HBES IoT device. The response may have only *Functional Block* resources to reduce the message size. However, the response shall contain all implemented *Functional Blocks* standardized by KNXA and supported content types (ct). In addition, the following resources shall be in the response if supported by the device:

- /{base-path}/dev
- /{base-path}/auth
- /{base-path}/swu
- /k

The multicast response shall only contain the link-format line as described in 5.6.1.3.6.1.

<>;ep="knx://sn.001caffe1234 knx://ia.33a3.20a"

The *Functional Block* resources in the response shall support the application/link-format. This allows clients to discover Points with subsequent requests (see 5.5.10).

The response shall contain the namespace identifier "urn:knx" since the device context is unknown to the client at this stage.

CoAP client discovers device Functional Blocks.
REQ: GET coap://{ipv6-unicast}/.well-known/core RES: CONTENT (Content-Format: application/link-format (40)) Payload: </auth>;rt="urn:knx:fb.auth";ct=40, </dev>;rt="urn:knx:fb.0";ct=40, </swu>;rt="urn:knx:fb.swu";ct=40, </k>;rt="urn:knx:if.g.s"ct=60, </f/rts>;rt="urn:knx:fb.321";ct=40

5.5.5 Device API Resource (.well-known/knx)

5.5.5.1 General requirement

An HBES IoT device shall support the "/.well-known/knx" resource for HBES-specific device operations, such as device individualization, security configuration (see Clause 6), reset to the default state, or device restart.

5.5.5.2 API Version Resource Object

The "/.well-known/knx" resource also provides the HBES IoT API version information and relationship information to identify the API entry point for a MaC or any other client. A basic response contains the data listed in Table 7. mandatory ("M") attributes shall be present in the message.

Table 7 — Data in the basic response

JSON Key	CBOR Key	CBOR Type	Support	Description
"api"	"api"	map	M	The API object contains the most recent supported API version on the HBES IoT device.
"version"	"version"	Text string	M	Major and minor version of the HBES IoT Point API ("<major.minor.patch>").
"base"	"base"	Text string	M	The {/base-path}/ member value is part of the entire URL of an HBES IoT Point API. The device shall reply with "/" if the {base-path} is empty or at the end of the {base-path}. This simplifies concatenation with relative paths. A client shall include the resource path base member value when requesting content from an HBES IoT device (see 5.5.2.2).

Semantic versioning contains major, minor, and patch information. The version member value shall contain the major, minor, and patch number according to the format major.minor.patch, represented as a string. The major number starts from the value 1; the minor and patch numbers begin with the value 0.

On any compatible change with the current HBES IoT Point API, the major number remains, and the minor number shall be incremented. The base member value shall not be changed.

On any incompatible change with the current HBES IoT Point API, the new major number shall be incremented. The new minor number shall be set to 0.

The HBES IoT device Point API shall have version 1.0.0 if the API complies with this standard version and implements the mandatory resource definitions defined in 5.5.2.4".

Table 8 defines mandatory ("M") resources of an HBES IoT device and the corresponding resource path names that shall be used.

Table 8 — Mandatory and option configuration resources

Resource path	rt & Data Types	Format	Method	Support	Request/Response	Notes
/.well-known/knx	NA	cbor (json)	GET	M	Res: Content-Format: application/json Payload: <pre>{ "api": { "version": "1.0.0", "base": "/" } }</pre>	Device KNX Point API version and base path.

5.5.5.3 Device Command Resource Object

5.5.5.3.1 Overview

The functionality of the reset resource is derived from the functionality of the Master Reset Service as defined in EN 50090-7-1.

An HBES IoT device shall support the reset command (cmd), with supported erase codes 0x2 ("Reset to default state") and 0x7 ("Reset to default without IA").

With erase code 0x2 ("Reset to default state"), all addressing information and security configuration data shall be reset to the default state. With erase code 0x7 ("Reset to default without IA"), all configuration data shall be reset to default state except addressing information (IA, Device IP address, Installation ID) and security configuration data (credentials) that are needed after the reset to access the device without the need to discover the device again and/or renew addressing information and security credentials.

Table 9 defines JSON keys and the CBOR mapping for CoAP request and Table 10 response members of "/.well-known/knx". Mandatory ("M") attributes shall be present in the message, and optional ("O") may be present in the message, but all attributes shall be supported on the device.

Table 9 — Command members of "/.well-known/knx"

JSON Key	CBOR Key	CBOR Type	Support	Description
"value"	1	unsigned	M	Erase-code value (see EN 50090-7-1).
"cmd"	2	text string	M	Service command. Enum: reset, restart.

Table 10 — Response members of "/.well-known/knx"

JSON Key	CBOR Key	CBOR Type	Support	Description
"code"	"code"	unsigned	M	Device response error code (see EN 50090-7-1).
"time"	"time"	unsigned	M	Process time (see EN 50090-7-1).

5.5.5.3.2 (Master) Reset Command

An HBES IoT device provides the functionality to reset it to its default state, which typically corresponds to the KNX default state of the device’s configurable data. Resetting a device to the default state is a device-specific operation. An HBES IoT device, therefore, shall support the following command to allow a MaC, or any other client with appropriate rights, to reset the HBES IoT device to its default configuration state.

Table 11 defines mandatory ("M") resources of an HBES IoT device and the corresponding resource path names that shall be used.

Table 11 — Device Reset Command

Resource path	rt & Data Types	Format	Method	Support	Request/Response	Notes
/.well-known/knx	NA	cbor	POST	M	Req: Content-Format: application/cbor Payload: { 2: "reset", 1: <erase-code> } Res: Content-Format: application/cbor Payload: { "code": <error-code>, "time": <process-time> }	The CoAP reset request shall be confirmed by the device before the CoAP connection breaks down.

To perform a Master Reset, the HBES IoT device shall reset its configuration data according to the following:

- The Function Point Tables shall be cleared. Hence the Master Reset stops runtime interworking (group communication).
- Depending on the erase code, any network and security parameters shall be reset to their KNX default default state (applicable for erase codes 0x2 and 0x3, but explicitly not for erase code 0x7); with erase-code 0x7, all security parameters (auth/at, auth/crts) shall be reset, except the parameters under access scope "if.sec".
- A (Basic) Restart is executed.

For the definition of erase codes, see EN 50090-7-1. From the list of the erase codes defined there, an HBES IoT device shall support erase codes 0x2 and 0x7. It may support further erase codes, e.g., 0x3, for explicitly resetting the network and security parameters.

Before executing the reset function, the HBES IoT device shall return a response with CoAP response code "2.04 Changed" and with a payload containing Error Code and Process Time in seconds as defined for the response to a Master Reset Request for non-IoT EN 50090 compatible HBES devices, see EN 50090-7-1. The following example shows how a MaC sets an HBES IoT device to its default configuration state.

(Master) Reset to the default state.
REQ: POST coap://{ipv6-unicast}/.well-known/knx (Content-Format: application/cbor (60)) Payload: { 2: "reset", 1: <erase-code> }
RES:

2.04 CHANGED (Content-Format: application/cbor (60))

Payload:

```
{ "code": <error-code>, "time": <process-time> }
```

5.5.5.3.3 (Basic) Restart Command

An HBES IoT device shall support the following command to allow a MaC, or any other client with appropriate rights, to restart an HBES IoT device.

Table 12 defines mandatory ("M") resources of an HBES IoT device and the corresponding resource path names that shall be used.

Table 12 — Device Restart command

Resource path	rt & Data Types	Method	Support	Request/Response	Notes
/.well-known/knx	NA	POST	M	Req: Content-Format: application/cbor Payload: { 2: "restart" }	The device shall not confirm the CoAP restart request, and the restart may result in a breakdown of the CoAP connection.

To perform a (Basic) Restart, the HBES IoT device shall

- switch off Programming Mode
- terminate the validity of the responder key of an open PASE session (access token with profile "coap_pase", see 6.6.3)
- apply changed configuration parameters at the latest 30 s after completing the restart.

It is explicitly not required but optional to restart the network stack or the overall hardware/firmware of the device. Consequently, a restart may result in a breakdown of the CoAP connection.

5.5.5.4 Device Configuration Fingerprint (.well-known/knx/f)

An HBES IoT device shall provide a resource to read a fingerprint of the device configuration settings (e.g., a checksum). The device configuration fingerprint shall include parameters that can be changed of the following resources:

- /{base-path}/fp/g
- /{base-path}/fp/r
- /{base-path}/fp/p
- /{base-path}/p

A MaC may use the device configuration fingerprint to decide whether a differential download is possible or not. The calculation of the device configuration fingerprint is manufacturer specific but shall reflect distinct device configuration settings. The HBES IoT device shall reply with the configuration fingerprint only in "loaded" state.

Suppose the HBES IoT device cannot immediately respond since it is busy (e.g., status = "unloading" or "loadcompleting"); the device shall return response code "5.03 Service Unavailable" but use the Max-Age option to indicate the number of seconds after which to retry.

Table 13 defines mandatory ("M") resources of an HBES IoT device and the corresponding resource path names that shall be used.

Table 13 — Device Configuration Fingerprint resource

Resource path	rt & Data Types	Method	Support	Request/Response	Notes
/well-known/knx/f	NA	GET	M	Req: Content-Format: application/cbor Payload: { 1: <value> }	Device-specific fingerprint to identify changes in the resources "fp/g", "fp/r", "fp/p", and "p".

5.5.5.5 Device Individualization Resource (.well-known/knx/ia)

Each HBES IoT device, i.e., a Router or an end device, shall have a unique *Individual address* in a network. The *Individual address* is a 2-octet value that consists of an 8-bit Subnetwork Address (see "dev/sna") and an 8-bit Device Address (see "dev/da").

<i>Individual address</i>	
Octet 0	Octet 1
Subnetwork Address	Device Address

EXAMPLE Subnetwork Address: 0, Device Address: 1 = 0x0001

The individualization process is derived from the A_IndividualAddress_Write-service defined in EN 50090-4-1. However, HBES IoT splits the functionality into a *Programming Mode* discovery (see 5.6.1.2.3 and 5.6.1.3.6.3) and a device address configuration step.

The "/.well-known/knx/ia" resource shall be used by the MaC to modify the *Individual address* and the installation ID on an HBES IoT device (see 5.4.2).

The device individualization request shall contain the new *Individual address* and installation ID. Table 14 defines the mandatory ("M") resource object members and CBOR keys. If a member is marked as optional ("O"), a client can omit the member in the request. However, the server shall support the member handling.

Table 14 — Device Individualization Resource Object

JSON Key	CBOR Key	CBOR Type	Support	Description
"ia"	12	unsigned	M	New Individual address device configuration (2 Byte).
"fid"	25	unsigned	O	New KNX Fabric ID device for unicast address configuration.
"iid"	26	unsigned	M	New KNX Installation ID for device multicast configuration.

Supposed the HBES IoT device has changed its local *Individual address* (see "dev/sna", "dev/da", "dev/iid", and "dev/fid" in 5.5.6), then the HBES IoT device shall return a response with CoAP response code "2.04 Changed". After responding to the client, the HBES IoT device shall change the Ipv6 address if the KNX Fabric ID has changed (see ULA 5.6.1.2).

Table 15 defines mandatory ("M") resources of an HBES IoT device and the corresponding resource path names that shall be used.

Table 15 — Device Individualization Resource

Resource path	rt & Data Types	Method	Support	Request/Response	Notes
/well-known/knx/ia	NA	POST	M	Req: Content-Format: application/cbor Payload: { 12: <Individual address>, 26: <iid> }	Device individualization for configuring the device <i>Individual address</i> and installation ID.

5.5.6 Device Object Resource (dev)

The device object contains general device-specific settings and configurations. The MaC configures some Properties (see 5.4.3), and other Properties are configured in the factory (e.g., *Serial number*).

Table 16 specifies the mandatory ("M") or optional ("O") device resources (dev) and the corresponding resource path names that shall be used. (PUT) is optional since the property value is usually preconfigured.

Table 16 — Device resources and paths

Resource path	rt & Data Types	Format	Method	Support	Request/Response	Notes
dev	:fb.0	link-format	GET	M	Content-Format: application/link-format Payload: </dev/sn>;rt=":dpa.0.11";ct=60, </dev/fww>;ct=60, </dev/hwt>;ct=60, </dev/model>;rt=":dpa.0.15";ct=60, </dev/sna>;rt=":dpa.0.57";ct=60, </dev/da>;rt=":dpa.0.58";ct=60, </dev/hname>;ct=60, </dev/ipv6>;ct=60, </dev/fid>;ct=60, </dev/iid>;ct=60, </dev/mport>;ct=60, </dev/pm>;rt=":dpa.0.54";ct=60	Device <i>Functional Block</i> .
dev/sn	:dpa.0.11 :dpt.serNum	cbor (json)	GET	M	Content-Format: application/cbor Payload: { 1: "001caffe1234" }	Device <i>Serial number</i> .
dev/mid	:dpa.0.12 :dpt.value2 Ucount	cbor (json)	GET	M	Content-Format: application/cbor Payload: { 1: 0x00fa }	Manufacturer Identifier

Resource path	rt & Data Types	Format	Method	Support	Request/Response	Notes
dev/hwv	:dpt.version	cbor (json)	GET	O	Content-Format: application/cbor Payload: { 1: [1,2,3] }	Hardware version.
dev/fwv	:dpa.0.25 :dpt.version	cbor (json)	GET	M	Content-Format: application/cbor Payload: { 1: [1,2,3] }	Firmware version.
dev/hwt	:dpt.varString8859_1	cbor (json)	GET	M	Content-Format: application/cbor Payload: {1: "000102030A0B"}	The hardware type is a manufacture-specific ID for a device type (MaC uses this ID for compatibility checks). Length: exactly 12 octets of string characters, formatted as upper-case, in hexadecimal notation. The MSB shall be "00".
dev/model	:dpa.0.15 :dpt.utf8	cbor (json)	GET	M	Content-Format: application/cbor Payload: { 1: "QAA987" }	Device Model Maximum length: 10 octets.
dev/sna	:dpa.0.57 :dpt.value1 Ucount	cbor (json)	GET	M	Content-Format: application/cbor Payload: { 1: 1 }	Subnet Address. Default: 255
dev/da	:dpa.0.58 :dpt.value1 Ucount	cbor (json)	GET	M	Content-Format: application/cbor Payload: { 1: 2 }	Device Address. Default: 255
dev/hname	:dpt.varString8859_1	cbor (json)	GET (PUT)	M	Content-Format: application/cbor Payload: {1: "knx-001caffe1234.local" }	Device hostname for DNS resolution with length definitions according to RFC 1035, 2.3.4. The device needs to support at least 60 bytes for the hostname length.

Resource path	rt & Data Types	Format	Method	Support	Request/Response	Notes
dev/ipv6	:dpt.ipv6	cbor (json)	GET (PUT)	M	Content-Format: application/cbor Payload: { 1: h'20010db8000300006 cd98ad28e881f68'} or Content-Format: application/cbor Payload: [[1: h'20010db8000300006 cd98ad28e881f68'], {1: h'fde58dba82e100010 00000fffe000401'}]	Array of device ipv6 addresses. The resource interface shall be "if=if.b" (see 5.5.3.8).
dev/fid	:dpt.value8 Ucount	cbor (json)	GET (PUT)	M	Content-Format: application/cbor Payload: { 1: 0x11222233a3 }	40-bit KNX Fabric ID (see ULA in 5.6.2). Maximum: 0x00ff'ffff'ffff
dev/iid	:dpt.value8 Ucount	cbor (json)	GET (PUT)	M	Content-Format: application/cbor Payload: { 1: 0x00000033a3 }	40-bit KNX Installation ID (see ULA in 5.6.5). Maximum: 0x00ff'ffff'ffff
dev/port	:dpt.value2 Ucount	cbor (json)	GET (PUT)	M	Content-Format: application/cbor Payload: { 1: 5683 }	Port number used for unicast comm. (see 5.6.3.1).
dev/mport	:dpt.value2 Ucount	cbor (json)	GET	M	Content-Format: application/cbor Payload: { 1: 5683 }	Port number used for multicast discovery (see 5.6.4.1)
dev/pm	:dpa.0.54 :dpt.binary Value	cbor (json)	GET PUT	M	Content-Format: application/cbor Payload: { 1: false }	Programming Mode enable/disable.

5.5.7 Function Point Table (fp)

5.5.7.1 Introduction and Overview

This clause describes the configuration of group events on an HBES IoT device. A MaC can choose between several message exchange configurations. The definition of when to apply which message exchange pattern is out-of-scope of this document. An HBES IoT device shall support at least the following message exchange configurations:

- Multicast message exchange
- Direct unicast message exchange

An HBES IoT device may support, and the HBES IoT Router shall support, the following message exchange configuration:

- Endpoint subscription message exchange (see Clause 10).

In Figure 8, the MaC configures a multicast message exchange, for example, between one or many sensors (e.g., light switches) and one or many actuators (e.g., light actuators). In addition, the actuator status is sent to an HMI (e.g., room unit) for visualization. For multicast group events, the MaC shall configure the following Function Point Tables:

- GA and Points in the Group Object Table on all devices.
- Multicast send address in the Function Point Recipient Table on devices with "if.o" interfaces.
- Multicast listening address in the Function Point Publisher Table on devices with "if.i" interfaces.

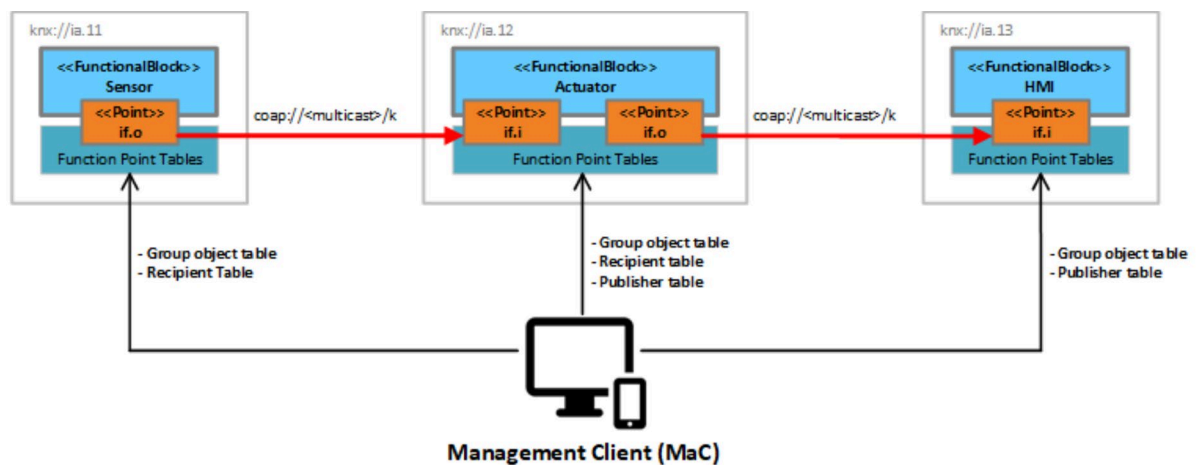


Figure 8 — Multicast Message Exchange

In the next example, the MaC configures direct unicast message exchange in networks where multicast reachability between devices is impossible or for applications that need reliable (confirmed) communication between devices. Figure 9 depicts a unicast example with a controller (e.g., HVAC controller) which gets sensor updates (e.g., room temperature sensor) and commands an actuator accordingly (e.g., radiator valve). The MaC shall configure for unicast group events the following Function Point Tables:

- GA and Points in the Group Object Table on all devices.
- Unicast destination address in the Function Point Recipient Table on devices with "if.o" interfaces (*Individual address 21 and 22 are sending messages to Individual address 22 and 23*).
- No Function Point Publisher Table configuration on all devices.

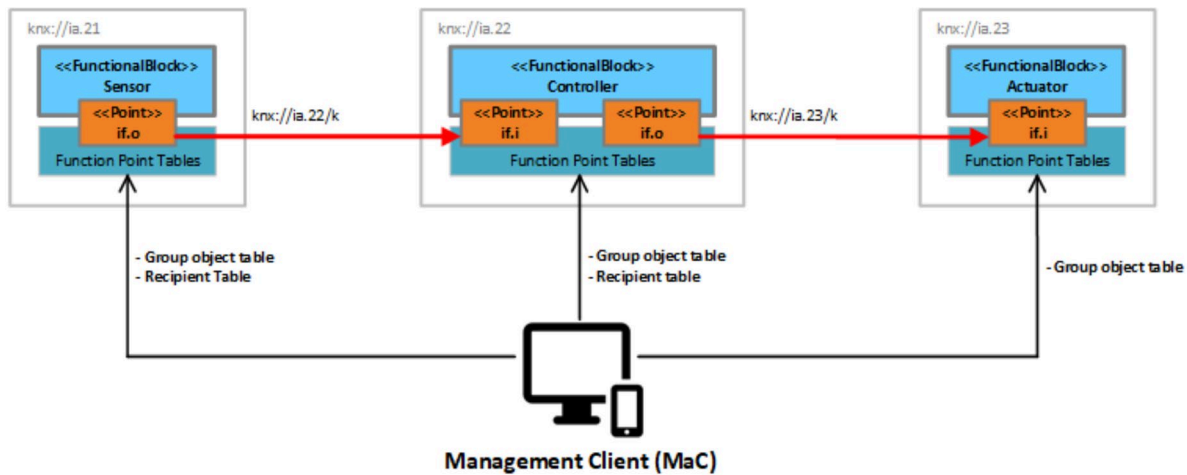


Figure 9 — Direct Unicast Message Exchange

5.5.7.2 Device Load State Machine State

If a CoAP request method (GET, PUT, or POST) to Group Object Table, Function Point Recipient Table, or Function Point Publisher Table items or collections is possible depends on the current Device Load State Machine state (see 5.5.8). CoAP GET shall be possible in all Device Load State Machine states. However, in states loaded, unloaded, unloading, and loadcompleting (state $\neq$ loading), the HBES IoT device shall return response code "4.05 Method Not Allowed" to PUT, POST, and DELETE requests.

5.5.7.3 Group Object Table Resource (fp/g)

5.5.7.3.1 Definition

An HBES IoT device shall implement a Group Object Table. The Group Object Table shall be writable; it is configured by the MaC. The Group Object Table contains configuration information for each *Group Object* used by the device.

A *Group Object* is represented by a manufacturer-specific resource path ("href": "{point-path}"). A *Group Object* has a set of mandatory and optional metadata members (see 5.5.11.3 and 5.5.11.4). A Metadata member is accessible as a resource behind the {point-path}. A MaC shall use metadata members such as CoV, heartbeat, min/max limitations, etc., to configure *Group Object* communication behavior.

If a function has extended features with additional configuration parameters, then the MaC may write values to other manufacturer-specific resource paths ({point-path}). This document does not specify how to implement manufacturer-specific Parameters and Diagnostic Properties.

Every *Group Object* shall implement a ("dpt") metadata for retrieving Datapoint Types and, if possible, semantic Datapoint Annotations ("rt").

Datapoint Types are identified by a prefix (urn:knx:dpt) and a Datapoint Type separated by a dot as specified in EN 50090-6-2. For example, a DPT_BinaryValue is encoded as "urn:knx:dpt.binaryValue". Datapoint Types with the same main number have the same data type (e.g., integer or boolean), format (e.g., B1 or B2), and encoding. A different subnumber indicates a different dimension (different range and/or different unit). Datapoint Types are laid down in EN 50090-3-3, and are used accordingly in HBES IoT. The Datapoint Type shall be defined for every *Group Object* and can be retrieved by reading {point-path}?m=dpt.

All *Group Objects* should have a semantic Datapoint Annotation, for example, for filtering discovery requests. Datapoint annotations have the following structure: urn:knx:dpa.<functionalblock-id>.<property-id>. Datapoint Annotations are inherited from the existing *Functional Block* and Property standard defined by KNXA. A Datapoint Annotation does not imply that a specific *Functional Block* functionality is implemented. Every *Group Object*, or rather the assigned {point-path}, should have a Datapoint Annotation unless the appropriate *Functional Block* and Property has not yet been defined by KNXA.

A *Group Object* contains a list of *Group Addresses* ("ga"). However, only one sending *Group Address* can be used for a *Group Object*. Hence, if more than one *Group Address* is configured, then the first list item shall be used as sending *Group Address*. In addition, if multiple *Group Objects* are linked to the same Point ("href"), then the *Group Object* with the lowest "id" shall contain the sending *Group Address*. The "href" member shall reference a {point-path}.

Table 17 specifies the mandatory ("M") or optional ("O") Group Object Table resources and the corresponding resource path names that shall be used.

Table 17 — Group Object Table resources and paths

Resource path	rt & Data Types	Format	Method	Support	Request/Response	Notes
fp/g	NA	cbor	POST	M	Req: Content-Format: application/cbor Payload: (see example below)	Write Group Object Table. Single Elements are identified by their "id". Existing elements are replaced, and not existing elements are added. An empty element (only "id") in the payload will cause the deletion of a corresponding existing element.
fp/g	NA	cbor	GET	M	Resp: Content-Format: application/link-format Payload: </fp/g/1>;ct=60	Group object item linked list.
fp/g/{group-object-id}	NA	cbor	GET DELETE	M	Req: Content-Format: application/cbor Payload: (see example below)	Read/delete single list item. The group-object-id corresponds to the resource object "id" value as a number in ASCII format.

5.5.7.3.2 Function Point Group Object Resource Object

Table 18 defines the mandatory ("M") Function Point Group Object resource object CBOR keys for configuring and reading Group Object Table items. mandatory ("M") attributes shall be present in the message.

Table 18 — Function Point Group Object resource object CBOR keys

JSON Key	CBOR Key	CBOR Type	Support	Description
"id"	0	unsigned	M	The ID of the Group Object Table Entry. It shall be unique in the table (range 0...65 535). Note: There is no sorting requirement for IDs.

JSON Key	CBOR Key	CBOR Type	Support	Description
"ga"	7	unsigned array	M	A List of all connected <i>Group Addresses</i> . The Group Object Table shall support >= 20 array elements. An array element shall support values up to 4.294.967.295. The device shall only accept Function Point Group Object resource objects with at least one "ga" element assigned to it, otherwise a response code of "4.00 Bad Request" will be returned from the device.
"cflag"	8	unsigned	M	Group Object Configuration Flags (see 5.5.7.3.3).
"href"	11	text string	M	Reference to the Point, which represents the data of this Group Object.

In the following example, the MaC configures two Group Object Table items on an HBES IoT device. The "cflag" for the first item is configured as "read", "write", "transmit", and "update" (0b11011000). The "cflag" for the second item is configured as "transmit" (0b01000000).

Write Group Object Table items.
REQ: POST coap://{ipv6-unicast}/fp/g (Content-Format: application/cbor (60)), OSCORE(code(POST), kid(<osc-id>))) Payload: [{ 0: 1, 11: "/p/ldsb1/soo", 7: [2305, 2401], 8: 216 //0b11011000 }, { 0: 2, 11: "/p/ldsb1/rsc", 7: [2306], 8: 64 //0b01000000 }]

5.5.7.3.3 Group Object Configuration Flags Resource Object "cflag"

An HBES IoT device shall support for each *Group Object* the non-volatile configuration flags "read", "init", "write", "transmit", and "update", describing the communication behavior of the corresponding *Group Object*. Like in

non-IoT EN 50090 compatible HBES devices, the configuration flags are configured by a MaC and are written to the device as part of the Group Object Table configuration, see 5.5.7.3.

The meaning and action of the configuration flags "read", "init", "write", "transmit", and "update" are explained in Table 19.

Table 19 — Group Object configuration flags

Flag	Bit	Description
Reserved	0, 1	Default = false
Communication	2	Not used in HBES IoT since the flag is just a logical disjunction (OR) of the other configuration flags (default = false).
Read	3	false = <i>Group Object</i> value cannot be read. true = With a read command to a <i>Group Object</i> , the device sends this <i>Group Object's</i> value.
Write	4	false = <i>Group Object</i> value cannot be written. true = The device receives and overwrites its <i>Group Object</i> value (default for inputs).
Init (Value read on initialization)	5	false = Disable read after initialization. true = <i>Group Object</i> read request after initialization (e.g., power up). The read-Flag on the producer side is expected to be true to get a response.
Transmission	6	false = <i>Group Object</i> value is not transmitted. true = For CoV, event, or heartbeat, the device sends its <i>Group Object</i> value (default for outputs).
Update	7	false = <i>Group Object</i> value is not updated. true = The <i>Group Object</i> value is updated with a group value response message if flag w=true.

5.5.7.3.4 Group Object Table Modifications

A MaC shall create or update *Group Object* items by sending a CoAP POST request to `/[base-path]/fp/g`, which contains the HBES IoT device *Group Object* configuration (see 5.6.7).

A new item shall overwrite the entire item (incl. "ga", "cflag", etc.) in the case an item with the same "id" already exists on the device.

However, the MaC shall not configure multiple *Group Object* items with the same "href" Point reference but different "cflags" settings, as shown in Figure 10, since this functionality is not supported in non-IoT EN 50090 compatible HBES devices.

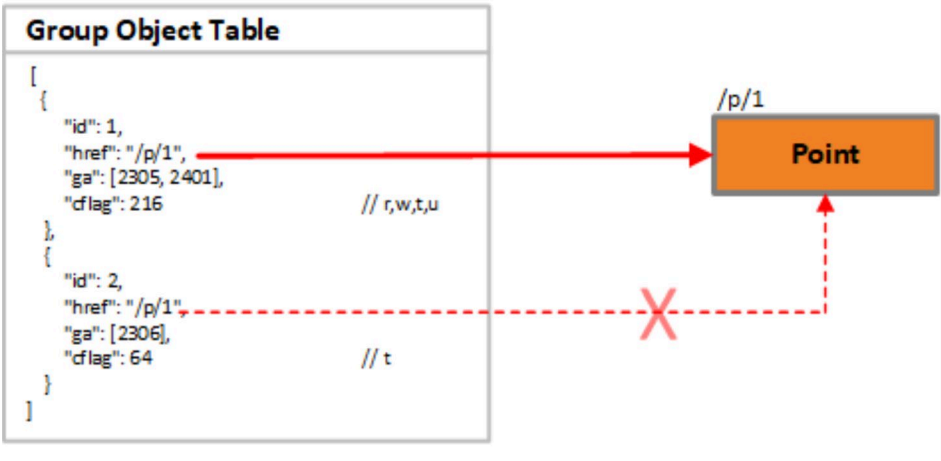


Figure 10 — Group Object Table Misconfiguration

5.5.7.4 Function Point Recipient Table Resource (fp/r)

5.5.7.4.1 Definition

The Function Point Recipient Table contains the destination address of outgoing group events (see 5.3.4.2). This table shall be writable and is configured by the MaC. The Function Point Recipient Table shall support the following endpoint configurations:

- IPv6 unicast destination address of an HBES IoT Recipient device that provides a CoAP endpoint.
- or an *Individual address* or FQDN that can be resolved to an IPv6 unicast address of an HBES IoT Recipient device.
- or IPv6 multicast addresses to send a message to multiple HBES IoT Recipients.

Table 20 specifies the mandatory ("M") resources and the corresponding resource path names that shall be used for the Function Point Recipient Table.

Table 20 — Function Point Recipient Table resource path and methods

Resource path	rt & Data Types	Format	Method	Support	Request/Response	Notes
fp/r	NA	cbor	POST	M	Req: Content- Format: application/cbor Payload: (see example below)	Write Function Point Recipient Table. Single elements are identified by their "id". Existing elements are replaced, and not existing elements are added. An empty element (only "id") in the payload shall cause the deletion of a corresponding existing element.