



BSI Standards Publication

Home and Building Electronic Systems (HBES)

Part 6-3: 3rd Party HBES IoT API

National foreword

This British Standard is the UK implementation of EN 50090-6-3:2023.

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 2023
Published by BSI Standards Limited 2023

ISBN 978 0 539 21681 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 31 July 2023.

Amendments/corrigenda issued since publication

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

English Version

Home and Building Electronic Systems (HBES) - Part 6-3: 3rd Party HBES IoT API

Systèmes électroniques pour les foyers domestiques et les
bâtiments (HBES) - Partie 6-3 : API tierce IdO HBES

Elektrische Systemtechnik für Heim und Gebäude (ESHG) -
Teil 6-3: Drittanbieter ESGH IoT API

This European Standard was approved by CENELEC on 2022-12-22. 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	3
1 Scope.....	4
2 Normative references	4
3 Terms, definitions and abbreviations	4
3.1 Terms and definitions.....	4
3.2 Abbreviations	10
4 HBES IoT 3rd Party API	11
4.1 Introduction	11
4.1.1 Mapping of terms	11
4.1.2 General.....	12
4.1.3 API - Content	13
4.1.4 API - Conformance.....	13
4.1.5 API - Version.....	14
4.1.6 API – Availability	14
4.1.7 API – Data Format	14
4.1.8 API – Examples Format	14
4.1.9 Motivation	14
4.1.10 Resources	15
4.2 Specifications.....	16
4.2.1 Server requirements	16
4.2.2 Client requirement	34
4.2.3 Message data structure	34
4.2.4 URL structure	52
4.2.5 Resource path definitions	53
4.2.6 Discovery	121

European foreword

This document (EN 50090-6-3:2023) 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) 2024-01-07
implemented at national level by publication of
an identical national standard or by
endorsement
- latest date by which the national standards (dow) 2026-07-07
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 defines a 3rd Party API for the Home and Building HBES Open Communication System.

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 50090-6-2, *Home and Building Electronic Systems (HBES)- Part 6-2 IoT Semantic Ontology model description*

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 terminological 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* (executed by a specific *procedure*, with an expected *result*) that changes an Installation state during Runtime

Note 1 to entry:

- The term Actuator can be mapped to `sosa:Actuator` in the SSN Ontology.
- The subject *actuation* can be mapped to `sosa:Actuation` in the SSN Ontology.
- The subject *procedure* can be mapped to `sosa:Procedure` in the SSN Ontology.
- The subject *result* can be mapped to `sosa:Result` in the SSN Ontology.

3.1.2

Application Function

function that uses a set of Functions to achieve the desired behaviour 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 with their input and output Datapoints that are logically connected to each other. The Functional Blocks may be located in one or more devices.

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

Note 2 to entry: The Application Function and Application are meant to be the same. Reason to introduce an alias term is to use a clear (understandable) reference from Application/ Application Function to the corresponding KIM class:ApplicationFunction or to the Function in the Management Client.

3.1.3

aspect

generally, a specific perspective on a system that contains things with different properties; a referencing mechanism to organize KIM elements in a specific perspective

EXAMPLE A Function Point is an ex officio Aspect with an important specific perspective. It is a referencing mechanism to organize together all to a Function Point interoperating Points (all GOs linked to a GA).

3.1.4

BIM

Building Information Model

digital process to describe and document a building in all its life cycle phases, from its planning, construction, operation up to its demolition

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 or may be an expansion above and beyond defined Functional Blocks or may be independent of a Functional Block if none is defined for the function associated with the Channel. The concept of a Channel is well-understood by the market participant, e.g. installers.

3.1.6

datapoint

represents 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

Note 1 to entry: In case of implementation as a Group Object, state data are communicated with the use of Function Points.

Note 2 to entry: The term Datapoint is the common term; to specifically denote a Datapoint available on an IoT 3rd Party API, the term IoT Datapoint is used.

3.1.7

device

physical element that is part of the network; it is a physical, concrete object that a customer can buy

3.1.8

endpoint

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

3.1.9

Feature of Interest

abstraction of a real-world thing (phenomenon, equipment, person, event...) defined by its observable or actuatable properties

Note 1 to entry: In colloquial terms, a FOI is a property carrier.

Note 2 to entry: A Sensor operates on a FOI with observable properties, an Actuator with actuatable properties.

Note 3 to entry: A FOI is not a "classification/type" tag itself; the "classification/ type" is accomplished with the help of tags. Examples are defined in 4.5.1.4.

3.1.10

function

describes a part of the intended behaviour of a FB in a building context

3.1.11

Functional Block

consists of one or more Functions that belong together and that cannot be separated across two devices but big enough that a device with only one such Functional Block could be marketed

Note 1 to entry: A Functional Block has a well-defined black box behaviour.

3.1.12

Function Point

runtime system state information of a specific Application Function

Note 1 to entry: Shared by at least two Datapoints.

Note 2 to entry: Has a unique identifier that addresses a group of controlled objects. This identifier is called a Group Address.

EXAMPLE < Light Switch > in living room 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 IoT, this can be realized by simple UDP communication or by using a message broker system or other.

3.1.15

Group Object

foreseen for Group Communication using Group Address(es)

Note 1 to entry: May be accessed via point-to-point communication without an assigned Group Address. With assigned Group Address, it becomes a member of that Function Point represented by the Group Address.

3.1.16

HBES Information Model

ontology based model of HBES System relevant parts, including additional semantic (dictionary) information

Note 1 to entry: It is managed by the HBES Association, hence the abbreviation KIM.

3.1.17

Industry Foundation Classes

open standard to describe BIM data in a digital way

Note 1 to entry: IFC data and models are specified in ISO 16739-1.

3.1.18**installation**

assembly of materials and components (devices) placed in position to provide a service

Note 1 to entry: An Installation is a deployed system (e.g. HVAC system or fire protection system) and consists of equipment and Functions that are used for a particular purpose.

Note 2 to entry: In relation to this term created data correlates to the installation model, described in 4.2.

[SOURCE: ISO 6707-1:2020, modified – added “(devices)” and Notes to entry.]

3.1.19**IoT Datapoint**

represents an Endpoint at an IoT 3rd Party API that:

- a) corresponds to one or more Function Points, such as a state data representation of a discrete state in a building context:

EXAMPLE 1 brightness → discrete state “brightness” is represented by the value 65 (percent)

- b) is a fully qualified URL e.g. provided by an IoT 3rd Party Server

EXAMPLE 2 `https://gateway.knx.local/knx/api/v1/datapoints/{Id}`

3.1.20**IoT Function**

represents a Function at an IoT 3rd Party API that:

- is as a collection of IoT Datapoints that fulfils a – by the user – intended behaviour

EXAMPLE “living room – rear light dimming”, “kitchen – floor heating”

Note 1 to entry: In a Mac, an IoT Function is instantiated data of a MaC Function in an Installation respectively MaC project. The MaC Function itself may base on an Application Function.

3.1.21**IoT 3rd Party API**

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

3.1.22**IoT 3rd Party Client**

device or service interacting with the Installation from outside using the IoT 3rd Party API

Note 1 to entry: The IoT 3rd Party Client connects to a single device that provides the IoT 3rd Party API and can use this single device to fully interact with the Installation, possibly depending on a specified authorization mechanism.

EXAMPLE 1 A mobile phone (from inside the network, or from an Internet connection) with typically short period connections.

EXAMPLE 2 A weather service permanently feeding in its weather information using the IoT 3rd Party API.

3.1.23**IoT 3rd Party Server**

device that implements the IoT 3rd Party API

Note 1 to entry: This can be a dedicated device; this can be a function of a device that supports other HBES IoT and non HBES functionalities; it may be located within the local LAN of the IoT installation or outside.

3.1.24

MaC Catalog Entry

created management client data correlating to the product model, described in 4.2

3.1.25

MaC Function

Application Function created by the MaC and assigned to a building structure element, grouping several Group Addresses

3.1.26

MaC Project

project created by a MaC documenting the Configuration of an Installation

3.1.27

Management Client

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

Note 1 to entry: The MaC is used to configure and commission Devices, as well as to plan, design and diagnose an entire Installation. As a final step the MaC writes specific configuration data such as Device parameters to the Devices.

3.1.28

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

3.1.29

Object Property

in OWL a built-in concept that **connects** pairs of individuals, an object property expression **represents** the (entire) relationship between the pairs of individuals

3.1.30

OWL

OWL 2 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 document 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: This document uses the term Point as an umbrella for data that can be accessed from outside of the Device, for instance to interact with other Points from other Devices. Consequently, term Point is a generic superset of the term Datapoint (which describes 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 [RFC7228] supporting device individualization, device linking and accessing device runtime data (e.g. Functional Block or Channel Datapoints)

3.1.33**Quality Kind**

represents a certain combination of observable or actuatable properties, available as predefined parts of the Semantic Dictionary or created individually during Configuration; the latter is the case when a Quality Kind with the intended combination of properties respectively tags is not (yet) part of the dictionary

Note 1 to entry: A QK is not a “classification/type” tag itself; the “classification/ type” is accomplished with the help of tags. Examples are defined in 4.5.1.4.

3.1.34**RDF**

Resource Description Framework, as specified by the <https://www.w3.org/RDF/>

Note 1 to entry: RDF is a framework to represent information in the web by using triples. The information can be serialized and stored in many formats such as the TURTLE or JSON(-LD) format. The general RDF concept description can be found under <https://www.w3.org/TR/rdf11-concepts/>

3.1.35**runtime**

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

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

3.1.36**Semantic Export**

project exported by the MaC reflecting an Installation in a linked data format

Note 1 to entry: The exported data are:

- structured according to the KIM, such as using Object Properties defined in KIM;
- annotated with additional semantic information from the Semantic Dictionary;
- referencing concepts of external Ontologies.

3.1.37**semantic dictionary**

set of standardized terms allowing to annotate required parts of an Installation

Note 1 to entry: For details, see 4.2.8.

3.1.38**sensor**

point performing an *observation* (executed by a specific *procedure*, triggered by a *stimulus*), responding a *result* as an Installation state during Runtime

Note 1 to entry:

- The term Sensor can be mapped to `sosa:Sensor` in the SSN Ontology.
- The subject observation can be mapped to `sosa:Observation` in the SSN Ontology.
- The subject stimulus can be mapped to `ssn:Stimulus` in the SSN Ontology.
- The subject *procedure* can be mapped to `sosa:Procedure` in the SSN Ontology.
- The subject *result* can be mapped to `sosa:Result` in the SSN Ontology.

3.1.39

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)

Note 1 to entry: A Tag is a concept-less term, without an integration in a broader concept such as the concept of a Datapoint (used in an Application Function), it has a limited semantic meaning.

EXAMPLE Term “flow” has a weak meaning on its own, but if you relate it in a FOI with the other term “water” this expresses at least that you observe/ actuate the water flow.

In this document a Tag is almost exclusively a term from the Semantic Dictionary.

3.1.40

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>

Note 1 to entry: TD relevant relations are described in the clause of Semantic Export.

3.2 Abbreviations

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

API	Application Programming Interface
GA	Group Address
GO	Group Object
FB	Functional Block
FP	Function Point
HBES	Home and Building Electronic Systems
IFC	Industry Foundation Classes
IOO	Info On off
IO	Input Output
IoT	Internet of Things
KIM	KNX Information Model managed by KNX Association and described in EN 50090-6-2
KNXA	KNX Association
M	Mandatory
MaC	Management Client
OP	Object Property
O	Optional
TD	Thing Description

4 HBES IoT 3rd Party API

4.1 Introduction

4.1.1 Mapping of terms

The KNX Information Model expresses things that correspond to or reflect the wording and meaning used in the IoT world.

NOTE In the HBES community, the terminology has evolved and was developed over time having its origins more than 20 years ago. By making HBES IoT Installations accessible for customers via an open API, there is a need to intuitively adapt the language to make it understandable and usable by IoT developers. Unfortunately, it is not possible to preserve the historical HBES terminology and use it “as is” when addressing external non-HBES developers.

The following Table 1 provides a mapping of terms used in the KNX Information Model, HBES IoT 3rd Party API and comparing it to the HBES-internal naming convention.

Table 1 — Mapping of terms

Customer <i>API Client Development</i>	HBES IoT <i>Server Development</i>	KIM	HBES Classic <i>Installer</i>
Function	IoT Function	<i>:ApplicationFunction</i>	Function
Datapoint0F	IoT Datapoint	<i>:FunctionPoint/:Datapoint</i>	Group Address/Group Object
building/floor/room/site	building/floor/room/site	<i>loc:Building/ loc:Floor/ loc:Room/ loc:Site</i>	Building/Floor/Room/-
NA	NA	<i>:Channel</i>	Channel
Device	Device	<i>:Device</i>	Device

- The *Customer* terminology is relevant for an HBES IoT 3rd Party Client respectively for a developer building such a client based on the HBES IoT 3rd Party API. This developer is typically HBES agnostic.
- The *HBES IoT* and *KIM* terminology addresses the HBES manufacturer, building a HBES IoT 3rd Party Server, respectively is relevant for KNX Association itself specifying the Ontology.
- The *HBES Classic* terminology is well known for an installer of HBES S-Mode installations.

For a server development it is of essence to understand how to differentiate the terms from the table above from a technical perspective, for this see below.

Relations

HBES Classic makes use of the concept of a Group Address and Group Object: a **Group Address** links several **Group Objects** (a **Function** groups at least one **Group Address**, a **Device** has at least one **Group Object** assigned).

However, both a GA and GO are represented with a **Datapoint/ IoT Datapoint**.

Possibility to distinguish between a GA/ GO at the level of a **Datapoint/ IoT Datapoint**,

- a) At level of the HBES IoT Server Development

Use of the corresponding KIM concepts for a GA or GO according the table above. Moreover, a Function Point also contains a property for the GA value.

b) At the level of the API Client Development

In the case where the server includes as part of the response the type information from the KIM concepts above and/or the optional property for the GA, with this it will be possible to distinguish a **Datapoint/IoT Datapoint** as being a GA or a GO in a KNX Classic Installation.

For details on how these relations are semantically expressed, refer to EN 50090-6-2.

4.1.2 General

The HBES IoT 3rd Party API shall support the following features:

- discovery of the HBES IoT 3rd Party Server
- reading and writing of End points
- setup notifications on End points
- access permission control of End points

Information about reliability and freshness of the Installation state data are not dealt with in this version of the HBES IoT 3rd Party API.

The HBES IoT 3rd Party API hosts in the HBES IoT 3rd Party Server the main server project and Installation concepts as depicted in Figure 1. The picture expresses the following aspects:

- with red dotted lines: the available relations in the API (which are fewer than the EN 50090 System or KIM describe, see explanation on D below)
- the item cardinality at the end of every relation (in UML notation)

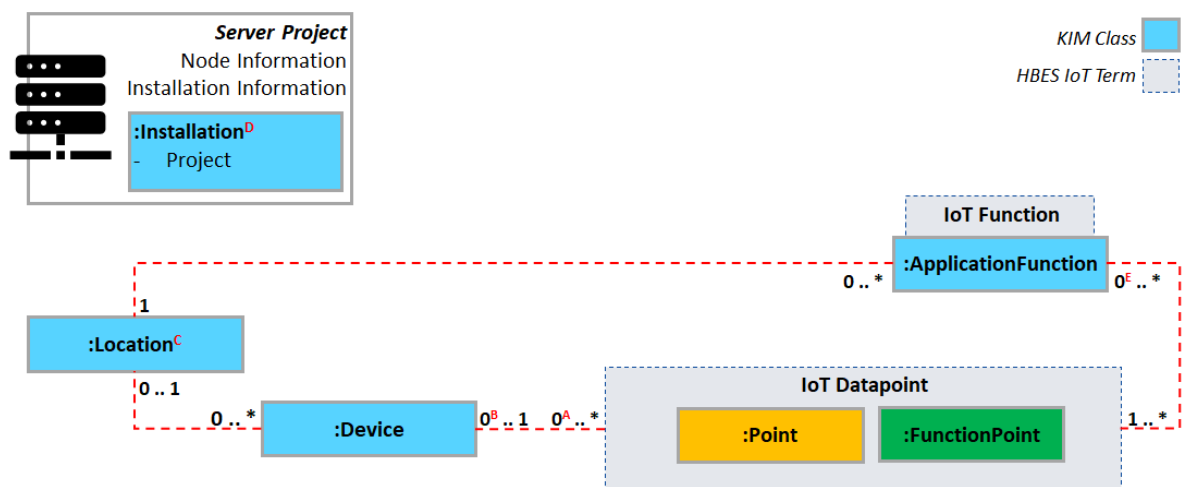


Figure 1 — Project elements in the HBES IoT 3rd Party Server

The letters in the above Figure denote the following:

- A. when a Device does not support a Point (plain hardware like power supply, without any application) or when the IoT Datapoint represents a Function Point
- B. when an IoT Datapoint represents a Function Point
- C. includes floors/rooms/... and other

- D. an Installation to location relation is not an explicit relationship in the API; it is expected that the resource path definitions of the category location (or similar) are always the known and fixed entry point for one (or more) Installations. In case of more installations, all content is merged, the API end points delivers a common perspective, see also 4.2.5.1.3.
- E. when an IoT Datapoint represents a Point

Server Project - Node Information

Provides node meta information assigned to the server: geographically assigned location address, node (firmware) version and more

Server Project - Installation Information

Is the MaC project assigned to the server, in the form of the Semantic Export.

- Functions, Datapoints, Devices or Location elements such as a Building, Floor or Room.
- The version of the MaC Project, to track changes of the server project configuration. To identify a change the OP `:lastModified` of the `:Installation` can be used, see “Installation” in EN 50090-6-2).
- The HBES Information Model version (see the “Introduction” in EN 50090-6-2), as part of the Semantic Export.

NOTE Data and users are added to the actual server project in the HBES IoT 3rd Party Server in a plug-and-play way. Consequently, the server project is a superset/ extension of the original MaC project. Therefore, any additional data of the server project needs to be maintained by the HBES IoT 3rd Party Server and not by the MaC.

4.1.3 API - Content

The HBES IoT 3rd Party API consists of:

- Resource access methods (to set or retrieve Installation state data).
- End points hosting concepts comprising the Runtime communication or representing the Configuration of a (HBES Classic) Installation, additionally end points allowing to setup notifications on changes of Installation state data, provided to subscribers that are clients to the Installation.
- Methods to filter for specific resource items from a collection of resource items.
- Methods for authorization by an HBES IoT 3rd Party Client including a secure client/server communication.
- Access (permission) scopes, for security reasons the actual access to some end points (such as writing a Datapoint) is gated by the HBES IoT 3rd Party Server, this access will be granted as part of the authorization.

For all End points, their expected request/ response document formats, and their content is not described to the full extend in this document. Whether an element is mandatory or optional can be found in the electronic document (see 4.1.6). If an End point is mandatory or not, refer to 4.2.5.

NOTE Except by means of written text, Open API does not allow to define with a statement that an End point is optional.

4.1.4 API - Conformance

The HBES IoT 3rd Party API includes the above-described content and consists of the following:

- The API electronic document content is described with Open API (OAS3).

- The API concepts are described according JSON API, in case of deviations compared to the JSON API this is mentioned in this document. As a rule, an HBES IoT 3rd Party Server shall
 - ignore any valid JSON API content that is not supported or expected by the server;
 - respect the JSON API standard when supporting other members, even if they are not detailed in this document.

EXAMPLE JSON API top-level members not mandated or described in this document, such as top-level links or `included` member.

EXAMPLE HBES manufacturer attributes not defined in this document, as part of the attributes member of a certain resource object.

EXAMPLE Providing a single JSON object or an array of JSON objects, including if no resource items are available.

4.1.5 API - Version

The HBES IoT 3rd Party API shall be versioned to allow compatibility between communication peers, however only as a whole and not in parts, even if several elements of the API are unchanged between different API versions.

The version of this document (see clause document updates) relates to the version 2.0.0 of the electronic document (see 4.1.6). For details how to handle API versioning see 4.2.5.1.1.1.

4.1.6 API – Availability

The URL <https://schema.knx.org> allows to retrieve the most recent electronic document of the HBES IoT 3rd Party API. Older versions are also available.

The above-mentioned URL has no relationship to the common HBES IoT 3rd Party API URL structure, for this see 4.2.4.

4.1.7 API – Data Format

The HBES IoT 3rd Party API electronic document is available as a YAML file.

4.1.8 API – Examples Format

For readability, in this document all HBES IoT 3rd Party API code (snippet) examples are in the YAML format, regardless of the actual format at Runtime.

4.1.9 Motivation

The motivation of this document is to ensure that the HBES IoT 3rd Party API

- can be easily and consistently consumed by clients with basic HTTP support;
- provide smooth developers experience, in this way making use easy and intuitive.

Services typically provide language-specific frameworks to wrap their APIs, most of their operations eventually boil down to requests/ responses with JSON payload.

Most modern APIs use HTTP protocol concepts, blended with some concepts from other computing technologies.

NOTE A lot of web APIs are defined in terms of End points that have parameters. End point and parameters are not terms or concepts that are native to HTTP or REST- they are concepts carried over from Remote Procedure Call (RPC) and related technologies.

An API should focus on the underlying entities of the “domain of interest” it exposes, rather than a large set of functions that manipulate those entities. Fewer unique features of the API have to be

managed over time. This belief is really an expression of the more fundamental idea to specify only the fewest number of concepts necessary to solve the problem.

Fewer concepts result in increased consistency, allowing teams to leverage common code, patterns, documentation, and design decisions. These guidelines aim to achieve the following:

- Allowing 3rd parties to integrate service solutions with the products of HBES manufacturers implementing the HBES IoT 3rd Party API.
- Providing easy access to the HBES IoT services via REST interfaces for application developers.
- Making the integration of the HBES IoT 3rd Party API easy for device developers.

4.1.10 Resources

4.1.10.1 General

A resource is the nucleus of any rest-oriented API design, it models and represents the intended Application's Domain. With an URI, a resource is uniquely defined, an URL defines a method to access the resource.

Resources shall generally be described with an URI.

4.1.10.2 Items, collections and relationships

An API is typically designed by dividing resource types (naming schemes) into documents, collections, stores and controllers. In this document the focus lies on documents and collections.

- A document (from now on called item) is one single or individual "object of interest".
- A collection is a directory of singular items, hosted in a server.

Items, and collections (of items) are the first fundamental concept.

In addition to simple items, most "domains of interest" include conceptual relationships. The natural construct for representing relationships in HTTP is the link. Links indicate where you can go from where you are. They are the signposts for the paved paths to traverse the system.

Relationships (linked data) are the second fundamental concept.

Examples can be found further down.

4.1.10.3 Resource path

The following path structure to a resource should be defined, a path to retrieve an individual item of a specific resource using a resource {id} and one path to retrieve a collection of such items.

- item path .../item/{id}
- collection path: .../item (same resource path principle is .../item/{id}/item)

Hyphens shall be used to separate path segments, such as .../segment1-segment2/.

4.1.10.4 Resource names

The use of a domain specific nomenclature for resource names helps developers understand the functionality and basic semantics of resources. It also reduces effort to document the API.

A collection provides usually multiple items, such resource names should be pluralized.

```
/buildings
/buildings/{id}
```

URLs are often treated case sensitive, therefore lowercase shall be used.

4.1.10.5 Resource IDs

JSON API requires for the {id} a string format, it is recommended for the {id} to use the UUID format.

To simplify the encoding of a resource id in URLs the {id} shall not use reserved characters as specified in [RFC 3986] section 2.2.

In case the resource {id} is represented by a UUID the above rule is fulfilled automatically, as a UUID never uses any reserved character specified above.

4.1.10.6 Examples

The resource path examples below show a path to a single location and function item and a resource path for a collection of locations, ids are formatted as UUID resource {id}.

/locations/{id}	A specific location item as part of the locations collection in an Installation, https://.../locations/3a19fee4-ba8a-4fd2-ba10-42b7f40f5acf
/functions/{id}	A specific function item as part of the functions collection in an Installation, https://.../functions/7e1418b3-58cb-43dc-a743-1280bc2c6bd7
/locations	A collection of all locations in an Installation, https://.../locations

The resource path example below shows a relationship of a location to its functions included there.

/locations/{id}/functions	A collection of all function items in a specific location, https://.../locations/3a19fee4-ba8a-4fd2-ba10-42b7f40f5acf/functions
---------------------------	--

4.2 Specifications

4.2.1 Server requirements

Use of common web techniques (e.g. additional HTTP header tags, used cypher suites on TLS, ...) that go beyond those that are specified in this clause and in 4.2.5.8 are acceptable, provided they do not violate any of the explicit protocol and security requirements.

4.2.1.1 Protocols

4.2.1.1.1 IP

The HBES IoT 3rd Party API communication shall be possible on both IPv4 as well as on IPv6. The implementer is free to choose the version; dual stack implementations are possible as well.

4.2.1.1.2 HTTP

A HBES IoT 3rd Party Server shall support HTTP/1.1 as documented in the following RFCs:

- [RFC 7230] – *Message Syntax and Routing* → general purpose
- [RFC 7231] – *Semantic and Content* → Content-Type header

A HBES IoT 3rd Party Server may support HTTP/1.1 as documented in the following RFCs:

- [RFC 7232] – *Conditional Requests* → ETag header

HTTP/2 with [RFC 7540] is recommended but optional.

This document lists some of the required elements of the above mentioned *Semantic and Content* elements below.

- handle different HTTP resource access methods;

- handle different (URL) resource paths;
- handle possible (URL) resource path query parameters;
- handle possible request/ response headers;
- handle a HTTP version number.

Even if requirements laid down further in this document may refer to HTTP/1.1, it is explicitly supposed that such requirements are platform-agnostic to HTTP/2 or HTTP/1.1.

NOTE In this document further used examples use HTTP/1.1.

4.2.1.2 Access Method

Resources can be accessed with the standard HTTP access methods for restful services.

Readable resources can be fetched by sending a `GET` request to its corresponding End point. Writable resources can be updated by sending a `PUT` request to its corresponding End point. In specific cases a writable resource can be accessed also with `POST(create)`, `PATCH (update)` and `DELETE` requests.

Which HTTP methods are used in which use case,

consult the HBES IoT 3rd Party API standard in 4.2.5. In case a client is using an access method not defined for an individual End point, the HBES IoT 3rd Party Server shall respond with error code 405.

4.2.1.3 Header

The HBES IoT 3rd Party Server shall support to handle the following headers specified in HTTP.

- `Accept`

Client request header, containing a specific requested `Content-Type` respectively `media-type`.

- `Accept-Charset`

Client request header, containing a specific requested `Accept-Charset`.

NOTE Well known browser clients generally omit this header.

Above-described header behaviour is closely related to the content negotiation handling, see 4.2.1.7.

- `Authorization`

Client request header, containing specific user authorization credentials.

- `Host`, `Content-Length`, `Date`

Used for subscription post notifications, see 4.2.5.6.4.6.

The HBES IoT 3rd Party Server may support handling of the following header specified in HTTP.

- `ETag`

Server response header, as described in 2.3 in [RFC7232]

NOTE `ETag` is a caching mechanism for reliability and scalability and an indicator for domain respectively application data freshness/validity. It affects the server response, containing a specific resource identifier (version or fingerprint).

Client requests to specific End points of the HBES IoT 3rd Party API with other request headers as specified above (such as If-None-Match) are out of scope of this document.

4.2.1.4 Query parameters

This document uses the following parameter types:

- query (parameter is part of the URL, see clauses below)
- path (parameter is part of the URL, described as part of the resource path, see 4.2.5)
- header (parameter is part of a HTTP header)

This document describes optional and mandatory query parameters, the used query parameter names fulfil the JSON API. Other (vendor specific) query parameters are optional.

In case square brackets “[]” and the colon “:” are used as part query parameter names, they shall be %-encoded. For the general name definition for query parameters see JSON API, section Query Parameters.

The HBES IoT 3rd Party API Server shall support a query parameter string according to [RFC 3986], section 3.4. For examples, see in next clauses.

- Several query parameters shall be concatenated with an ampersand “&” separator, each query parameter shall be handled by the server. The concatenation shall not be handled with any logical operation for the parameters, for this see 4.2.1.4.1 and 4.2.1.4.2 below.
- Several query parameter values for the same query parameter shall be supported as a comma separated list. The list shall not be handled with any logical operation for the values, for this see 4.2.1.4.2 below.

The query parameter consists of a parameter **name** and a parameter **value**, separated by a “=”. Technically this is a key/value pair used to search for resources related to the key, also matching the requested (parameter) value.

NOTE The server technology to define such relations is out of scope here (SQL database, triple store, ...).

1. The query parameter **name** (key) is defined as a string, covering a set of terms, see next clauses.
2. The query parameter **value** is defined as a string, covering any term including timestamps and numbers. Depending on the related resource item member the value needs to be interpreted in different ways. The parameter value may be surrounded with quotes, which needs to be ignored by the HBES IoT 3rd Party Server.

NOTE The Datapoint `attributes` member value is a possible query parameter, see also examples in the next clauses below. It is defined as a JSON string, see 4.2.5.4. A query parameter value `4.10` is interpreted on a string-based DPT as a string `4.10`, on a number-based DPT as a number `4.1`.

String

String values are interpreted by %-decoding the given value.

Number

Numbers are interpreted according to JSON, Clause 6.

Time

Time is interpreted according to the API reference member timestamp.

IRI

IRI is interpreted the same as for **String** above. They cover KIM properties, optionally external ontology properties. Using a full IRI as value is not recommended, for this the RDF namespace IRI concept should be used (further called namespace prefix).

The corresponding namespace prefix can be omitted as part of the client request. The HBES IoT 3rd Party API Server shall always prioritize the KIM properties (see examples in next clauses).

NOTE The option to omit a namespace prefix such as `dic-tag:`, `dic:`, `loc:` or `unit:` means that the HBES IoT 3rd Party API Server shall implement different priorities to search for related content.

1. in KIM included properties, with and without namespace prefix
2. Vendor specific properties

When a vendor specific or external Ontology (non-KIM) property has the same value as in the KIM, a client shall indicate this with the corresponding namespace prefix. Otherwise, it is handled by the server as the KIM property (in such a case the server does not realize that the query parameter was intended to be a different one).

The server behaviour is unspecified in case two KIM properties exist with the same name when dropping both prefixes.

A user visible translation for the query parameter **name** and **value** (including specific parts of the name such as [operator]) shall be handled by the client.

4.2.1.4.1 Pagination

Constrained server/clients may have problems to handle a large amount of collection items in a response. For this the HBES IoT 3rd Party API Server supports a page-based pagination strategy.

- An HBES IoT 3rd Party Client may limit the expected amount of responded items from a collection with a specific “collection offset” or “response size” to finally read all items of a resource collection item by item.
- A HBES IoT 3rd Party Server may limit the maximal number of items provided in a response. The limitation may depend on the server (resource) capabilities and the expected number of items in a collection (closely related to the size of a MaC project).

The following **query** parameters shall be supported by the HBES IoT 3rd Party Server.

- `page [...]` paginate the response data to a given criterion, “...” values see clauses below

If different (pagination) query parameters are concatenated in a single request each shall be met individually, the parameters are not logically and'ed.

4.2.1.4.1.1 page[number]

The parameter skips several pages before starting to collect the result set of items. It occurs only once per request, the HBES IoT 3rd Party API Server shall otherwise raise an error (see 4.2.1.4.4).

This filter is applied to returned resource collections of resource path category **function**, **location**, **datapoint**, **device**, **information**, **subscription** or **timeseries**, for this see the electronic document (see 4.1.6).

- The parameter **name** is specified with `page[number]`.
- The parameter **value** is a number.

The default parameter value for the HBES IoT 3rd Party Server shall be zero.

EXAMPLE A client request without a query parameter of **page[number]** would not be able to read all items that may be in front of the default value – if not set to zero

In case a query parameter was not set in the client request accessing a collection, the HBES IoT 3rd Party Server shall respond with the default parameter value (as specified above) as part of the `collection` member (see 4.2.3.2.1).

The server value for the size of a page as such is vendor specific.

NOTE Addressing a collection item (as a zero-based offset) → `number` = offset / page size

4.2.1.4.1.2 `page[size]`

The parameter defines the maximal number of items to return in the response. It occurs only once per request, the HBES IoT 3rd Party API Server shall otherwise raise an error (see 4.2.1.4.4).

This filter is applied to returned resource collections of resource path category ***function***, ***location***, ***datapoint***, ***device***, ***information***, ***subscription*** or ***timeseries***, for this see the electronic document (see 4.1.6).

- The parameter **name** is specified with `page[size]`.
- The parameter **value** is a number.

The default parameter value for the HBES IoT 3rd Party Server shall be set larger than zero, the actual value is vendor specific.

NOTE A client request without the query parameter `page[size]` would return an empty list - if set to zero.

In case a query parameter was not set in the client request accessing a collection, the HBES IoT 3rd Party Server shall respond with the actual parameter value (considering the default parameter value) as part of the `collection` member (see 4.2.3.2.1).

NOTE The actual (size) parameter value in a response may be lower than the default (size) parameter value if the last page does not contain enough items. If the default (size) parameter value is larger than the number of items in a collection, the response contains the entire collection.

EXAMPLE A (%-encoded) GET request fetches all location items from the `/locations` collection, starting from `page[number]` "20", returning the below collection information.

Request

```
GET /locations?page%5Bnumber%5D%3D20 HTTP/1.1
Accept: application/vnd.api+json
```

Response

```
meta:
  collection: // a response with collection information, see 4.2.3.2.1
  number: 20 // client requested to skip 20 pages, return from page #21
  size: 10 // no size limitation from client, use server default value
(10)
  total: 400 // a client may derive from the value above a server page size
10
data: [{...}]
```

4.2.1.4.2 Filter

A client may filter for specific item properties in collections. For this the HBES IoT 3rd Party API Server supports a filter strategy.

- An HBES IoT 3rd Party Client wants to limit the resource items in a response such as to present only items matching a specific user search criterion, all others from the collection are of no interest to the client.

The following **query** parameters may be supported by the HBES IoT 3rd Party Server.

- `filter[...][operator]` filter resources according to a given filter criterion and operator, for the “...” and *operator* options see clauses below

This filter is applied to returned resource collections of resource path category **function**, **location**, **datapoint**, **timeseries** or **device**, for this see the electronic document (see 4.1.6).

If different (filter) query parameters are concatenated in a single request each shall be met individually, the parameters are logically and'ed.

NOTE An or'ed filtering can be realized with multiple requests.

4.2.1.4.2.1 filter[...][operator]

The parameter name part *operator* filters for specific resource items as part of a collection that are related to a logical/comparison operation, such as all *datapoints* with a temperature (attribute) value lower than 30°C.

- The parameter name **filter[...][operator]** is specified for the part *operator* as follows:

Table 2 — Specification of parameter name filter for part operator

Op.	Definition	True Rule	Query parameter value list
eq	equal	resource item member value = query parameter value	not allowed, see error handling
le	lower or equal	resource item member value = < query parameter value	not allowed, see error handling
ge	greater or equal	resource item member value = > query parameter value	not allowed, see error handling
lt	lower than	resource item member value < query parameter value	not allowed, see error handling
gt	greater than	resource item member value > query parameter value	not allowed, see error handling
and	logical and	all requested query parameter values are contained in a list of a resource item member ^a	allowed
or	logical or	at least one requested query parameter value is contained in a list/ matches the value of a resource item member	allowed

^a As implementer's note, it is also false when addressing a resource item member with a (single) value or a list with not enough comparable elements included in it.

The order to use for the comparison operator **le**, **ge**, **lt**, **gt**, **eq** is defined as follows:

1. chronological order if both values are formatted as a time (see **Time** in 4.2.1.4),
2. numerical order if both values are formatted as a number (see **Number** in 4.2.1.4),
3. otherwise in lexicographical order.

For the points 1 and 2 above: Except for the datapoint value from the attribute member value, here the corresponding DPT shall be used instead (such as a string “4.10” on a string DPT or a float “1.3E-3” on a float DPT).

The parameter name part *operator* is optional, if not present this is interpreted by the server as the default eq.

4. The parameter **value** is specified for each [...] part in the next clauses.

EXAMPLE Logical filter expressions in different variants.

1. `filter[value][le]=30`
// datapoint (runtime) attributes member value shall be lower/equal than 30
2. `filter[timestamp][gt]=2021-02-17T17:17:13Z`
// datapoint shall be last updated strictly after 2021-02-17, 5:17:13PM
3. `filter[state][or]=Tested,Accepted`

```

// location attributes member state shall be "Tested" or "Accepted"
4. filter[enum][and]=on,off
   // datapoint attributes member enum shall contain "on" and "off"
   // → this is a valid combination for the DPT 1.001
5. filter[unit][and]=DEG_C,DEG_F
   // datapoint attributes member unit shall contain "DEG_C" and "DEG_F"
   // → such a DPT does not exist
6. filter[unit]=DEG_C&filter[value][lt]=30
   // datapoint attributes member unit shall be "DEG_C" and the value shall be
   // lower than 30
   // several and'ed parameters, each of them shall be handled and met
   // individually

```

4.2.1.4.2.2 filter[meta.@type][operator]

The parameter filters for specific resource items as part of a collection that are related to a specific resource type, such as all *datapoints* that are of the additional type loc:Floor.

NOTE Additional resource type definition in a response is indicated as part of the meta member, hence there will be no conflict with other members, such as the general type definition as part of the data member.

- The parameter **name** is specified with *filter[meta.@type]*, for the part *operator* as specified in 4.2.1.4.2.1.
- The parameter **value** is specified with a KIM class name or (if available) the URN assigned to this class.

The KIM classes for datapoint, device and function resources, their corresponding namespace prefixes and the actual class names can be found in EN 50090-6-2 respectively in the electronic file.

The KIM classes for location resources, their corresponding namespace prefixes and the actual class names can be found in Table 12.

NOTE The KIM class name is the remainder of the IRI when removing the corresponding namespace prefix, such as loc: from <http://schema.knx.org/2020/ontology/loc#Floor> → Floor.

NOTE Any (parent/ child) inheritance of location resources defined in Table 12 are not taken into account when filtering. As an example, filtering for all buildings will not include any room.

EXAMPLE Type filter expressions in different variants.

```

- filter[meta.@type]=Floor // KIM location type without namespace
- filter[meta.@type]=loc:Floor // KIM location type with namespace
- filter[meta.@type][or]=Floor,Space // locations either a Floor or Space
- filter[meta.@type]=Floor,Space // two parameter values, may cause a 422
- filter[meta.@type][lt]=Floor // types lexicographically lower than
// string Floor (may not be very useful)
- filter[meta.@type]=dpa.321.51 // KIM datapoint type without namespace
- filter[meta.@type]=dic:dpa.321.51 // KIM datapoint type with namespace
- filter[meta.@type]=urn:knx:dpa.321.51 // KIM datapoint type as URN
- filter[meta.@type]=td:Thing // external element with namespace
- filter[meta.@type]=oms:HeatMeter // external element with namespace
- filter[meta.@type]=switchinCtrl // KIM app. function without namespace
- filter[meta.@type]=urn:knx:fct.switchingCtrl // KIM app. function as URN

```

4.2.1.4.2.3 filter[tagKey][operator]

The parameter filters for specific resource items as part of a collection that are related to specific tags, such as a datapoint that is related to the (phenomenon) tag *temperature*, a building that is related to the (location usage) tag *airport* or an application function that is related to the (trade) tag *HVAC*.

- The parameter **name** *filter[tagKey]* is specified for the part *operator* as specified in 4.2.1.4.2.1, for the part *tagKey* with a structure of the KIM object property relationship predicates, as follows:

1. *hasTagSet.hasTag*

The structure expresses a combination of two KIM OPs, addressing Tags that are **indirectly** assigned to a source (such as a location or datapoint). The structure shall be used with the following OPs:

Table 3 — Parameter name including two KIM Object Properties

parent OP	hasTagSet	hasTag	Comment
child OP	— hasQualityKind — actsOnQualityKind — observesQualityKind — adjustsQualityKind — measuresQualityKind	— hasEquipmentType — hasPhenomenonType — hasLocality — hasQuantityKind	Addresses a Quality Kind.
child OP	hasProcessKind	— hasEquipmentType — hasPhenomenonType — hasVirtualEntityType — hasLocality — hasStateKind	Addresses a Process Kind.
child OP	No child properties defined.	— hasOrigin — hasPointOperation — hasPointFunctionType	Addresses an Operation Kind.

NOTE 1 A QK/PK or Operation Kind may have been assigned multiple times to a source, each structure from above needs to be considered separately, for this see examples below.

The structure cannot be split into individual parts, such to address QKs/PKs directly with the first OP.

NOTE 2 To address QKs/PKs directly, this would require introducing a structured naming convention for them. A parameter value with such a name would not indicate a meaningful range/structure to filter on a specific QK/PK, the actual interest is on the tags as such and not on the names of the containers holding the tags.

2. *hasTag*

The structure expresses a single KIM OP, addressing Tags that are directly assigned to a source (such as a location or datapoint). The structure shall be used with the following OPs:

Table 4 — Parameter name including one KIM Object Property

parent OP	hasTag
child OP	- hasPointInterface - operatesForTrade - operatesOnProtocol - hasLocationUsage

NOTE 3 As described further above some tags may be assigned only indirectly to a source, hence the filter result may be not as expected when using child properties of : *hasTag*, not listed in Table 4.

The described parent OPs have a dedicated range to all possible tag individuals of KIM class :Tag, when using them all tags will be filtered and included in the result. To limit the results, related child OPs need to be used, mainly from `hasTag`, but also from `hasTagSet` (or a combination).

NOTE 4 The above statement assumes that the server data needs to include child properties, otherwise the returned result is always empty.

If a chosen parameter name/ value combination does not match content wise or logically, the server may not be able to retrieve/ calculate a matching, for this see clause error handling below.

— The parameter **value** is specified with the name of a KIM individual of class :Tag.

The KIM individuals for the tag resources, their corresponding namespace prefixes and the actual individual names can be found in EN 50090-6-2 respectively in the electronic file.

A client needs to know the possible filter values. Common practice for a client is to learn them from the responses. Therefore, it is strongly recommended that a response contains the individual tags and/or the QKs/PKs as entire resource objects as described in 4.2.3.3.5. Other options are less transparent, for example to preprogram them in the client code or reading them externally (such as from the ontology).

NOTE 5 The KIM individual name is the remainder of the IRI when removing the corresponding namespace prefix, such as dic-tag: from <http://schema.knx.org/2020/ontology/dictionaries/dic-tag#actuator> → actuator.

EXAMPLE Filter expressions in different variants.

- `filter[actsOnQualityKind.hasEquipmentType]=actuator`
// Source: Equipment, Datapoint (Device Catalog/Instance Data), Location, FP
// parameter name addresses a source with indirectly assigned tags as part of a QK
// query parameter with one value; filter all rooms with a QK with equipment type actuator
- `filter[hasLocationUsage]=airport`
// Source: Location (Instance Data)
// parameter name addresses a Location with directly assigned tags (with a parent OP)
- `filter[hasLocationUsage][or]=pool,wellness`
// Source: Location (Instance Data)
// parameter name addresses a Location with directly assigned tags (with a child property)
// query parameter with two values; filter all rooms with location usage pool or wellness
- `filter[hasTag]=switch`
// Source: Datapoint (Device Catalog/Instance Data), Location, FP, Application Function
// parameter name addresses a source with directly assigned tags (with a parent OP)
- `filter[hasPointInterface][and]=if.p,if.i`
// Source: Datapoint (Device Catalog/Instance Data)
// parameter name addresses a Datapoint with directly assigned tags (with a child OP)

EXAMPLE Filter expressions with different query parameter structures, incorporating QKs. To understand the filter results, the following example setup is assumed.

Datapoint A	Datapoint B	Datapoint C
QK1: Locality: <i>wire</i> QK2: Locality: <i>site</i>	QK1: Locality: <i>wire</i>	QK1: Locality: <i>wire, site</i>

- `/datapoints?filter[hasQualityKind.hasLocality][or]=wire,site`
 // parameter **name** addresses a Datapoint with indirectly assigned tags as part of a QK
 // succeeds for Datapoint A,B,C → at least **one** assigned QK has one tag included
- `/datapoints?filter[hasQualityKind.hasLocality][and]=wire,site`
 // parameter **name** addresses a Datapoint with indirectly assigned tags as part of a QK
 // succeeds for Datapoint C → at least **one** assigned QK has both tags included
- `/datapoints?filter[hasQualityKind.hasLocality]=wire&filter[hasQualityKind.hasLocality]=site`
 // **each** parameter **name** addresses a Datapoint with indirectly assigned tags as part of a QK
 // succeeds for Datapoint A,C → each given query parameter definition needs to be // resolved individually to generate the overall result

4.2.1.4.2.4 filter[attributeKey][operator]

The parameter filters for specific resource items as part of a collection that are related to specific resource attributes, such as all devices with the order number (attribute) *5WG100-200-AB*.

- The parameter **name** *filter[attributeKey]* is specified for the part *operator* as specified in 4.2.1.4.2.1, for the part *attributeKey* with the *attributes* members as defined in 4.2.3.3.1.1. Vendor specific attributes are also possible, see example below.

Member of *attributes* that are defined as a combination together with (sub)properties needs to be a concatenation of each property individual, separated by a dot. This method is the same as it is defined in the previous clause for the QK/PK. For another example, see below.

- The parameter **value** is specified in relation to the used *attributes* members as:

1. MaC project (string) terms,
2. timestamp (time) values,
3. Datapoint (string/number) values,
4. KIM individual (IRI) names.

For the interpretation of the value, see 4.2.1.4.

The KIM individuals for the corresponding resources, their corresponding namespace prefixes and the actual individual names can be found in EN 50090-6-2 respectively in the electronic file.

NOTE The KIM individual name is the remainder of the IRI when removing the corresponding namespace prefix, such as unit: from https://qudt.org/vocab/unit/DEG_C → DEG_C.

EXAMPLE Filter with some attributes members.

1. `filter[orderNumber]=5WG100-200-AB`
 // device order number (*attributes* member *orderNumber*) is queried with *5WG100-200-AB*
2. `filter[gira:deviceUpdateCycle]=immediately`
 // device (KNX manufacturer member *deviceUpdateCycle*) is queried with *immediately*
 // the namespace prefix separator "": shall also be % encoded
3. `filter[unit]=DEG_C`
 // datapoint unit (attributes member *unit*) is queried with *DEG_C*
 // the namespace prefix unit: is not needed (property is part of the KIM)

4. `filter[version.hardware]=2.0`
// device hardware version (attributes member *version*) is queried with (string) 2.0

EXAMPLE Filter with attributes member **timestamp** for updates of an individual Datapoint (.../timeseries).

1. `filter[timestamp][le]=2021-02-17T17:17:17Z`
// filter all updates, occurred before ...
2. `filter[timestamp][ge]=2021-02-17T17:17:13Z`
// filter all updates, occurred after ...
3. `filter[timestamp][ge]=2021-02-17T17:17:13Z&filter[timestamp][le]=2021-02-17T17:17:17Z`
// filter all updates, occurred between ...

EXAMPLE Filter with attributes member **timestamp** for updates of all Datapoints (.../datapoints).

1. `filter[timestamp][le]=2021-02-17T17:17:17Z`
// filter where the **last** update, occurred before ...
2. `filter[timestamp][ge]=2021-02-17T17:17:13Z`
// filter where the **last** update, occurred after ...
3. `filter[timestamp][ge]=2021-02-17T17:17:13Z&filter[timestamp][le]=2021-02-17T17:17:17Z`
// filter where the **last** update, occurred between ...

4.2.1.4.3 Competency questions

In this clause some (simplified) HTTP queries (questions) to an Installation (using filters) are given. All below examples make use of the option to omit KIM specific namespaces for the parameter name and value.

1. What locations (rooms) in the building are used as a kitchen or a bath?

```
GET      /locations?filter[hasLocationUsage][or]=kitchen,bath
RESPONSE: location collection
```

2. What are the set points and the current temperature of a specific location?

```
Step 1
GET      /locations/{locationId}/functions?filter[meta.@type]=temperatureAir
RESPONSE: application function collection

Step 2
GET      /functions/{functionId}/datapoints?filter[hasTagSet.
hasPointFunctionType][or]=sp,processValue
RESPONSE: datapoint collection
```

3. What is the current operation mode of the primary system?

```
Step 1
GET      /functions?filter[meta.@type]=generationCtrl&filter[operatesForTrade]=heating
RESPONSE: application function collection

Step 2
GET      /functions/{functionsId}/datapoints?filter[hasTagSet.
hasStateKind]=controlMode
RESPONSE: datapoint collection
```

4. What is the current HVAC operation mode of a specific location?

Step 1

```
GET      /locations/{locationId}/functions?filter[meta.@type]=airCtrl&
        filter[operatesForTrade][or]=heating,ventilation,cooling
RESPONSE: application function collection
```

Step 2

```
GET      /functions/{functionsId}/datapoints?filter[hasQualityKind.
        hasPointOperation]=enumeration
RESPONSE: datapoint collection
```

5. Which heating zone has air handling units?

Step 1

```
GET      /functions?filter[meta.@type]=emissionCtrl&filter[operatesForTrade]
        =heating
RESPONSE: application function collection
```

Step 2

```
GET      /functions/{functionsId}/datapoints&filter[hasQualityKind.
        hasEquipmentType]=ahu
RESPONSE: datapoint collection
```

6. It is cold in a room. Where can I change the setpoint?

Step 1

```
GET      /locations/{locationId}/functions?filter[meta.@type]=emissionCtrl
RESPONSE: application function collection
```

Step 2

```
GET      /functions/{functionsId}/datapoints?filter[hasQualityKind.hasPhenomenType]
        =air
        &filter[hasQualityKind.hasQuantityKind]=Temperature&filter[hasTagSet.
        hasPointFunctionType]=sp
RESPONSE: datapoint collection
```

7. Read all history datapoint values before a given time.

```
GET      /datapoints/{datapointsId}/timeseries?filter[timestamp][le]=2019-04-
        22T14:07:17Z
RESPONSE: time series item collection for this datapoint
```

8. Read all history datapoint values from a given time onwards.

```
GET      /datapoints/{datapointsId}/timeseries?filter[timestamp][ge]=2019-04-
        22T14:07:17Z
RESPONSE: time series item collection for this datapoint
```

9. Read all history datapoint values for a given period.

```
GET      /datapoints/{datapointsId}/timeseries?filter[timestamp][ge]=2019-04-
        22T14:07:17Z&
        filter[timestamp][le]=2021-04-22T14:07:17Z
RESPONSE: time series item collection for this datapoint
```

10. Read all datapoint values of a specific heat distribution system (heating circuit) that operates for the metering trade (to summarize the energy consumption).

Step 1

```
GET /functions/?filter[meta.@type]=distributionCtrl&filter[operatesForTrade]=heating
```

RESPONSE: application function collection

Step 2

```
GET /functions/{functionsId}/datapoints?filter[operatesForTrade]=metering
```

RESPONSE: datapoint collection

```
data:
- id: 8951704b-269b-34f9-9a8f-4e415c5723e1
  type: datapoint
  meta:
    "@type": [dic:dpa.1103.51, dic:meteringValue, oms:VM1]
  attributes:
    title: "[M-BUS INTEGRATED] Total volume"
    readable: true
    writable: false
    unit: unit:M3
    valueType: string
    value: "10.055"
    timestamp: 2021-09-08T11:01:44+02:00
    KNX:operatesForTrade: dic-tag:metering
    KNX:operatesOnProtocol: dic-tag:mbus
  relationships:
    datapointDevice:
      links:
        related: /datapoints/8951704b-269b-34f9-9a8f-4e415c5723e1/device
- id: 8951704b-269b-34f9-9a8f-4e415c5723e2
...
```

4.2.1.4.4 Error handling

The above-described query parameters can be used independently in a request. Moreover, more than one query parameter may be part of an individual request.

Query parameters may be not applicable to a specific server end point or not specified at all for an end point (details on which resource path what query parameters could be applied, refer to the electronic document in 4.1.6). If a server would simply ignore in such situations unexpected query parameters or their values (and proceed with possibly the next valid parameters), this would make the response data unpredictable for a client.

The HBES IoT 3rd Party Server:

- shall return a 400 response code (and optionally an error object) if the query parameter naming rule is violated (see JSON API, section Query Parameters).
- shall handle the query result as false if an HBES IoT 3rd Party Client provides a (syntactically correct) query parameter in its request where the parameter value is not specified or applicable for the server

EXAMPLE A client request with query parameter *filter[meta.@type]=Kitchen* to an endpoint returning a location collection; the value “kitchen” is not a defined location type as part of the KIM.

EXAMPLE A client request with query parameter *filter[meta.@type]=Datapoint* to an endpoint returning a location collection; the value “Datapoint” as a resource type is not applicable for a location resource.

This results in an empty response payload for the resource. The result behaviour is the same as for the standard case when querying with a specified/ applicable query parameter value to a collection that has currently no such “matching” resource items included.

- shall return a 422 response code (and optionally an error object) if a HBES IoT 3rd Party Client provides a (according to the RFC syntactically correct) query parameter in its request to a resource path addressing an individual with reserved query parameters `page[...]` or `filter[...]`.

EXAMPLE A client request with query parameter `filter[hasPointInterface]=if.i` to an endpoint returning a datapoint individual; the query parameter `filter[...]` is not allowed on an individual.

EXAMPLE A client request with query parameter `page[number]=2&page[size]=3` to an endpoint returning a location individual; the query parameter `page[...]` is not allowed on an individual.

- may return a 422 response code (and optionally an error object) if a HBES IoT 3rd Party Client provides a (according to the RFC syntactically correct) query parameter in its request with a for the server unknown query parameter definition.

EXAMPLE A client request with query parameter `filter[value][and]=10,12` to an endpoint returning a collection of datapoint values; this is a valid query parameter expression for a collection that does not make sense from an **applicative** point of view (the actual value of a datapoint can have only one value at a time).

EXAMPLE A client request with query parameter `filter[value][or]=10` or `filter[value][and]=12` or `filter[value][eq]=10,12` or `filter[value]=10,12` to an endpoint returning a collection; this is an **invalid** query parameter expression for a collection.

EXAMPLE A client request with query parameter `foo=bar`, this is an unknown query parameter expression.

4.2.1.5 Response object

A request to a resource path addressing an item or a collection on the HBES IoT 3rd Party API creates a response document.

Such a response document contains (message body) data, it can be further detailed with a structure containing meta information or relationships to other resources.

The HBES IoT 3rd Party API structure of the message body data are designed with the specific goal to provide human readable content (in case of security decoded content). For the definition of the message body data, see 4.2.3.

4.2.1.6 Response status codes

A HBES IoT 3rd Party Server **shall** respond with defined HTTP response status codes from [RFC 7231], respecting their intended semantics. Response status codes other than those described in the below clauses are vendor specific (including those to properly support the ETag mechanism).

The column “Details” indicates whether a specific behaviour of the HBES IoT 3rd Party API is required for a response status code.

4.2.1.6.1 Status codes 1xx

1xx response status codes indicate an interim response for communicating connection status, as given in Table 5.

Table 5 — Response codes 1xx

Status	Title	Support	Details
101	Switching Protocols	M	-

4.2.1.6.2 Status codes 2xx

2xx response status codes indicate that the client's request was successfully received and accepted, as given in Table 6.

Table 6 — Response codes 2xx

Status	Title	Support	Details
200	OK	M	Request accepted. NOTE According to JSON API, the response code is used on a successful request to an individual resource or resource collection. EXAMPLE The accessed URL resource path <code>/functions</code> End point does not hold any items as the Installation did not have the data (the MaC project does not have MaC Functions). The successful (200) response contains a JSON API conform payload indicating empty data by using the data member and optionally the collection member (see 4.2.3.3). EXAMPLE The accessed URL resource path <code>/functions/{id}</code> End point does not hold the resource item with <code>{id}</code>, therefore an unsuccessful (404) response shall be used.
201	Created	M	-
202	Accepted	M	Request accepted; further processing not completed. NOTE An HBES IoT 3rd Party Server signals that there is no guarantee that the completion of the request by the server will be successful. Common use case is to immediately release the client from its duty to further observe the request, although the request has not yet been processed by the server. EXAMPLE When writing datapoint values, the server may need to wait on a (positive/negative) confirmation from a subsystem. Without this any response code does not carry a mature status of the request. The use of this response code signals this situation.
204	No Content	M	Successful request, no further response data will be sent. NOTE An HBES IoT 3rd Party Server signals that requested changes happened as intended. Common use case is simply to inform the client that the request was done, an ETag flag may also be used in addition to this. EXAMPLE Deleting a server subscription normally does not require a confirmation from a subsystem, the response code signals the immediate success. EXAMPLE When writing datapoint values the server may get a confirmation immediately, such as on very low latency subsystems or if the datapoint is hosted at the server only. The response code signals the immediate success.

4.2.1.6.3 Status codes 4xx

4xx response indicates that the client seems to have sent erroneous requests, as given in Table 7.

Table 7 — Response codes 4xx

Status	Title	Support	Details
400	Bad Request	M	-
401	Unauthorized	M	-
403	Forbidden	M	-
404	Not Found	M	Server cannot find the requested resource (path). NOTE According to JSON API, the response code is used for a request to a not existing resource. EXAMPLE The accessed URL resource path <code>/functions/{id}</code> End point does not hold the resource item with <code>{id}</code>. EXAMPLE The accessed URL resource path <code>api/v200/functions</code> End point does exist. Normally a client requests beforehand all api versions supported by a server with their corresponding base path definitions (<code>api/v200</code>), by reading it from the corresponding End point <code>"/.well-known/knx"</code>.
405	Method Not Allowed	M	-
406	Not Acceptable	M	-
408	Request Timeout	O	-
409	Conflict	M	-
412	Precondition Failed	O	-
413	Request Entity Too Large	O	-
414	URI Too Long	M	-
415	Unsupported Content-Format	M	-
422	Unprocessable Entity	M	-
429	Too Many Requests	M	-

4.2.1.6.4 Status codes 5xx

5xx response status codes indicate that the server is incapable of performing the requested method, as given in Table 8.

Table 8 — Response codes 5xx

Status	Title	Support	Details
500	Internal Server Error	M	-
501	Not Implemented	M	-
503	Service Unavailable	M	-
504	Gateway Timeout	M	-
505	HTTP Version Not Supported	M	-

4.2.1.6.5 Common error handling

In case a server encounters multiple problems, it **may** continue processing. For instance, a server may process multiple attributes and then return multiple validation problems in a single response.

When a HBES IoT 3rd Party Server encounters multiple problems of the **same** error domain (such as 4xx) for a single request, the most applicable HTTP error code **shall** be used in the response.

When a HBES IoT 3rd Party Server encounters multiple problem of a **different** error domain (such as 4xx and 5xx) for a single request, the error code 500 with additional error details in an error object (see 4.2.1.6.6) **shall** be used in the response.

NOTE For instance, response status codes 400 *Bad Request* may be appropriate for multiple 4xx errors or 500 *Internal Server Error* for multiple 5xx errors.

Above listed points are derived from JSON API, see section Errors.

4.2.1.6.6 Error objects

On any problem encountered during server operations while processing data, the server should return error objects (together with a suitable HTTP response status code). By doing so, it is possible to supply more details than just the status code itself, whether caused by the client or the server (i.e. both for [4xx] or [5xx] error codes).

The following shall be supported by a HBES IoT 3rd Party Server.

1. A HBES IoT 3rd Party Server **shall** report errors with the top-level `errors` member including a **non-empty** array of individual error objects as described in JSON API.
2. A HBES IoT 3rd Party Server **may** include error details in its response as defined in [RFC7807] or as defined in JSON API, section Error Objects.

The error object shall contain at least a `title` (as error object identifier) and `status` member and should contain a `detail` member with additional error details specific to the request. In addition, an error object may contain additional custom error members.

EXAMPLE A JSON API `errors` member containing two (artificial) error objects with mandatory/optional members and a customer error member `authentication_uri`.

```

errors:
- title: Forbidden // mandatory
  status: 403 // mandatory
  detail: Service requires other access rights // optional, JSON API
  links: https://schema.knx.org/2020/api // optional, JSON API
  authentication_uri: /oauth // optional, custom
  title: Not Implemented
  status: 501
  detail: Not implemented for this version of the API

```

Stack traces **shall** not be part of an error object. Stack traces can leak sensitive information that partners, and third parties are not allowed to receive and may disclose insights about vulnerabilities to attackers.

4.2.1.7 Content negotiation

The content negotiation allows a future HBES IoT 3rd Party API to extend the API content or to support different content type formats on the same API resource path and/or name structure.

The HBES IoT 3rd Party Server **shall** support a defined API content type and character encoding on the HBES IoT 3rd Party API. For this the HTTP element `Content-Type` **and the internet Media-Type** and `charset` included in it are **used**.

NOTE The HTTP Content-Type defines a header field [RFC 7231] that indicates the used HTTP payload (Internet) Media-Type [RFC 2046], formerly known as MIME Type.

4.2.1.7.1 Content type

1. A HBES IoT 3rd Party Server **shall** support the following `Media-Type` variants. Consequently the message body data **shall** be compliant with the `Media-Type` indicated in the body.

<code>application/vnd.api+json</code>	(normal req/resp End point and error handling)
<code>application/json</code>	(discovery responses and authorization responses)
<code>application/x-www-form-urlencoded</code>	(authorization requests)
<code>application/pkcs7-mime</code>	(certificate responses)

NOTE All media-types are already registered at IANA.

2. A HBES IoT 3rd Party Server **shall** support the following default `charset`.

<code>utf-8</code>	(for general content encoding)
--------------------	--------------------------------

The above-described `Media-Type` respectively the corresponding JSON format of JSON API requires to use the above defined `charset`, for this see JSON, 8.1.

4.2.1.7.2 Content type handling

For the common handling of `Media-Type` and `Content-Type` in relation to client requests and server response payload generation, refer to JSON API, 4.2.

NOTE JSON API details the handling of the `MediaType` as such and parameters disallowed here including the corresponding error handling with error code 406/ 415.

1. A HBES IoT 3rd Party Server **shall** reply with error code 400 if a client uses the following invalid request framing:
 - no payload content included, but expected from the End point (such as on a POST service)
 - payload content included, but not expected from the End point (such as on a GET service)

2. A HBES IoT 3rd Party Server **shall** send all data in its response documents only according to its payload `Media-Type`, which **shall** be specified also in its `Content-Type` response header.
3. In addition to JSON API a HBES IoT 3rd Party Server **shall** respond with an error code 406 if a client defines the following **Accept** header elements as part of its request:
 - no `Media-Type` matches the one that is specified (this also covers a list as part of the request)
4. In addition to JSON API a HBES IoT 3rd Party Server **shall** reply with error code 415 if a client defines the following **Content-Type** header elements as part of its request:
 - a `Media-Type` other than specified for the End point
 - an empty or not present `Media-Type`

An HBES IoT 3rd Party Server **shall** respond with `Media-Type` `application/vnd.api+json` if a client request specifies the `Content-Type` with the `Media-Type` `application/json` instead of a presupposed/specified `Media-Type` `application/vnd.api+json`.

NOTE The requirements under this point detail the error/ fallback handling in case of **entity** headers. The presence of a `Content-Type` in a request is mostly the case on access methods with additional data in the request, such as on POST/PUT/PATCH methods. The above definition from the second bullet is used as a fallback in case a client requests a similar format that the server can understand. To signal an error is not intended in such a case.

1. A HBES IoT 3rd Party Server **shall** respond with the default `charset`:
 - for all data included in its response documents
 - a part of `Content-Type` if it was requested by a client as part of the `Accept-Charset`; or if the `Accept-Charset` requested by the client cannot be served (this is the better behaviour rather than stopping the communication with an explicit error).
2. A HBES IoT 3rd Party Server may reply with no `charset` included if no `Accept-Charset` was defined in a client request.

4.2.2 Client requirement

Clients in general, respectively HBES IoT 3rd Party Clients as such are not in scope of this document and full requirements cannot be drawn up for them. Client requirements that **shall** be respected can be found in the JSON API section Content Negotiation.

If applicable all implicit client requirements not part of JSON API and of relevance for this document are addressed in this document as an explicit requirement on the HBES IoT 3rd Party Server respectively on the HBES IoT 3rd Party API.

4.2.3 Message data structure

This section describes the structure of the HBES IoT 3rd Party API HTTP request/ response message body data, identified by the `Media-Type`.

The first type listed in 4.2.1.7.1 is in line with JSON API, for this see JSON API standard, clause *Content Negotiation*. End points expressing the request/ response message body using this type **shall** be compliant with the JSON API document structure, for this see JSON API standard, clause *Document Structure*.

At the top level of the message body a JSON root object is present, other JSON API members follow. This document details the usage of the JSON API members `meta`, `data`, `attributes`, `relationships`, `links` and `typedescription`.

NOTE JSON prohibits the use of BOM and requires the use of UTF-8 for character encoding, see JSON, Clause 8.

4.2.3.1 Member names

NOTE According to JSON, a member is a **name**/value pair, more precisely a <string name-separator value > pair, for this see JSON, Clause 4.

EXAMPLE Member "HBES":30, with **name** "HBES" and value 30, first with quotes to denote left part as string, right part as a number.

A member **name** used in a JSON document **shall** be treated as case sensitive by a HBES IoT 3rd Party Server. This is in line with the JSON API conditions as described in section Member Names.

As also described for resource IDs in 4.1.10.5 member **names** **shall** not use any reserved characters specified in [RFC 3986]. This simplifies an easy mapping of member **names** from the HBES IoT 3rd Party API, possibly used as part of a resource path in URLs (see 4.2.4).

The requirement above further restricts the JSON API conditions as described in 5.8.

4.2.3.2 Meta object

The JSON API `meta` member describes additional non-standard meta/processing information.

It can appear as a top-level JSON API member, but may also appear as member of other JSON API members such as the `data` member or `relationships` member. The `meta` object **shall** not be part of the `attributes` member.

4.2.3.2.1 Top level usage

In this document the top-level usage of the `meta` member is used for the following purposes:

- Collection → information for a collection in the response
- No Data → information why no data are present in the response (no collection items/ single item)

Only a successful request with a response status code 2xx (see 0) the server shall respond with a top-level `meta` member in its response.

Collection

The number of returned items from a request to a resource collection may be limited with the help of request query parameters (see 4.2.1.4).

For a resource path that responds a collection of items, the server **shall** add a `meta` member containing a `collection` member that **shall** contain the following members.

- `number`: the number of pages skipped in the response
- `size`: the actual number of items returned in the response
- `total`: the total numbers of existing items in the whole collection

The request/ response behaviour including the use of the default values **shall** be taken into account as specified in 4.2.1.4.1.

No Data

For various reasons a response may not carry any primary data as part of the `data` member, requested items are not available in the response payload. The usual case for this behaviour is when calling the relationship of a resource object. Some detailed reasons can be found in 4.2.3.3.2.

EXAMPLE Asking (in a non-concurrent scenario where only one client is accessing the same server) a single device with {deviceId} x, this will return the device item with {deviceId} x. Asking a device with {deviceId} x its assembly location may fail to return a single location item with {locationId} y.

The reason can be detailed with an additional information.

- For a resource path that responds a single item, the server may add a `meta` member that contains a `nodata` member

EXAMPLE A GET request to retrieve the single (parent location) from the corresponding relationship `parentLocation` would return the following top-level meta object:

```
meta:
  nodata: no parent location item available
  data: null // a non-existing single parent location item, see JSON API
```

- For a resource path that responds a collection of items, the server may add directly as part of the mandatory existing meta (and collection) member a `nodata` member.

EXAMPLE A GET request to retrieve a collection of (child location) items from the corresponding relationship `childLocations` would return the following top-level meta object:

```
meta:
  nodata: no further child location items available
  collection: // mandatory for a collection
    offset: 0 // offset, not given in request, use default value
    items: 0 // limit, not given in request, use default value
    total: 0 // zero collection items present
  data: [] // an empty collection, no child location items, see JSON API
```

4.2.3.2.2 General usage

In this document the `meta` member is used to add information on a specific request or response payload data that is indicating a single item. Some examples:

- added callback information on a subscription request
- subscription characteristics/type information on a responded subscription item
- additional HBES manufacturer or KIM properties (how to express them as part of the `meta` member, refer to 4.2.3.3.5)
- additional information of the type of a responded item
- additional information of the node or installation as such

To describe the above-mentioned data in more detail, the `meta` object shall be used as part of the `data` member, as shown in the below examples and the examples given in 4.2.5.3 and 4.2.5.4.

Type information

The following resource items may host additional type information, for this the `@type` member shall be used, it is defined as an array. The content is described in the KIM as a KIM Class that is expressed as an IRI. The indicated type can be the KIM Class name or a URN that may be also used standalone or added to the indicated type.

NOTE The KIM class name is the remainder of the IRI when removing the corresponding namespace prefix, such as loc: from <http://schema.knx.org/2020/ontology/loc#Floor> → Floor.

- The KIM classes for datapoint, function and device resources, their corresponding namespace prefixes and the actual class names can be found in EN 50090-6-2 respectively in the electronic file.

The KIM classes for location resources, their corresponding namespace prefixes and the actual class names can be found in Table 12.

Location

Add for an individual location additional (location type) information in addition to the generic type “location”. The content **shall** be indicated with the most specific single type, e.g. ‘Room’ rather than ‘Space’.

NOTE Returning a common class type limits the possibility of a client to filter for **specific** resource types, clients are using these type values as an input for filtering.

```
data:
...
meta:
  "@type": [loc:Room]
  // KIM class reference, part of the semantic export
data:
...
meta:
  "@type": [Floor]
  // KIM class reference, part of the semantic export

data:
...
meta:
  "@type": [urn:knx.loc.room]
  // URN as type info of a location standardized by KNX Association (with ref. to IFC)

data:
...
meta:
  "@type": [loc:Room,urn:knx.loc.room]
  // KIM class reference and URN as type info
```

Datapoint

Add for an individual IoT Datapoint additional (type) information to the generic type “datapoint”. It is recommended that the content is indicated with the most specific single type or several types.

```
data:
...
meta:
  "@type": [dic:dpa.418.63]
  // KIM class reference of datapoint, part of the semantic export
data:
...
meta:
  "@type": [urn:knx:dpa.418.63]
  // URN reference of datapoint
```

Function type

Add for an individual application function additional (function type) information to the generic type “function”. It is recommended that the content is indicated with the most specific single type or several types.

```
data:
...
meta:
  "@type": [urn:knx:fct.switchingLight]
  // URN as type info of a KNX standardized Application Function
```

Device Type

Add for an individual device additional (device type) information to the generic type “device”. It is recommended that the content is indicated with the most specific single type or several types.

```
data:
...
```