



BSI Standards Publication

Home and Building Electronic Systems (HBES)

Part 6-2: IoT Semantic Ontology model description

National foreword

This British Standard is the UK implementation of EN 50090-6-2:2025. It supersedes BS EN 50090-6-2:2021, which will be withdrawn on 30 June 2028.

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 34478 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 6-2: IoT Semantic Ontology model description

Systèmes électroniques pour les foyers domestiques et les
bâtiments (HBES) - Partie 6-2: Description du modèle
ontologie sémantique IoT

Elektrische Systemtechnik für Heim und Gebäude (ESHG) -
Teil 6-2: Beschreibung des IoT semantischen
Ontologiemodells

This European Standard was approved by CENELEC on 2025-05-12. 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 Information Model	11
4.1 Introduction	11
4.1.1 General.....	11
4.1.2 Models.....	11
4.1.3 Taxonomy	13
4.1.4 Versioning.....	16
4.1.5 Availability	16
4.1.6 IRIs and Namespaces	17
4.2 Semantic Dictionary.....	18
4.2.1 General.....	18
4.2.2 Content.....	19
4.2.3 Semantic Dictionary Versioning.....	21
4.3 Core Model.....	21
4.3.1 Introduction	21
4.3.2 Classes.....	23
4.3.3 Relations	37
4.3.4 Examples	43
4.4 Location Model.....	53
4.4.1 Introduction	53
4.4.2 Classes.....	55
4.4.3 Relations	59
4.4.4 Examples	61
4.5 Tag Model	62
4.5.1 Introduction	62
4.5.2 Cardinalities.....	63
4.5.3 Classes.....	66
4.5.4 Relations	77
4.6 HBES Model.....	81
4.6.1 Introduction	81
4.6.2 IRI Scheme	81
4.6.3 URN Scheme.....	82
4.6.4 Classes.....	83
4.6.5 Relations	101
4.6.6 Examples	107
5 Appendix.....	113
5.1 Semantic Export.....	113
5.2 Examples	117
5.2.1 General.....	117
5.2.2 Room Temperature Control	118
5.2.3 Light Switch Control	123
Bibliography	128

European foreword

This document (EN 50090-6-2:2025) 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) 2026-06-30 implemented at national level by publication of an identical national standard or by endorsement
- latest date by which the national standards (dow) 2028-06-30 conflicting with this document have to be withdrawn

This document supersedes EN 50090-6-2:2021 and all of its amendments and corrigenda (if any).

EN 50090-6-2:2025 includes the following significant technical changes with respect to EN 50090-6-2:2021:

- certain HBES concepts were split to have a better distinction between definitions and their types (e.g. datapoints and their types);
- improved linking between different concepts such as devices and application functions;
- copies of external ontology parts were removed from the HBES ontology and replaced by references.

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 the HBES Information Model and a corresponding data exchange format 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 50090-6-3:2023, *Home and Building Electronic Systems (HBES) - Part 6-3: 3rd Party HBES IoT API*

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

use of 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**

specific perspective on a system that contains things with different properties, or 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**

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

Note 1 to entry: 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****FOI**

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

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

3.1.11

Functional Block

FB

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

FP

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

GA

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

GO

object which is foreseen for Group Communication using Group Address(es), which may be accessed via point-to-point communication without an assigned Group Address and becoming a member of that Function Point represented by the Group Address when assigned 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 KNX Association, hence the abbreviation KIM.

3.1.17

Industry Foundation Classes

IFC

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**

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

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

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

3.1.20**IoT Function**

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

MaC

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

OP

<in OWL> built-in concept that **connects** pairs of individuals

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

3.1.31

point

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 techniques 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

EXAMPLE Functional Block or Channel Datapoints.

3.1.33**Quality Kind****QK**

certain combination of observable or actuatable properties, available as predefined parts of the Semantic Dictionary or created individually during Configuration, it being 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**

standard model for data interchange on the Web

Note 1 to entry: RDF is specified by the <https://www.w3.org/RDF/>

Note 2 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/>

[SOURCE: <https://www.w3.org/RDF/>, Notes 1 and 2 to entry added.]

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 4.2.9.

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

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 specification a Tag is almost exclusively a term from the Semantic Dictionary.

3.1.40

thing description

TD

semantic metadata model to describe (abstract or physical) things

Note 1 to entry: It is specified by the thing description <https://www.w3.org/TR/wot-thing-description/> and thing Ontology <https://www.w3.org/2019/wot/td>

Note 2 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.

DHWC	Domestic Hot Water Controller
FOI	Function of Interest
BOC	Boiler Controller
BUC	Burner Controller
FTC	Flow Temperature Controller
GA	Group Address
GO	Group Object
FB	Functional Block
FP	Function Point
HBES	Home and Building Electronic Systems
HDTRT	Heat Demand Transformer Room Temperature
HFDM	Heat Flow Demand Manager
HIRC	Heating Individual Room Controller
HPM	Heat Producer Manager

HVA	Heating Valve Actuator
HZC	Heating Zone Controller
IFC	Industry Foundation Classes
IOO	Info On off
IO	Input Output
IoT	Internet of Things
KIM	HBES Information Model managed by KNX Association
KNXA	KNX Association
MaC	Management Client
OP	Object Property
QK	Quality Kind
RSM	Room Setpoint Manager
RTC	Room Temperature Controller
RTS	Room Temperature Sensor
SOO	Switch on/off
TD	Thing Description

4 HBES Information Model

4.1 Introduction

4.1.1 General

Facility management often uses heterogeneous visualization tools and applications to maintain a building control system. For this, an Installation is created based on a MaC project that includes instantiated devices and application functions assigned to specific locations. HBES IoT defines the HBES Information Model (abbreviated as KIM) to describe a MaC project with semantic information.

A MaC can export project data with additional semantic information (so-called Semantic Export); other clients or tools can import this project data. This is an easy and powerful way to integrate existing project data into a third-party tool or any service of choice.

4.1.2 Models

The KIM consists of several models, extended with HBES related definitions, as shown in Figure 1. Together they describe the different aspects of the building control domain.

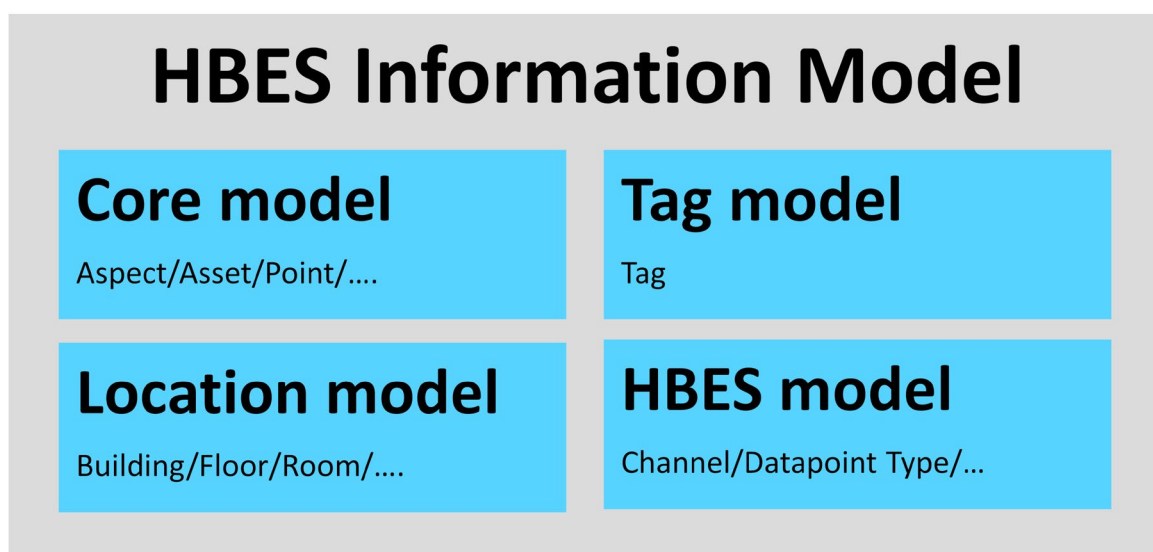


Figure 1 — Main HBES Information Model sources

Each model uses an individual namespace prefix, defines a specific set of semantically standardized entities and relationships amongst them.

1) Core model

Describes the common concepts of the building control domain for an Installation.

EXAMPLE A datapoint (I/O interface) of a device.

2) Location model

Describes the spatial location structure of an Installation.

EXAMPLE A location structure representing a building with rooms, floors, and others.

3) Tag model

Describes a set of entities used as indications for information classification. Each entity has a standardized semantical meaning that is independent of an Installation.

EXAMPLE A device with datapoints, each assigned with specific entities, such as output or temperature.

4) HBES model

Describes HBES System specific entities of an Installation.

EXAMPLE A device with datapoints (core model concepts), whereby a specific datapoint type is assigned to each datapoint (an HBES model concept), such as a boolean type for the values on or off.

The HBES Model is derived from the Core, Tag and Location models in this standard. The entire HBES Information Model (KIM) can be seen as an aggregation of the models.

HBES Information Model = Core/ Location/Tag/HBES model + Semantic Dictionary (see Clause 4.2)

NOTE The separation between a model and its entities is needed to extend/ update them independently.

The resulting data after commissioning an Installation (MaC project data) is colloquially also called **instance data**, opposite to the so-called **catalogue data**, which are predefined product descriptions or content from the Semantic Dictionary. For devices, catalogue data represents all possible configurations, whereas instance data are one specific configuration out of all possible ones.

NOTE For configuration workflow details and commonly used terminology, please refer to Clause 5.1, section workflow.

4.1.3 Taxonomy

A taxonomy structures a domain of interest in the corresponding hierarchical sections and subsections (but not more). An Ontology identifies and distinguishes concepts and their relationships to a domain of interest by (re)using a taxonomy as follows:

- The taxonomy sections and subsections are represented by Ontology classes, also called concepts.
- To represent relationships between Ontology classes, Ontology properties are added (so-called class constraints).

Class/Concept/Individual

A taxonomy section (discrete domain of interest) corresponds to an Ontology classification or, rather, an Ontology concept. The Ontology concept is represented by the corresponding Ontology class, which can contain other Ontology subclasses, and this is depicted with *subClassOf* in this specification.

Individuals are a concrete representation of a specific Ontology class; they are class members.

EXAMPLE The Ontology class `loc:Room` is part of a location taxonomy; it reflects the “available rooms in an installation”. It is a subclass of `loc:Space`, which is a broader definition of a location (compared to a room). Other classes of this location taxonomy are `loc:Building` or `loc:Floor`. The actual room *myBathRoom* as part of an Installation is an individual; it is a member of `loc:Room`.

The main KIM Ontology class structure is shown underneath, and the parent and child subclass relations are represented by the textual indentation of classes.

```
owl:Thing                                // the ontology root node for any taxonomy
  • core:Thing                            // to integrate subsequent core ontology classes (core model)
    • core:Aspect                         // a specific system view to grouped datapoints
      • core:ApplicationFunction          // a set of functions to achieve a desired behavior
      • ...
    • core:Asset                          // a container of installation objects such as devices, equipment,
                                          software
    • core:Location                       // to integrate an individual location concept (in the core model)
      • loc:Location                     // a spatial location concept with rooms, floors, and others
    • core:Point                          // an interface to interact with Points at Runtime
      • core:Datapoint
      • ...
    • core:Product                        // a product concept
    • knx:DatapointType                   // to integrate standardized DPTs (HBES model)
    • knx:DatapointTypeField              // to integrate standardized DPTs (HBES model)
    • tag:Tag                             // to integrate tag-oriented terms of the Semantic Dictionary (tag
                                          model)
```

- tag:TagSet // to integrate tag combinations of the Semantic Dictionary (tag model)
- qudt:Unit // to integrate QUDT (SI) units (external concept)
- vcard:Address // to integrate a postal address (external concept)

NOTE The external concepts qudt/vcard are described where they are used and not as individual classes. Because of size limitations and stability, the external concepts are not (OWL) imported by the KIM, as this is usually done in OWL2. The needed individuals/classes are explicitly declared in the KIM, too.

Each Ontology class in the specific model stands for its discrete domain meaning. Hence the classes (their meanings) are typically disjoint from each other. Where needed, the KIM expresses a required disjoint statement.

EXAMPLE The Ontology class core:Product and core:Point are disjoint, each has its own meaning.

The Ontology class loc:Space and loc:Outside are not disjoint. A *myGarden* individual of member *space* can also be of member *outside*. If disjoint, this would not be possible; in OWL, membership in one of two disjoint classes implies no membership in the other (*myGarden* as a member of loc:Space cannot be a member of loc:Outside at the same time).

Properties

Object Properties

Object properties (OP) define a relationship between individuals (members of Ontology classes).

The main KIM Ontology object property structure is depicted underneath. The parent and child object property relations are represented by the textual indentation of properties.

- core:containsAsset // to relate between different installation assets (core model)
- core:groups // to relate an aspect to its datapoints included in it (core model)
- core:hasPart // to relate between different tangible pieces of equipment (core model)
- core:hasPoint // to relate between functionality and its points (core model)
- core:hasProduct
- core:hasProxy
- core:implements
- core:refersTo
- loc:hasLocation
- knx:hasFunctionPoint
- knx:hasTagSet
 - knx:hasOperationKind
 - knx:hasQualityKind
- qudt:unit
- tag:hasTag
 - knx:hasEquipmentType
 - knx:hasPhenomenonType
 - ...
- vcard:has Address

The object property relations are explained in the corresponding clauses in more detail. The definition also contains a second OP relationship predicate (in *italics*) that describes the corresponding inverse relation. In general, the object property definition is structured as follows:

- OWL object property domain and range are described for the main OP relationship; domain and range are usually interchanged for the inverse OP relationship. In case more than one class is applicable for the domain or range (logically a union of classes), this is represented with an “/”.
- OWL object property axioms, such as symmetric or functional, are not described (see Clause 4.1.4).

NOTE In OWL 2, an OP relationship refers to individuals.

Annotation Properties

Annotation Properties are used to define additional annotations for the Ontology; the audience for these properties are mainly human readers of the ontology (using a tool) or software that is analysing or preprocessing the Ontology as such.

EXAMPLE `rdfs:label`, `rdfs:comment` (Ontology class/property name/comment), `rdfs:isDefinedBy` (origin as URI/URL), `dct:identifier` (Ontology class URN), `dct:subject` (Ontology class keywords, key phrases, classification codes), `knx:interfaceObjectTypeAnnotation` (specific identifiers laid down by KNXA)

To also support translated content an annotation property may be present multiple times with an additional language identifier; at least they are available in English with the related language identifier.

Annotation Properties do not affect the semantical meaning of an Ontology.

Data Properties

Data properties are used to define additional attributes for individuals; the audience for these properties are mainly machines to classify data, compute results or transport machine-readable instance data in an exported file.

To support translated content, a data property may be present multiple times with an additional language identifier, mainly used for data presented to humans.

EXAMPLE `dct:description` or `dct:title` (both edited by a MaC user, may be present several times in different languages), `jsonSchema:maximum` (upper value range for datapoints)

Summary

Figure 2 below depicts the basic KIM Ontology classes. The red dotted arrows show OP relationships between entities (examples, see Figure 3). Each Ontology class may have additional annotation properties and data properties, which are not shown in the figure.

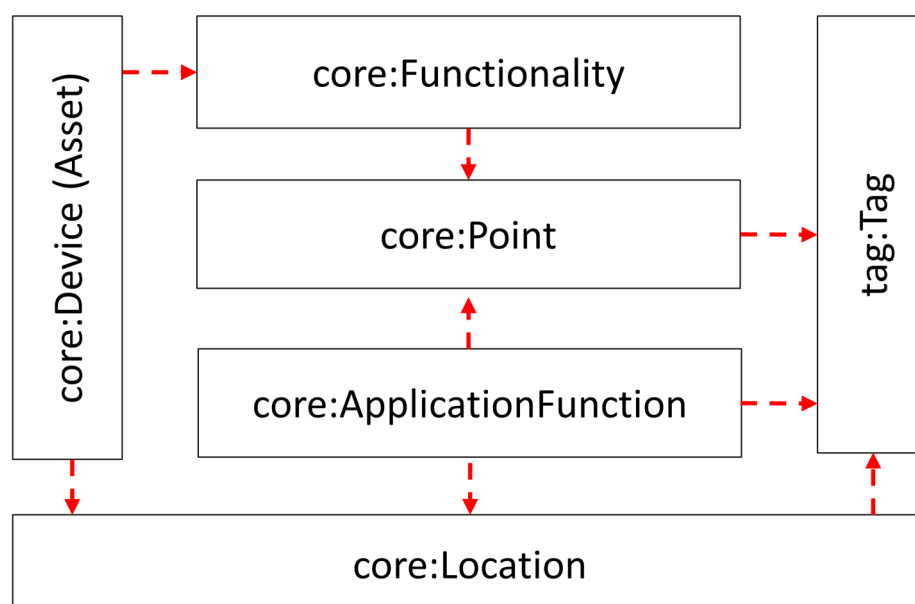


Figure 2 — Main KIM Ontology classes and their relations

4.1.4 Versioning

The KIM and the Semantic Dictionary are versioned.

- The versioning of the KIM is realized with a dedicated Ontology Version IRI, according to the W3C recommendations for OWL2 (clause 3.3).

The KIM is versioned as a whole; separate versioning for the used models is not foreseen. Any incompatible model or externally used concept change leads to a new KIM version.

NOTE Across different KIM versions, the Ontology Version IRI shall be different; the corresponding versioning identifier is described in Clause IRIs and Namespaces 4.1.6.

- The versioning of the Semantic Dictionary is described in Clause 4.2.3.

4.1.5 Availability

The KIM Ontology IRI underneath allows to retrieve the most recent Ontology as an electronic file by using the IRI as an URL. To get a specific KIM Ontology version, the corresponding KIM Ontology version IRI needs to be used instead as a URL.

- <http://schema.knx.org/2020/ontology> returns the most recent KIM Ontology version, including all used models and the content from the Semantic Dictionary
- <http://schema.knx.org/2020/ontology/v2> returns the version v2 of the KIM Ontology, using the same Ontology IRI as specified above, extended with the required version (here “v2”)

NOTE Regardless of the IRI scheme http://, for security reasons all downloads will be performed with the HTTPS scheme.

NOTE The electronic file is an integral part of this specification; the full extent of all Ontology classes, OPs, and others can be retrieved from this.

The KIM is available in a linked data format.

NOTE Current formats are TURTLE or RDF/XML, but this is not limited to this. The default format for the downloads from above is TURTLE, other formats can be obtained from <https://schema.knx.org>.

Data of the KIM is exchanged via Semantic Export; for details, see Clause 5.1.

4.1.6 IRIs and Namespaces

The KIM Ontology IRI is defined according to the following (simplified) principle.

{scheme} + “://” + {authority} + “/” + {path} + “/” + {version} + “#”

The KIM Ontology IRI is defined with “<https://schema.knx.org/2020/ontology>”.

NOTE With this the {scheme} and {authority} is defined as “http” and “schema.knx.org/2020/ontology”.

To express the IRIs for the different KIM models the {authority} and {path} shall be a string; for this, see Table 1.

The {version} shall be a string to express the different KIM versions, together with the Ontology Version IRI. It contains “v” concatenated with a non-zero leading decimal number, such as “v1” or “v249”, similar to the major part in the case of semantic versioning (major.minor.patch, SemVer).

NOTE The {version} is only part of the Ontology Version IRI.

Table 1 lays down the used namespace prefixes in the KIM, their corresponding IRIs, and Ontology references.

NOTE commonly used namespace prefixes in OWL such as rdfs: or rdf: are not listed.

Table 1 — KIM ontology names, namespaces prefixes and prefix IRIs

Ontology Name	Namespace Prefix Name	Prefix IRI ^a	Comment
JSON schema	<i>jsonSchema:</i>	https://www.w3.org/2019/wot/json-schema#	— data properties to express (min/max) limitations for datapoint values
QUDT	<i>qudt:</i>	https://qudt.org/schema/qudt/	— OPs and OWL classes to express units
	<i>unit:</i>	https://qudt.org/vocab/unit/	— Semantic Dictionary content from QUDT — individuals to express (SI) units (defined by QUDT)
	<i>quantitykind:</i>	https://qudt.org/vocab/quantitykind/	— Semantic Dictionary content from QUDT — individuals to express quantity tags (defined by QUDT)
VCARD	<i>vcard:</i>	https://www.w3.org/2006/vcard/ns#	— OPs and OWL classes to express geo addresses for a location

Ontology Name	Namespace Prefix Name	Prefix IRI ^a	Comment
DCTERMS	<i>dct:</i>	https://purl.org/dc/terms/	— data and annotation properties for description, title, and other
HTTP	<i>http:</i>	https://www.w3.org/2011/http#	— OPs for an URL authority part
KIM	<i>core:</i>	https://schema.knx.org/203/en52090-6-2/core#	— core model concepts — {authority} as specified, {path} = /core#
	<i>loc:</i>	https://schema.knx.org/203/en52090-6-2/loc#	— location model concepts — {authority} as specified, {path} = /loc#
	<i>tag:</i>	https://schema.knx.org/203/en52090-6-20tag#	— tag model concepts — {authority} as specified, {path} = /tag#
	<i>knx:</i>	https://schema.knx.org/2020/ontology/knx#	— HBES model concepts — uses a different {authority} as specified, {path} = /knx#
n/a	<i>mac:</i>	https://schema.knx.org/2020/ontology/mac#	— MaC related concepts — uses a different {authority} as specified, {path} = /mac# — how to integrate such content see Clause 4.6
	<i>gira:</i>	http://gira.de/2022/functions/xyz#	— manufacturer related content — uses an own {authority}, {path} = /xyz# — how to integrate such content see Clause 4.6
^a The IRIs do not necessarily need to be dereferenced as an URL to access any content explanation on a website. How to retrieve the KIM content, please refer to Clause 4.1.5).			

4.2 Semantic Dictionary

4.2.1 General

The semantical understanding and meaning of an Ontology needs to be guaranteed across different domains and especially across humans, who (pre/post-) process this information.

To support this, the KIM encompasses several KIM model Ontology classes, including assigned terms, to describe a semantically clearly specified information of the building control domain. This is called the Semantic Dictionary; its content is used to enrich the elements of an Installation with the information from above.

EXAMPLE A room in an Installation needs to be tagged for its usage as a kitchen. A semantically predefined dictionary term from the Tag model is used, the tag:kitchen with the label "Kitchen".

4.2.2 Content

The dictionary contains predefined content as well as content provided by manufacturers. Preferably the manufacturer content is directly integrated into the dictionary, but it may also be hosted separately.

NOTE Referring from an Ontology (e.g. manufacturer created) to an external (KIM) Ontology is a standard OWL (mapping) technology.

The entire dictionary content spans several Ontology classes described below.

Ontology Class	the corresponding KIM Model class where the dictionary content is hosted; the class as such is not part of the dictionary content
Entities	the general meaning of the dictionary content and its representation, including its naming convention (if applicable), Object/Data Properties are not part at all of the dictionary content
ns	the namespace prefix that is applied to the dictionary content, it shall be noted that it may differ from the namespace prefix of the class

Table 2 — Representation of dictionary content

Ontology Class	Entities	ns
core:ApplicationFunction	Used to describe standardized application functions categories with included applications functions, used in a MaC Project. Individuals not present Subclasses each subclass defines an individual application function category (see Clause 4.6.4)	knx:
core:FunctionalBlock	Used to describe standardized Functional Blocks with their Datapoints. Individuals not present Subclasses shall have the following naming convention: knx:fb.<functionalblock-id>	knx:
core:Datapoint	Used to describe by standardized Datapoints, used by Functional Blocks. Individuals not present Subclasses shall have the following naming convention: knx:dpa.<functionalblock-id>.<property-id> the IDs shall NOT use leading zeros;	knx:

Ontology Class	Entities	ns
tag:PhenomenonQualityKind tag:EquipmentQualityKind	Used to describe a meaningful combination of observable or actuatable properties as a tag set, used by Application Functions or Datapoints to express specific semantics. Subclasses not present Individuals shall have the following naming convention: ns:qk_<name> the <name> follows the principle to concatenate the property labels in the order of “type of interest” and shortcut to depict a phenomenon or equipment, quantity kind (if available), locality (if available, may also occur several times). Depending on the used phenomenon or equipment, the individual is assigned to the corresponding class from above. EXAMPLE tag:qk_ph_air_temperature_room, a member of class tag:PhenomenonQualityKind	tag:
tag:EquipmentType tag:Locality tag:LocationUsage tag:Origin tag:PhenomenonType tag:PointFunctionType tag:PointInterface tag:PointOperation tag:Protocol tag:QuantityKind tag:StateType tag:Trade	Used to describe semantically standardized tag categories, including their tags (individuals) from QUDT, Haystack, or KNX, also used from other dictionary content (datapoints, QK). Subclasses not present Individuals to be readable for humans, most tags follow the camelCase naming convention; for the Quantity Kind tags the PascalCase schema is used (tags from QUDT already use this scheme)	tag: quantityk ind:
knx:DatapointType	Used to describe standardized datapoint types, also used by Datapoints. Subclasses not present Individuals shall have the following naming convention: knx:<name> The <name> shall follow the naming convention as described in EN 50090–6-3:2023, Clause 4.2.5.4.2 Types. EXAMPLE knx:switch	knx:
knx:DatapointTypeField	Used to describe datapoint type fields. Subclasses not present Individuals follows no specific naming convention; existing DPT information will be reused EXAMPLE knx:type.dpt.field.11.1.i3.Reserved	knx:

Ontology Class	Entities	ns
qudt:Unit	Used to describe QUDT (SI) units, also used by other dictionary content (datapoint fields). Subclasses not present Individuals follows the naming convention from QUDT (the same IRIs are used)	unit:

4.2.3 Semantic Dictionary Versioning

The content of the Semantic Dictionary can be extended with new terms, or existing terms can be updated. For this, the Semantic Dictionary uses a dictionary version entity applied to the class core:Thing.

- For each new KIM version (see Clause 4.1.4), the dictionary version starts from zero again.
- The Semantic Dictionary is versioned as a whole, a change leads to an incremented dictionary version.

The Semantic Dictionary is maintained and updated by KNXA. An update assumes no compatibility break with the KIM models and the (previous) dictionary content. This means all classes with their individuals, as specified in Table 2, remain intact (content can only be extended). Otherwise, a new KIM version is needed.

4.3 Core Model

4.3.1 Introduction

To achieve the intended functionality of a building control system, an Installation defines several Application Functions. Usually, the functions observe or actuate upon a specific phenomenon/substance, whereas the physical values of that phenomenon are measured and adjusted by a piece of equipment (sensor/actuator).

From the viewpoint of a control system user/installer, the domain model needs to cover the following aspects of an Installation:

- Where is a Device installed, and which Points are hosted on a Device?
- Which Points of a Device belong to which Application Function?
- Where is the place of action of an Application Function?
- How does an Application Function or its Points interact with the physical world? What is the observed phenomenon/substance, and what is the measurement?
- How do Application Functions interact? For example, how does energy flow from energy producers to energy consumers?

Concepts

The involved entities (devices, points, ...) of the aspects mentioned above are aggregated as concepts in the core model; most of them are shown in Figure 3 below.

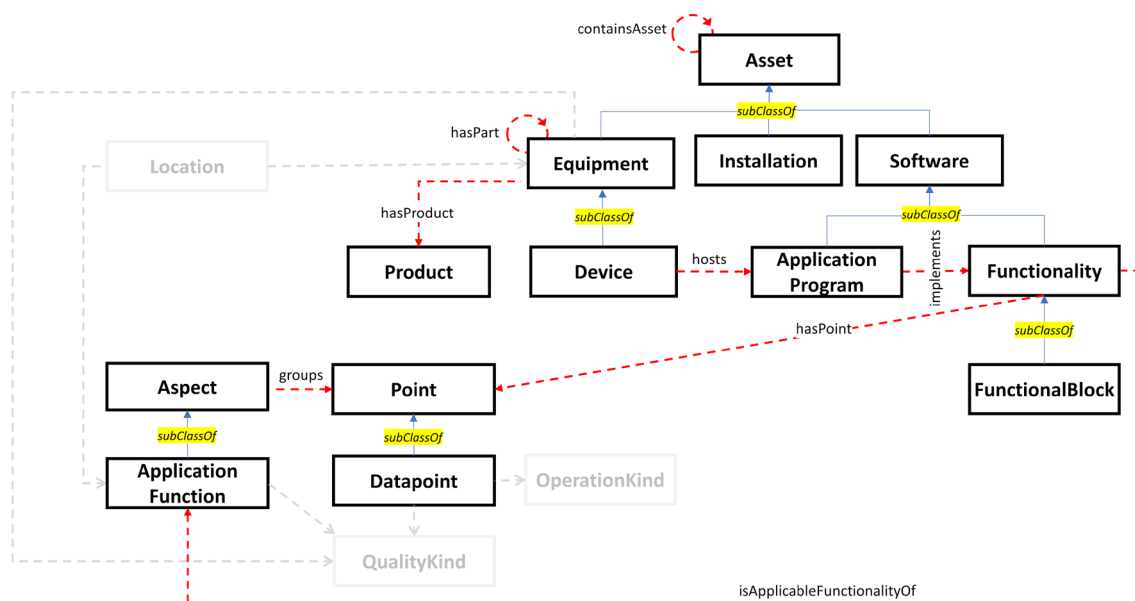


Figure 3 — Core Model

NOTE For readability, the namespace prefixes are omitted in the figure. Classes depict instance data (such as Device individuals) and catalogue data (such as Quality Kind individuals); this separation is not explicitly shown.

Figure 3 above depicts KIM core model classes. The red dotted lines and their direction describe OP relationships between core model classes. The OP is the name assigned to the dotted line (details for OP see Clause 4.3.3). Grey boxes show concepts from other models (grey dotted lines, too), depicted for better understanding.

EXAMPLE In an Installation, a *SmokeDetector* Device is assembled; it has a sensor (Datapoint) that measures the smoke in the kitchen.

SmokeDetector is an instance of *core:Device*, and one of its Datapoints is an instance of class *core:Datapoint*. A Quality Kind *OvenSmokeDetection* instance of class *tag:QualityKind* is assigned to the Datapoint (with the “type of interest” property *Smoke*).

Properties

The involved interactions (with the physical world) of the aspects mentioned above are described with different properties, each defining a specific semantic meaning in the building control domain. The properties are expressed as part of the Quality Kind concept.

- 1) A (what) property that describes the “type of interest” addressing equipment or phenomenon.

EXAMPLE *Air, Light or Person* for a phenomenon, a *Boiler* or *Actuator* for an equipment.

- 2) A (where) property that describes the possible location-oriented context for the measurement/adjustment (equipment) or observation/actuation (phenomenon).

EXAMPLE *Floor, Duct Section or Phase L1*.

- 3) A (how) property that describes the numerical quantity kind of the measurement/adjustment (equipment) or observation/actuation (phenomenon).

EXAMPLE *Length, Temperature, or Time*.

Some additional (whereby) properties define by what means the quantity kind is used, addressing an operation type, role, and origin. The properties are expressed as part of the Operation Kind concept.

EXAMPLE A user-desired Application Function for a temperature setting results in a Datapoint that actuates upon the Quality Kind “air temperature in a room”, also defining an Operation Kind as an *absolute* (operation type) *setpoint* (role) from a *user* (origin).

For details of a Quality Kind or Operation Kind, including some examples, please see in the clause below how the properties are used in the KIM and Clause 4.5.

4.3.2 Classes

4.3.2.1 General

The Core model Ontology class structure is depicted underneath; the parent and child subclass relations are represented by the textual indentation of classes.

- core:Thing // root node
 - core:Aspect
 - core:ApplicationFunction
 - core:Asset
 - core:Equipment
 - core:Cabinet
 - core:Device
 - core:Gateway
 - core:Installation
 - core:Software
 - core:ApplicationProgram
 - core:Functionality
 - core:FunctionalBlock
 - core:Location
 - core:Point
 - core:Datapoint
 - core>Actionpoint
 - core:Eventpoint
 - core:Product

4.3.2.2 core:Thing

Description

The Thing class separates HBES-specific Ontology concepts from other (external) Ontologies in the KIM (structurally and UI related).

Sub Class Of

owl:Thing

Object Properties

-

Data Properties

dct:description, dct:title

Disjoint With

vcad:Address, qudt:Unit

Notes

Intended Usage

dct:description, dct:title

The data properties allow to define (optional) title/description data, also in different languages. In the KIM, they are only assigned to individuals.

NOTE A title/description is assigned to Tag individuals from the Semantic Dictionary; both may be present several times (to support different languages). A MaC may only use one title/description for a specific installation individual, such as the name for a bathroom individual given by the installer (other tools may add more translated elements to this).

KIM Ontology classes don't use title/description definitions, which prevents ambiguous data on class inheritance. The titles/descriptions would be inherited (duplicated) from the subclasses to the parent class.

core:dictionaryVersion

The annotation property expresses a version identifier of the Semantic Dictionary as a nonnegative integer. It is used to identify dictionary updates (see Clause 4.2.3).

4.3.2.3 core:Aspect

Description

An Aspect is a generic grouping concept of Points that reflects a specific viewpoint or functionality of the Installation.

EXAMPLE A concrete Aspect is a Function Point that groups a set of Datapoints.

Sub Class Of

— core:Thing

Object Properties

— core:groups

Data Properties

— -

Disjoint With

— all first Ontology subclasses of core:Thing (see Clause 4.1.3)

Notes

-

4.3.2.4 core:ApplicationFunction

Description

An Application Function represents a partial functionality of a control system, characterized by input information from sensors, actuators, or upstream application functions being processed to give output information which, in turn, is intended for actuator or downstream application functions.

An Application Function also uses (intermediate) variables to reflect specific process variables or states in its control loop.

An Application Function directly or indirectly actuates upon QKs of primary "type of interest" with a physical (phenomenon or equipment) quantity; moreover, it operates in specific trades. To reflect a trade, a specific OP relationship predicate is defined.

An Application Function usually consists of the following elements:

— inputs from sensors

- outputs to actuators
- some control logic or computational Functional Block (both are software elements)

Some inputs or outputs may be (intermediate) variables received from or passed on to another Application Function. How these variables are used as part of the control loop, see Clause 4.3.4.2.4

Sub Class Of

- core:Aspect

Object Properties

- knx:hasFunctionPoint, tag:operatesForTrade, knx:hasQualityKind

Data Properties

- core:number, core:state, core:comment

Disjoint With

- knx:FunctionPoint

Notes

Application Functions can be structured in different categories, while a category is defined as a kind of order/hierarchy according to trades and cross cutting aspects that usually spans over more than one trade; some are specified in standards like VDI 3813 and 3814 or EN 15232.

EXAMPLE Category *Lighting Control* (contains light dimming relevant AFs, such as used on a floor), category *Air Temperature Control* (contains air temperature adjusting relevant AFs, such as used in a room) or category *Monitoring* (contains monitoring relevant AFs, to monitor cross cutting 'trade' aspect like fire/intrusion alarms or metering/weather data in a building).

Application function categories are defined with a corresponding Application Function class, if needed related subcategories are defined with corresponding subclasses. As described in Clause 4.2.2, they are part of the Semantic Dictionary.

For the Application Function categories and their relation to trades, see Clause 4.5.3.19.

EXAMPLE The AF category for '*air control*' (class knx:airCtrl) contains a sub category for '*air pressure control*' (class knx:airPressureCtrl). The AF category for '*lighting control*' (class knx:lightingCtrl) is assigned to the trade *Lighting*.

Intended Usage

tag:operatesForTrade

Allows to express for which trade(s) an Application Function operates.

knx:hasFunctionPoint

Allows to express the (Function) Points of an Application Function.

knx:hasQualityKind

Allows to express the semantics for an Application Function independently of its assigned Datapoints for the below-described reasons:

the assigned Datapoints of an Application Function do not carry any QKs

the assigned Datapoints carry QKs, but they do not describe the general meaning of the Application Function as such

EXAMPLE The meaning of an Application Function is to control the air temperature in a room; for this, it also includes an ordinary window contact. The corresponding Datapoint with the present QK does not support a tag combination (air, temperature, room) since it is implemented as a simplified binary window open/close control.

Usually, an Application Function expresses the assigned QKs with the OP knx:hasQualityKind. If applicable also one of the sub-properties can be used; in such a case, an explicit statement for a measurement/adjustment (equipment) or observation/actuation (phenomenon) is expressed for the Application Function.

NOTE In HBES, Function Points encapsulate the Datapoints involved in an Application Function.

For further details about the OPs, see Clause 4.6.5.

Background

During the configuration of an Installation, a MaC configures Device Datapoints and creates Application Functions with Points; the corresponding Device Datapoints are linked with Function Points.

Traditionally it is often impossible to derive automatically applicative knowledge for an Application Function or further semantical meaning during this process. Some problems are listed below.

- 1) The Points of an Application Function do not have a standardized description that can be interpreted; the name and functional description are insufficient to conclude in a complete and deterministic way on any Application Function. This also holds for the Device Datapoints assigned to the FPs.

EXAMPLE A *dimming* Application Function requires several Points, each for a specific setting (switch light on/off, dim light up/down).

The first FP may be linked with a common actuator I/O input Device Datapoint, technically sufficient to achieve the ordinary switch task.

The second FP may be linked with a common actuator I/O output Device Datapoint, technically sufficient to get the actuator status of the switch task.

To describe the Application Function (for the partial switch task), not enough semantical information can be gathered from the Datapoints (switch the light with an abs/rel. value), also not from the two FP (its status/control relation).

- 2) An Application Function is often assigned to one or more trades. Based on a (one dimensional textual) trade tag it is not possible to build a dedicated metrics in which an Application Function acts. Humans may infer an overall meaning, but not machines.

EXAMPLE The trade tag *Lighting* and *Ventilation* is assigned to an application function, the AF directly influences the emission control in a building, moreover it operates in energy-efficient lighting control. From the trades alone no individual equipment measurements/ adjustments can be made/concluded, such as CO₂ concentration and energy consumption (for this the individual datapoints are needed, with semantics assigned).

To resolve the shortcomings from above, an Application Function needs to be able to express the following KIM elements:

- the assignment to an Application Function category
- the assignment to a Trade (tag:operatesForTrade)
- the assignment of at least two involved (semantically annotated) Datapoints, linked together with the concept of a Function Point (knx:hasFunctionPoint) that encapsulates the Datapoints.
- the assignment to a QK (knx:hasQualityKind)

NOTE The essential assignment of possible QKs to an Application Function or FPs and the assignment between the status and control of FPs is specified in Clause 4.5.1.

The definition of an Application Function can be simplified if it is defined as a subclass of an existing Application Function. With this subclassing, all definitions from the parents are inherited, such as all Function Points or Tags expressed for the parent.

If an Installer creates a new Application Function with a MaC based on a predefined Application Function, then the semantics is available after its creation. Here the semantic information is applied or enriched consistently on all (created) individuals; predefined Application Functions are described in Clause 4.6.4.

In case an installer is individually creating an Application Function with a MaC, then no semantics will be available after its creation. All required semantics needs to be added manually by the installer.

4.3.2.5 core:Asset

Description

An [Asset](#) provides some form of economic value and is owned and managed by an entity (e.g. a service organization). It is something that is placed inside a building or attached to it but is not an integral part of that building's structure. An asset can be tangible or intangible.

- A tangible asset is a physical property – it can be touched. The term is most associated with fixed assets, such as buildings, HVAC systems, and equipment.
- An intangible asset is a non-physical asset that needs to be administered, such as licenses for a commissioning tool or office software.

Sub Class Of

- core:Thing

Object Properties

- core:containsAsset

Data Properties

- -

Disjoint With

- all first Ontology subclasses of core:Thing (see Clause 4.1.3)

4.3.2.6 core:Equipment

Description

[Equipment](#) is a tangible and replaceable asset; it can be anything in a building, for example, a whiteboard, a chair, a table, a fan, or a motor.

[Equipment](#) represents an asset as a part of an Installation. Owners may need to keep track of what equipment they have, who made and delivered it, when it needs maintenance, how to inspect it and track the history of service requests and work orders, etc.

EXAMPLE An equipment can have subparts; a device has “fittings” equipment with its corresponding “screws and rails”. All of them are assigned to the device with the OP core:hasPart.

Sub Class Of

- core:Asset

Object Properties

- core:hasPart, core:hasProduct, knx:hasQualityKind, loc:isEquipmentOf

Data Properties

- -

Disjoint With

- core:Software, core:Installation

Notes

Intended Usage

core:hasProduct

Allows to express a general relation to product-specific information; see class core:Product.

Knx:hasQualityKind

Allows to express a general relation to QKs; further details for the OPs see Clause 4.6.5.

NOTE The QKs are expressed independently at the equipment with their meaning. They do not stand for any inheritance of QKs, assigned to Datapoints from a Device (also equipment). The OP is also useful when equipment does not contain any Datapoints, such as sensor hardware or an ordinary power supply.

Loc:isEquipmentOf

Allows to express a relation to a location, in most cases, a building or room.

EXAMPLE A product category and/or product type assigned to a device in a room.

4.3.2.7 core:Cabinet

Description

A Cabinet is a physical enclosure for electrical or electronic equipment such as automatic switches, fuses, circuit breakers, control devices, timers, etc.

Sub Class Of

- core:Equipment

Object Properties

- -

Data Properties

- core:comment

Disjoint With

- core:Device

Notes

-

4.3.2.8 core:Device

Description

A [Device](#) can be any electronic equipment with some computing ability that has firmware, optionally with input and output terminals (I/O), an Application Program that provides a set of Points as the interface for the I/O operations, and parameter values controlling the behavior of the I/O operations at Runtime. Some above-described components are depicted in Figure 4 and Figure 5.

Sub Class Of

- core:Equipment

Object Properties

- knx:hasChannel, core:supportsFunctionalBlock, core:hasProxy, core:hosts

Data Properties

- core:lastDownloaded, core:lastModified, core:serialNumber, knx:installationHints, knx:individualAddress, core:hardware, core:firmware, core:state, core:comment

Disjoint With

- core:Cabinet

Notes

The class core:Device can be used as a bridging class to other Ontologies.

Intended Usage

knx:individualAddress

Represents the string value of an individual address of an HBES Device (internally a 16-bit number).

core:lastDownloaded

Represents the timestamp value of the last download of any Device data.

NOTE A device download covers the assigned Application Program but also MaC configuration data or the individual address from above; the latter are not an explicit part of the Application Program.

Background

A Device hosts a runtime interworking interface (Points) as part of an Application Program. The device below hosts an Application Program containing physical and logical inputs and outputs. These

inputs and outputs belong to a dedicated Functional Block with related I/O's and additional parameters and diagnostic Points.

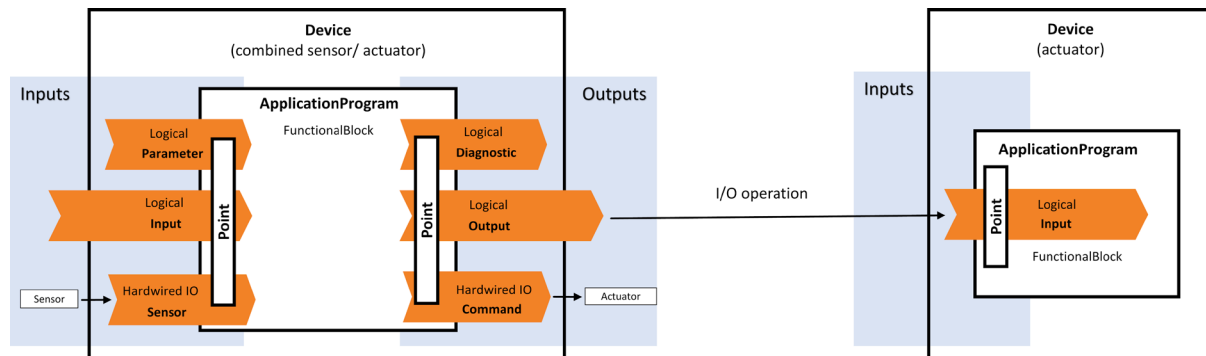


Figure 4 — Device Components

NOTE Black boxes denote core model concepts.

Logical bindings (I/O operation) between Devices are expressed by Function Points that define the runtime interworking of the Installation. A Function Point groups (OP core:groups) output Points from one Device with related input Points on other Devices.

NOTE The relation mentioned above is not set up automatically in an Installation; this is achieved by grouping (also known as “linking”) the intended Points together.

In HBES this is referred to as “link Group Objects with a Group Address”.

Points may belong to a specific Functional Block, with its mandatory and optional inputs, outputs, parameters, or diagnostic Points. Moreover, a Point may represent information about the Device (e.g. life check signalling, power on LED) or data related to the intended control functionality. A Functional Block can have the following interfaces:

- Logical Parameter Points are available to configure an Application Program.
- Logical Diagnostic Points are available for monitoring purposes.
- Logical Input Points receive data at Runtime.
- Logical Output Points send data at Runtime.
- Hardwired (physical analog/digital) I/O element.

The following figure provides a more detailed formalization of a Device as part of the core model.

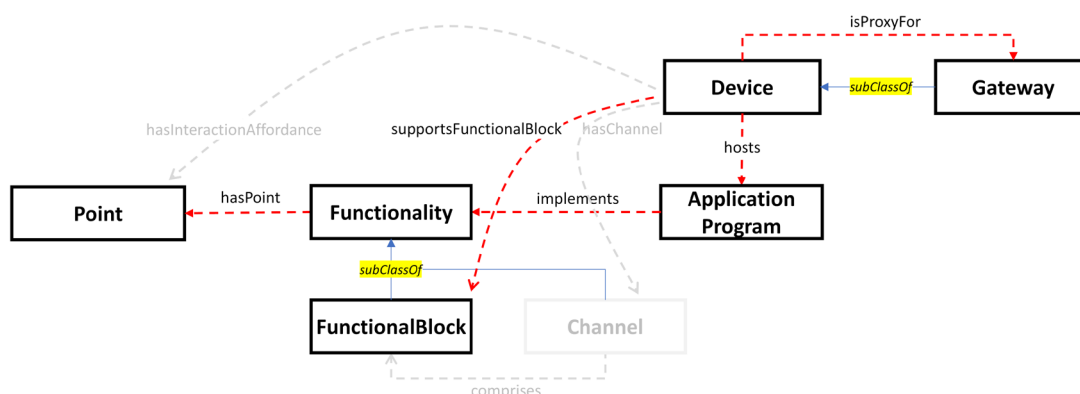


Figure 5 — Device view as part of the core model

NOTE The namespace prefixes are omitted in the figure for readability. The OP `td:hasPropertyAffordance` is not modelled in the KIM; it is used as part of the Semantic Export, see Clause 5.1.

Black boxes denote core model concepts. Red dotted lines and their direction describe OP relationships from the core model, the OP used here is the name assigned to the dotted line (details per OP see in Clause 4.3.3). Grey boxes show concepts from other models (grey dotted lines too), depicted for better understanding.

4.3.2.9 core:Gateway

Description

A Gateway is a network node (device) that connects two networks with different transmission protocols and data formats, typically between OSI Layers 4–5.

EXAMPLE The HBES IoT 3rd Party Server serves as an entry and exit point for an Installation with HBES TP devices (network 1) as all data passes through or communicates with this gateway before being accessible from an IP backbone (network 2).

Sub Class Of

- core:Device

Object Properties

- core:isProxyFor, http:authority

Data Properties

- -

Disjoint With

- -

Notes

-

4.3.2.10 core:Installation

Description

An Installation represents a deployed system consisting of Equipment and Application Functions used for a particular purpose.

EXAMPLE An HVAC system or fire protection system with cabinets, devices, and Application Functions, located in rooms and floors.

Sub Class Of

- core:Asset

Object Properties

- -

Data Properties

- knx:contractNumber, knx:macVersion, core:lastModified, knx:projectInstallationNumber, core:state

Disjoint With

- core:Equipment, core:Software

Notes

An Installation is a combination of a postal address and project data, configuration files, etc.

4.3.2.11 core:Software**Description**

Software represents a collection of data or instructions installed (hosted) on a Device or Computer and tells it how to work. Software is a generic term for applications, scripts, and programs that run on a Device.

Sub Class Of

- core:Asset

Object Properties

- -

Data Properties

- -

Disjoint With

- core:Equipment, core:Installation

Notes

Software refers to a piece of code running on a single device. In contrast, an Application Function refers to the resulting behavior of one or many devices that maybe exchange information.

4.3.2.12 core:ApplicationProgram

Description

An Application Program represents a piece of software that implements a functionality. Specific configuration parameters may be used to control how the functionality is operated at Runtime.

EXAMPLE An outside temperature sensor Device hosts an Application Program that implements a functionality containing some Points. At least one of the Points provides the measured outside temperature as a logical output Datapoint. A configuration parameter defines on which temperature threshold a change of (temperature) value is sent out, such as each 2K.

Sub Class Of

- core:Software

Object Properties

- core:implements

Data Properties

- core:manufacturer, core:version, core:model, knx:certificationStatus

Disjoint With

- all first Ontology subclasses of core:Thing (see Clause 4.1.3)

4.3.2.13 core:Functionality

Description

A Functionality represents a concrete piece of software and a set of interfaces (mainly Points) needed to accomplish a task for which a Device is designed.

Several subclasses are defined to differentiate the specific use of Points for a Channel (knx:Channel) or a standardized Functional Block (core:FunctionalBlock).

EXAMPLE An Application Program implements a functionality containing several Points to achieve two main tasks, room temperature control and a fourfold dimming actuator.

- 1) The room temperature control functionality is expressed as a set of Points, that comply with a specific Functional Block.
- 2) The dimming functionality is expressed as a set of Points of a Channel, which is available four times.
- 3) Some generic control functionality for the Application Program is expressed as a common set of Points.

Sub Class Of

- core:Software

Object Properties

- core:hasPoint, core:isApplicableFunctionalityOf

Data Properties

- -

Disjoint With

- all first Ontology subclasses of core:Thing (see Clause 4.1.3)

Intended Usage

core:hasPoint

A Functionality contains a group of Points that belong to it; for details, see the corresponding OP description in Clause 4.3.3.

core:isApplicableFunctionalityOf

Define the existence of an Application Function usable with a Functionality (or one of the subclasses), respectively, its Datapoints; for details, see the corresponding OP description in Clause 4.3.3.

4.3.2.14 core:FunctionalBlock

Description

A Functional Block represents a software element that has internal storage and produces a computable result with output values when executed with a particular set of input values.

- A Functional Block typically contains definitions for the (mandatory and optional) input/output Datapoints that express an application behavior.

NOTE The mandatory/optional presence of a Datapoint as part of a FB will be expressed with cardinality restrictions of exactly 1 or max 1, applied to the OPs `knx:isComposedByInput` or `knx:isComposedByOutput`.

- An individual Functional Block runs on the same Device; at Runtime, the FB is interworking with other Functional Blocks; typically, a FB output serves as one possible input for another FB input. For this, a Functional Block is defined with a template approach, allowing to express:
 - linked Datapoint inputs (usual case, an input is driven from other FB output Datapoints, see last bullet point below)
 - own Datapoint inputs (for cases where no further input information from a counterpart Datapoint output is available, such as when using the Datapoint input together with an external management station)
 - own Datapoint outputs (usual case, to allow its usage from other FB as input Datapoints; see first bullet point above)

NOTE Installation instances may claim to comply with a specific standardized FB, usually done with a corresponding type assignment to a specific FB (see Clause 5.2). The template approach allows the validation of Datapoints assigned to the instance against the FB template definitions (e.g. all datapoints from a device instance).

- Functional Blocks can be related to trades; others are of general use and named functions of common interest.

EXAMPLE Trade-oriented FBs are HVAC, fire, elevator, air handling unit (AHU), heat zone controller (HZC), general use FBs are PID control-loop, and named FBs are schedulers.

Sub Class Of

- `core:Functionality`

Object Properties

— -

Data Properties

- `knx:interfaceObjectType`

Disjoint With

- `knx:Channel`

Standardized Functional Blocks are defined as part of the Semantic Dictionary; for details, see Clause 4.6.1.

Intended Usage

`knx:interfaceObjectType`

A Functional Block is identified with a unique id (IOT) as a numerical number.

4.3.2.15 core:Location

Description

The location is the generic root element describing the topological structure of a site, building, space, etc. This generic location root node simplifies the replacement of the entire location model.

Sub Class Of

- core:Thing

Object Properties

- -

Data Properties

- core:comment

Disjoint With

- all first Ontology subclasses of core:Thing (see Clause 4.1.3)

Notes

The default location model used for the KIM is defined in Clause 4.3.2.15. Suppose the default location model does not fit. In that case, the location model can be changed by integrating its location model as an extension of the current location model with additional OPs or by defining new subclasses of Space better suited for the respective purpose.

EXAMPLE If a “Wing” entity as a new location is necessary as a part of a building, then a class “Wing” with specific wing properties can be added as a subclass to the class “Space”.

4.3.2.16 core:Point

Description

A Point is a means of interaction with a functionality of a device or service and provides an interface to data in the system.

A Point may be used in conjunction with its available subclasses described underneath; this allows for providing different degrees of informational value on a Point, such as to define its general operation types.

- class core:Datapoint
- class core:Actionpoint
- class core:Eventpoint

The subclasses are not defined as disjoint; this allows to describe a Point with an overlapping meaning from these subclasses.

EXAMPLE A point may be read from a client but may also spontaneously send its data; hence it is defined by its interface type (e.g. Datapoint and Eventpoint).

Sub Class Of

- core:Thing

Object Properties

- -

Data Properties

- core:comment

Disjoint With

- all first Ontology subclasses of core:Thing (see Clause 4.1.3)

Notes

Points can be mapped to interaction affordances from the Thing Description definition (properties, actions, events) and Properties from SOSA/SSN.

4.3.2.17 core:Datapoint

Description

A Datapoint exposes an internal state of a device that can be directly accessed (read) and optionally manipulated (write).

EXAMPLE A Datapoint with parameter properties to be written or diagnostic properties to be read.

Sub Class Of

- core:Thing

Object Properties

- tag:hasPointInterface, tag:operatesOnProtocol, knx:hasTagSet, knx:composes, tag:operatesForTrade, knx:datapointType, core:isMetaDataOf, core:influences

Data Properties

- core:readable, core:writable, jsonschema:minimum, jsonschema:maximum, knx:propertyIdentifier, knx:interfaceObjectType, knx:nonKNXmappingAddress

Disjoint With

- -

Notes

A Datapoint may have datapoint type information and/or additional Tag information from the Semantic Dictionary, including Quality Kinds and Operation Kinds. This allows specifying a Datapoint more precisely, deriving various processing functions to express any type of sensor, actuator, control device, and human interface.

A Datapoint with specific tags is also used to represent a standardized Datapoint of a Functional Block. For this, the correct DPT information and all necessary tags as part of the Semantic Dictionary shall be applied to the Datapoint.

The following examples explain the information quality of Datapoints depending on available tags:

1) Datapoint

Represents the minimal Datapoint informational value in a current MaC project (with at least information on the bit size and basic r/w and limitation).

2) Datapoint with DPT information

Represents the maximal Datapoint informational value in a current MaC project Installation

NOTE Gaining automatically additional tag information such as an input (tag:if.i) tag by evaluating combinations of GO communication flags is not considered here.

3) Datapoint with DPT and Tag information, Quality Kinds or Operation Kind

Represents a semantically detailed Datapoint, up to a standardized Datapoint of a Functional Block.

Intended Usage

core:readable, core:writable

It might be that a Datapoint cannot be read/written; this depends on the Installation's configuration. The flags can be used to represent the communication flags of a Group Object.

jsonschema:maximum, jsonschema:minimum

Limits the numerical Datapoint data values in its applicative behavior, such as the allowed temperature range of an RTC (Room Temperature Controller) setpoint; this limitation is not synonymous with the range from the Point (DPT) type information.

core:isMetaDataOf

For explanation, see details in Clause 4.3.3.

knx:hasTagSet

The OP is used with specific (child) OP relationship predicates to express an observation or actuation of specific properties a Datapoint is performing. The properties are defined as part of a QK and an Operation Kind.

knx:composes

A group of standardized Datapoints allows to express (to compose) a Functional Block; for details, see the corresponding OP description in Clause 4.6.5.

knx:interfaceObjectType

A standardized Datapoint of a FB is identified with a unique id (IOT) as a numerical number.

knx:propertyIdentifier

A standardized Datapoint of a FB is identified with a unique id (PID) as a numerical number.

knx:nonKNXmappingAddress

For details, see Function Point.

knx:datapointType

The OP adds datapoint type (DPT) information from the Semantic Dictionary to a Datapoint.

NOTE In an Installation, the DPT information is nearly in all cases assigned to a Datapoint; for a KNX Product, it shall be assigned.

4.3.2.18 core:Actionpoint

Description

An action point exposes a point that invokes an action on the receiver.

Sub Class Of

— core:Point

Object Properties

— -

Data Properties

— -

Disjoint With

— -

Notes

This point type is not used in this specification.

4.3.2.19 core:Eventpoint

Description

An event point triggers an event on the receiver (an occasion that does not need further data to be evaluated).

Sub Class Of

— core:Point

Object Properties

— -

Data Properties

— -

Disjoint With

— -

Notes

This point type is not used in this specification, except for the Semantic Export (see Appendix).

4.3.2.20 core:Product

Description

A Product is an item offered for sale; it can be assigned to any equipment. A product can be a device or a service; every product is made at a cost, and each is sold at a price.

Sub Class Of

- core:Thing

Object Properties

- -

Data Properties

- core:mediaType, knx:certificationStatus, core:manufacturer, core:orderNumber

Disjoint With

- all first Ontology subclasses of core:Thing (see Clause 4.1.3)

Notes

Manufacturers usually provide identifiers for each product, such as type/model, version number, or order number.

4.3.3 Relations

This clause describes specific OP relationship predicates used by core model concepts.

Object Property Relationship Predicate	Definition, Domain/Range
core:containsAsset/ core:isAssetOf	An OP relationship that allows defining the following relations: — the presence of equipment as part of an Installation — the presence of an asset included in other assets Domain core:Asset Range core:Asset
core:groups/ core:isGroupedBy	An OP relationship to express a relation to a Point from an Aspect. A Function Point (as a subclass of core:Aspect) is a Group Address created by a MaC, linking with the OP several Group Objects (as a Datapoint as a subclass of core:Point). Domain core:Aspect Range core:Point
core:hasPart/ core:isPartOf	An OP relationship to express the presence of (tangible) equipment, such as a Device, as part of a cabinet (that is also equipment). Domain core:Equipment Range core:Equipment

Object Property Relationship Predicate	Definition, Domain/Range
core:hasPoint/ core:isPointOf	An OP relationship to express a generic relation from functionality to several Points. The following sub-properties can be used to describe in detail which Point type is referenced. — core:hasActionPoint A functionality that refers to an action point. — core:hasDataPoint A functionality that refers to a Datapoint. — core:hasEventPoint A functionality that refers to an event point. Domain core:Functionality Range core:Point
core:hasProduct/ core:isProductOf	An OP relationship to express a relation to a Product from an equipment. Only equipment can have any information according to a product. Domain core:Equipment Range core:Product
core:hasProxy/ core:isProxyFor	This OP relationship allows to define a representative entry point for Datapoints in a gateway. Datapoints in an Installation can be exposed for different purposes, such as all Datapoints on a HBES IoT 3rd Party API (gateway). <u>Background</u> 1) The KIM expresses relations between Function Points and Points with the OP core:groups; the Points are related to a Device. 2) The Device expresses its proxy with the OP core:hasProxy. This chain of relations allows “forward/proxy” concepts of Function Points or Points of a gateway. The KIM generally allows expressing more than one proxy per Datapoint (aka FP, Point). — an FP is linked with several Points of different Devices; each of the devices has one single (but different) proxy — a Point is directly related to a Device; the device has several (different) proxies It is RECOMMENDED to express only one resulting proxy for each Datapoint. Moreover, Datapoints (Points) from a Device should have only one proxy. Domain core:Device Range core:Gateway
core:hosts/ core:isApplicationProgramOf	These OP relationships allow to define the existence of an Application Program as part of a Device. Domain core:Device Range core:ApplicationProgram