



BSI Standards Publication

Household appliances network and grid connectivity

Part 4-1: Communication Protocol Specific Aspects: SPINE, SPINE-IoT and SHIP

National foreword

This British Standard is the UK implementation of EN 50631-4-1:2023.

The UK participation in its preparation was entrusted to Technical Committee CPL/59, Performance of household electrical appliances.

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 16887 7

ICS 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 March 2023.

Amendments/corrigenda issued since publication

Date	Text affected

English Version

Household appliances network and grid connectivity - Part 4-1: Communication Protocol Specific Aspects: SPINE, SPINE-IoT and SHIP

Appareils domestiques connectés au réseau et réseau
intelligent - Partie 4-1: Aspects spécifiques des protocoles
de communication: SPINE, SPINE-IoT et SHIP

Netzwerk- und Stromnetz-Konnektivität von
Haushaltsgeräten - Teil 4-1: Spezifische Aspekte der
Kommunikationsprotokolle: SPINE, SPINE-IoT und SHIP

This European Standard was approved by CENELEC on 2023-02-13. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the CEN-CENELEC Management Centre or to any CENELEC member.

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

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



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

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

Contents

Page

European foreword.....	4
Introduction.....	4
1 Scope	6
2 Normative references	6
3 Terms and definitions	6
4 SPINE-IoT Protocol.....	10
4.1 General	10
4.2 Architecture overview	11
4.2.1 Introduction.....	11
4.2.2 API versioning	12
4.3 Device model	12
4.3.1 General	12
4.3.2 Device	13
4.3.3 Entity.....	13
4.3.4 Feature.....	14
4.4 Use Case model	16
4.4.1 General	16
4.4.2 Use Case information and instances	17
4.4.3 Use Case interface	19
4.5 Binding	19
4.5.1 General	19
4.5.2 Binding information and instances	20
4.6 Subscription.....	22
4.6.1 General	22
4.6.2 Subscription management	22
4.6.3 Callbacks	25
4.7 Requesting feature changes	27
4.7.1 General	27
4.7.2 Requesting changes information and instances	28
5 SPINE Protocol	30
5.1 General	30
5.2 Architecture overview	30
5.2.1 General rules.....	30
5.2.2 Common data types	31
5.2.3 Address level details.....	35
5.3 SPINE Datagram	36
5.3.1 Introduction.....	36
5.3.2 Header	37
5.3.3 Payload.....	45
5.4 Communication modes.....	58
5.4.1 General	58
5.4.2 Simple communication mode	58
5.4.3 Enhanced communication mode	59
5.5 Functional commissioning.....	59
5.5.1 General	59
5.5.2 Detailed discovery	60
5.5.3 Destination list.....	74
5.5.4 Binding	77
5.5.5 Subscription.....	86
5.5.6 Use Case discovery.....	94
6 SHIP	97
6.1 General	97
6.2 Architecture overview	97

6.2.1	General	97
6.2.2	General Considerations On Closing Communication Channels	98
6.2.3	SHIP Node Parameters	98
6.3	Registration	99
6.3.1	General	99
6.3.2	Successful Registration	101
6.3.3	Registration details and recommendations (informative)	101
6.4	Reconnection	102
6.4.1	General	102
6.4.2	Reconnection details in case of changed key material (informative)	102
6.5	Discovery	103
6.5.1	General	103
6.5.2	Service Instance	103
6.5.3	Service Name	103
6.5.4	Multicast DNS Name	103
6.5.5	Recommendations for re-discovery	105
6.6	TCP	105
6.6.1	General	105
6.6.2	Limited Connection Capabilities	105
6.6.3	Online Detection	106
6.6.4	TCP Connection Establishment	106
6.6.5	Retransmission Timeout	107
6.7	TLS	107
6.7.1	General	107
6.7.2	Cipher Suites	108
6.7.3	Maximum Fragment Length	108
6.7.4	TLS Compression	109
6.7.5	Renegotiation	109
6.7.6	Session Resumption	109
6.7.7	TLS extension for ECC	110
6.7.8	TLS Probing	110
6.8	WebSocket	111
6.8.1	General	111
6.8.2	TLS Dependencies	111
6.8.3	Opening Handshake	111
6.8.4	Data Framing	111
6.8.5	Connection Keepalive	112
6.9	Message Representation Using JSON Text Format	112
6.9.1	Introduction	112
6.9.2	Definitions	112
6.9.3	Examples For Each Type	113
6.9.4	XML to JSON Transformation	113
6.9.5	JSON to XML Transformation	120
6.10	Key Management	120
6.10.1	General	120
6.10.2	Certificates	121
6.10.3	SHIP Node Specific Public Key	125
6.10.4	Verification Procedure	127
6.10.5	Symmetric Key	133
6.10.6	SHIP Node PIN	134
6.10.7	SHIP Commissioning Tool	135
6.10.8	QR Code	137
6.11	SHIP Data Exchange	140
6.11.1	Introduction	140
6.11.2	Terms in the context of SHIP Data Exchange	140
6.11.3	Protocol Architecture / Hierarchy	142
6.11.4	SHIP Message Exchange	143
6.12	Well-known protocolId	183
	Annex A (normative) SHIP XSD	184
	Bibliography	191

European foreword

This document (EN 50631-4-1:2023) has been prepared by WG 07 “Smart Household Appliances” of CLC/TC 59X “Performance of household and similar electrical appliances”.

The following dates are fixed:

- latest date by which this document has to be implemented at national level by publication of an identical national standard or by endorsement (dop) 2024-02-13
- latest date by which the national standards conflicting with this document have to be withdrawn (dow) 2026-02-13

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.

Introduction

Energy management systems will more and more become necessary due to change from fossil and nuclear to renewable production and the associated decentralization. Since an appropriate standard for a home and building management is in preparation this document specifies how sets of products from multiple manufacturers can exchange information with Home and Building / Customer Energy Management Systems, located in a home network or in the cloud.

This document focuses on interoperability of household appliances and describes the necessary control and monitoring. It defines a set of functions of household and similar electrical appliances. The functions in this document cover next to energy-management main remote-control and – monitoring use cases.

This document does not deal with safety and security requirements. Safety requirements have been set in the IEC/EN 60335 series [1].

EN 50631 will provide interoperability on information exchange among various appliances in the home. The EN 50631 series will be re-arranged regarding the further development and will be split into 6 parts:

- EN 50631-1, *Household appliances network and grid connectivity — Part 1: General Requirements, Generic Data Modelling and Neutral Messages*
- EN 50631-2, *Household appliances network and grid connectivity — Part 2: Product Specific mappings, details, requirements and deviations*
- EN 50631-3-x, *Household appliances network and grid connectivity — Part 3: Specific Data Model Mapping*
- EN 50631-4-x, *Household appliances network and grid connectivity — Part 4: Communication Protocol Specific Aspects*
- EN 50631-5, *Household appliances network and grid connectivity — Part 5: General Test-Requirements and - Specification*
- EN 50631-6, *Household appliances network and grid connectivity — Part 6: SPINE Data Model Toolbox*

Data communication heavily depends on the environment of appliances. Sometimes low bitrate or energy efficient communication puts strict requirements to selected communication technologies. Therefore, popular and de facto standards had been and will be developed by the industry to fulfil such requirements. To not influence common data modelling for appliances because of such restrictions, the standardized data models and neutral message structures need to be applied to communication technologies.

This standard series therefore is intended to separate data modelling and neutral message structure from the attached communication.

Part 1 defines general requirements, generic data modelling and generic neutral messages without relation to any specific communication technology or any product specific layout.

Part 2 lists and specifies product specific requirements and implementation guidance based on the generic data model and generic neutral messages.

Part 3 defines the mapping of neutral messages to examples of typical data models like SPINE, OCF, and so forth. These data models are neither mandatory nor to be seen as complete spectrum of data models.

Part 4 defines the mapping of neutral messages to examples of typical communication protocols. These communication protocols are neither mandatory, nor do they provide an exhaustive list of communication protocols.

Part 5 defines testing requirements and testing specifications. This part will be covered in the future by a New Work Item Proposal.

Part 6 provides the technical reference specification for the SPINE data model. This part will be covered in the future by a New Work Item Proposal.

1 Scope

This document specifies the application of relevant transport protocols for Home and Wide Area Networks as well as cloud connectivity; in this case, SPINE (Smart Premises Interoperable Neutral-Message Exchange), SPINE-IoT, and SHIP (Smart Home IP).

This document is part of the EN 50631 series, which defines the information exchange between Smart Appliances and management systems in homes and buildings including energy management.

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.

IETF RFC 793:1981, *Transmission Control Protocol*

IETF RFC 3280:2002, *Internet X.509 Public Key Infrastructure Certificate Revocation List (CRL) Profile*

IETF RFC 6455:2011, *The WebSocket Protocol*

IETF RFC 6763, *DNS-Based Service Discovery*

3 Terms and definitions

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

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

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

3.1

CA

Certificate Authority

Certification Authority

entity which can provide a digital signature for certificates

Note 1 to entry: Other SHIP nodes can check this digital signature with the certificate from the CA itself, the “CA-certificate”.

3.2

Commissioning Tool

<SHIP> instrument to establish the trust between different devices in the smart home installation, e.g. distribute trustworthy credentials from some SHIP nodes to other SHIP nodes

Note 1 to entry: E.g. a smart phone, a web server or a dedicated device can embody the role of a commissioning tool. So far, the SHIP specification does not specify a commissioning tool; an interoperable protocol for commissioning can be used on the layer above SHIP.

Note 2 to entry: A manufacturer may also use their own solutions.

3.3

DNS

Domain Name System

[SOURCE: IETF RFC 1035]

3.4**DNS host name**

fully qualified domain name used within DNS as host name to get the IP address of the corresponding internet host

3.5**DNS-SD**

Domain Name System – Service discovery

[SOURCE: IETF RFC 6763]

3.6**EUI**

Extended Unique Identifier

Note 1 to entry: <https://standards.ieee.org/develop/regauth/tut/eui64.pdf>

3.7**factory default**

setting that allows the user to reset the SHIP node to the as-new condition

Note 1 to entry: This means that all data that has been provided and stored by the SHIP node during its operation time SHALL be deleted.

3.8**IANA**

Internet Assigned Numbers Authority

3.9**IP**

Internet Protocol

3.10**LAN**

Local Area Network

3.11**MAC**

Media Access Control

3.12**mDNS****multicast DNS host name**

fully qualified domain name used within mDNS as host name to get the IP address of the corresponding local SHIP node

3.13**M/O/NV/C**

abbreviations which refer to:

1. M = mandatory
2. O = optional
3. NV = not valid
4. C = choice, i.e. a presence or support depends also on the selection from multiple possibilities

and which are primarily used within specific definition tables describing certain specialized data model definitions

3.14

numerical representation

written system for expressing numbers

EXAMPLE 0xab represents a decimal value of 171

3.15

PIN

Personal Identification Number

specification which makes use of a PIN as secret for SHIP specific verification procedures

3.16

PKI

Public Key Infrastructure

3.17

Push Button

switching mechanism to control some aspects of a machine or a process

Note 1 to entry: A push button event does not necessarily mean that a real physical button has to be used to trigger this event. A push button event may also be generated by other means, e.g. via a smart phone application or a web-interface (secure connection to SHIP node required). A push button shall provide a simple mechanism for a user to bring the device to a certain state or start a certain process.

3.18

QR Code

registered trademark of DENSO WAVE INCORPORATED which is the short form for “Quick Response Code” and used for efficient encoding of data into a small graphic

Note 1 to entry: Among others, international standard ISO/IEC 18004:2015 specifies the encoding of QR code symbols.

3.19

RFC

request for comments

3.20

SHIP

Smart Home Internet Protocol

Note 1 to entry: This term is used throughout this document to refer to the described communication protocol.

3.21

SHIP ID

identification which is used to uniquely identify a SHIP node, e.g. in its service discovery, and which is present in the mDNS/DNS-SD local service discovery

Note 1 to entry: Each SHIP node has a globally unique SHIP ID.

Note 2 to entry: See 6.5.

3.22

SHIP Client

role which is assigned to the SHIP node that also embodies the TCP client role for a specific peer-to-peer connection

3.23**SHIP Commissioning**

<SHIP> term which denotes the distribution of trustworthy SKIs from certain SHIP nodes to other SHIP nodes

Note 1 to entry: The distribution of the trustworthy SKIs is handled by a so-called SHIP commissioning tool.

3.24**SHIP Commissioning Tool**

instrument which is used to distribute trustworthy SKIs from certain SHIP nodes to other SHIP nodes, and which allows a user to handle the trust relationships in the whole SHIP installation (if each node in the installation supports commissioning) over one simple user interface

3.25**SHIP Node**

logical device which communicates via the described SHIP protocols and can be integrated into a web server or physical device

Note 1 to entry: One physical device may have more than one logical SHIP node. In this case, each SHIP node MUST use distinct ports (e.g. a physical device provides 3 open ports with 3 different SHIP services).

3.26**SHIP Server**

role which shall be assigned to the SHIP node that also embodies the TCP server role for a specific peer-to-peer connection

3.27**SKI****Subject Key Identifier**

identifying certificates that contain a specific public key and is used as a cryptographically backed identification and authentication criterion

Note 1 to entry: Each SHIP node has a specific public key.

3.28**SPINE**

Smart Premises Interoperable Neutral message Exchange
Technical Specification of EEBus Initiative e.V

3.29**Trusted SHIP Node**

term only applicable from a specific SHIP node point of view

Note 1 to entry: If SHIP node A has a communication partner and a trusted relationship to SHIP node B, SHIP node B is called a trusted SHIP node from SHIP node A's point of view; a trusted relationship can be established in different ways.

3.30**UCS**

Universal Character Set

3.31**UTF**

UCS Transformation Format

computing industry standard for the consistent encoding, representation, and handling of text expressed in most of the world's writing systems

3.32**WAN**

Wide Area Network

3.33**web server based SHIP node**

SHIP node that is hosted by a web server

3.34**WiFi**

IP networks based on the IEEE 802.11 set of standards, used for wireless IP communication

4 SPINE-IoT Protocol**4.1 General**

This document specifies the application of relevant transport protocols for Home and Wide Area Networks as well as cloud connectivity; in this case, SPINE (Smart Premises Interoperable Neutral-Message Exchange), SPINE-IoT, and SHIP (Smart Home IP).

Figure 1 shows the use of the transport protocols within this document. In case of local operation (the Smart Appliance is connected to a local Customer Connectivity Manager via the HAN), the transport protocols SPINE and SHIP SHALL be used. In case of IoT/Cloud operation (the Smart Appliance communicates via its cloud representation or directly with an IoT/cloud based Customer Connectivity Manager), the transport protocol SPINE-IoT via HTTP / REST API / Open API SHALL be used.

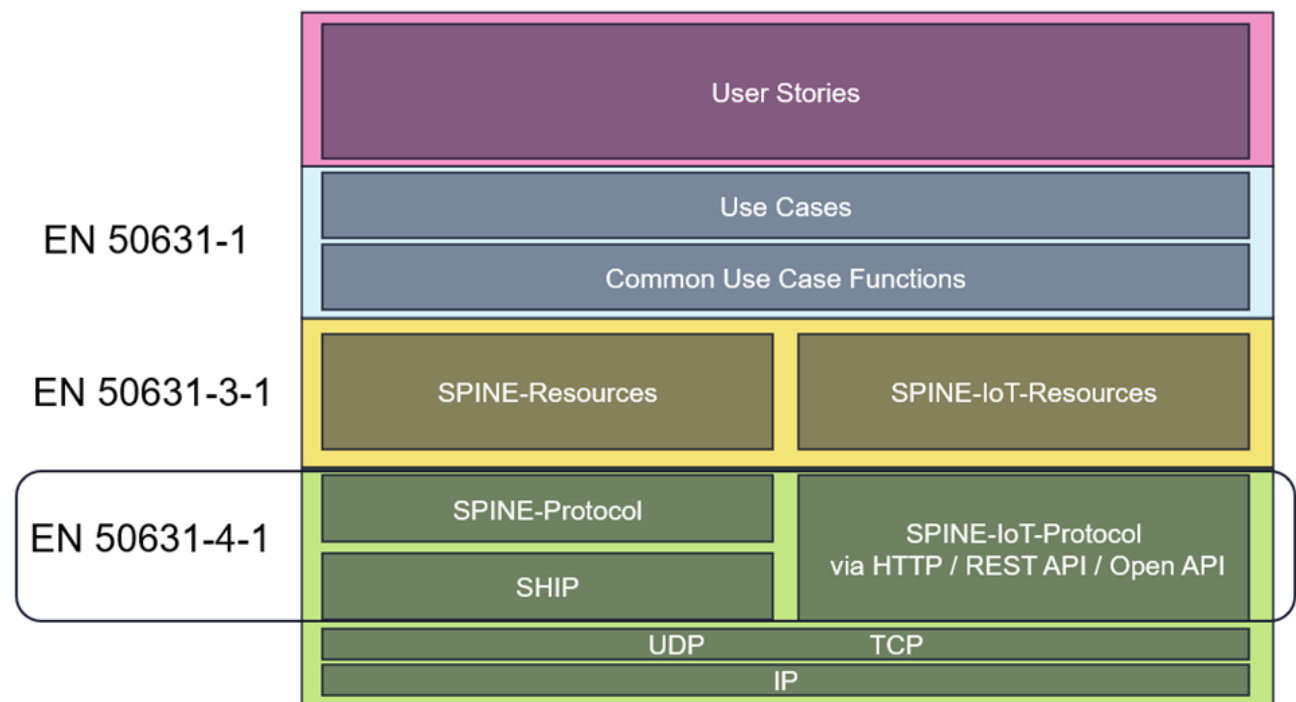


Figure 1 — Overview of transport protocols within EN 50631-4-1

SPINE-IoT (Smart Premises Interoperable Neutral-message Exchange for Internet of Things) defines a neutral IP-based layer which helps connecting different communications technologies (i.e. Wi-Fi, Ethernet, Mobile telephony) to build a smart home / smart grid system. This allows devices to talk directly to the cloud-based energy management system as well as other IoT devices. This direct connection permits different manufactures to keep a connection to their devices and their users, and having access to important data of connected devices.

4.2 Architecture overview

4.2.1 Introduction

SPINE-IoT is based upon the OpenAPI specification to model an API that permits interactions with devices. Devices can comprise sub-devices, called entities. The data of the entities is organized in so-called features. A feature is an address with an explicitly named data type.

One of the most important capabilities of SPINE-IoT is the explicit support of Use Cases. Each device can report which Use Cases it supports. Via so-called Use Case interfaces it is possible to work with the device's data (i.e. the features) and hence the functionality in a more simple way in comparison to work directly with the features: A Use Case interface permits to get specifically data (features) of the requested Use Case and it can provide Use Case specific commands to alter the data (e.g. to trigger a specific behaviour at the device).

The API models all device, entity, and feature information, which permits to analyse the devices in detail. However, a client may also search directly for the supported Use Cases instead. Once a client is configured for selected Use Case instances, the further interactions can basically be focused on the Use Case interfaces.

The rest of this introduction provides some more technical details on the architecture.

The OpenAPI Specification defines a standard, language-agnostic interface to RESTful APIs which allows both humans and computers to discover and understand the capabilities of a service.

An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases [2].

The SPINE-IoT Protocol is an OpenAPI-based specification to model ONE API of a given (cloud) provider with HTTP protocol concepts and JSON payload. It specifies a LIST of devices. Each device can have a LIST of Use Cases. The API model is available as YAML files (see EN 50631-3-1).

Details on device model will be explained in subsequent sections. With regard to a device's functional data, there are in general two interaction styles possible:

- Features: A feature is the core of the API design, it models and represents the application domain. The feature is uniquely defined with by the tuple (deviceId, entityId, featureObjType, featureType).
- Use Case Interfaces: A Use Case Interface retrieves the features of the requested Use Case instances. Many Use Cases require the use of more than one feature.

A Use Case defines actors with particular roles and information (data) exchange to fulfil a defined task. The Use Case discovery allows to discover which Use Cases are supported and which actor a device embodies in a corresponding Use Case. This allows to derive information about which data a device supports as a client or as a server, as defined by each Use Case scenario.

Use Case discovery simplifies finding the right entity address where a server's data of a Use Case can be found. It also permits announcement of Use Cases that are available only conditionally.

Once a client read out present data at the required addresses, it can compare the data with requirements stated by a Use Case specification to see if all requirements are fulfilled for a given scenario.

Next to Use Case discovery and Use case interfaces, SPINE-IoT supports the following mechanisms:

- Discovery: Allows to discover devices and their components and data.
- Binding: A binding is required to obtain particular permissions to control a (part) of a device.
- Subscription: Subscribes to an event stream to monitor for changes.

In this document Oauth2 will be used according to RFC 6749 [3]. Oauth2.0 Authorization Code Grant Flow is used to get an access token. The Oauth2 requires that a client is registered on the authorization server before the client can be granted access to the resources of the resource server. The Oauth2 does not oblige the communication patterns (data and its format) to share necessary information between a resource server and an authorization server. The client needs to use the access token as bearer token in each request to the

API. The security tokens contain information about authorization and are intended to be used for accessing resources on a resource server or refreshing an access token.

Some definitions and values in SPINE-IoT should be identical to SPINE. The identical definitions and values are explained in details in Clause 5.

4.2.2 API versioning

The API SHALL be versioned to allow compatibility between communicating devices. Only as a whole and not in parts, even if several elements of the API are unchanged between different API versions.

SPINE-IoT follows the version numbering scheme according to the “Semantic Versioning 2.0.0” specification [4]. The SPINE-IoT version number has the format MAJOR.MINOR.PATCH, with the major, minor and patch constituents being non-negative integer numbers as specified in the “Semantic Versioning 2.0.0” specification [4].

This edition defines the major version “1”, with the server variable “majorVersion” set to “v1” accordingly (see YAML models in EN 50631-3-1). This technical solution defines the SPINE-IoT Protocol Specification version 1.0.0.

For a given MAJOR.MINOR.PATCH version, the patch number is increased if and only if the documentation contains backwards compatible corrections to the previous specification version. For a given MAJOR.MINOR version, the minor version is increased if the specification is extended with backwards compatible extensions. For a given MAJOR version, the major version is increased if incompatible changes to the previous version are introduced.

For a given base URL, the server provides exactly one version of the API. This version belongs to exactly one major version. A server may provide other versions of the API at different base URLs.

For information on the API itself and potentially further API versions supported by the server (at other base URLs), this path is available:

- Path “/api”

Access to information through this path depends on the authorization scope. In particular, a command like “GET /api” needs to return information on other API versions only if the authorized requester is permitted to access the other versions or at least can ask for access to the other versions.

The properties of a successful response are shown in Table 1.

Table 1 — Properties of “api” information

Element name	Type	M/O/C	Brief explanation
thisVersion	string	M	The version number of this API. The format is “MAJOR.MINOR.PATCH” (without quotation marks).
otherVersions	array of string	O	Array of further version numbers supported by the server (at other URLs). For each major version, it is sufficient to specify just the highest version number.

4.3 Device model

4.3.1 General

Like SPINE, a SPINE-IoT device is subdivided into one or more entities (see Clause 5). An entity represents a logical sub-device. Each entity can comprise other entities or so-called “features”. A SPINE-IoT feature is a collection of data, e.g. a measured power value or a table of time-dependent prices, as described in Figure 2.

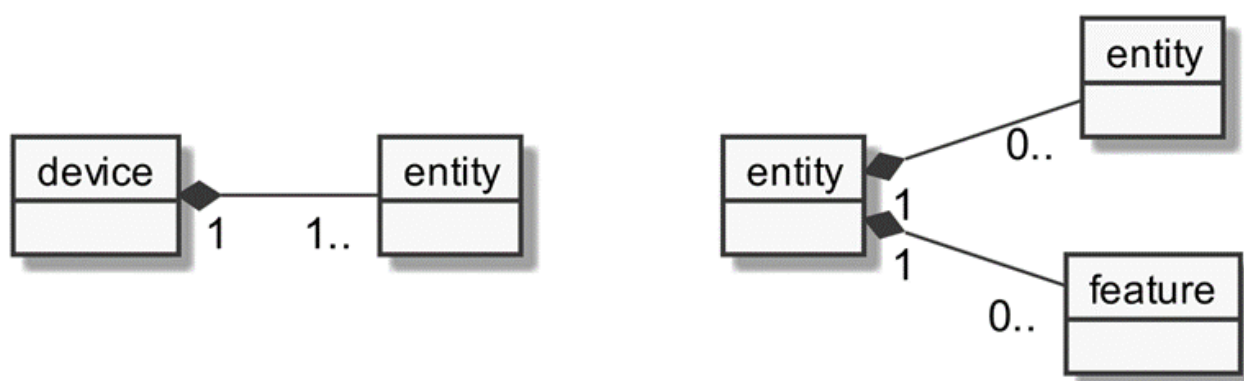


Figure 2 — Primary device model

4.3.2 Device

In the SPINE-IoT model, each level can be accessed individually. For the device information level, this path is available:

— Path “/devices”:

This path provides information on available devices.

The access to information through these paths and other SPINE-IoT paths depends on the authorization scope. In particular, a command like “GET /devices” will only return information on those devices where the authorized requester was granted sufficient access rights.

Within the given API, each device is uniquely identified by its deviceId. The device properties of a successful response are shown in Table 2. Please note that in this subclause only the properties of a single device are described, not the complete response. The full response is described in EN 50631-3-1 (see YAML models).

With a GET command on the above-mentioned paths it is possible to get an overview on available device types and available entity types or to query for a particular device or device type. The deviceId is an essential parameter in queries of almost all paths if information of a particular device is required.

Table 2 — Properties of “device” information

Element name	Type	M/O/C	Brief explanation
deviceId	string	M	SPINE-IoT device identifier. SHALL be unique within the API.
deviceAddress	string	O	SPINE device address. SHALL be unique. See Clause 5.
deviceType	string	M	SPINE device type. See Clause 5.
entitiesOverview.	object	M	Overview of available entity properties.
entitiesOverview. entityTypes	array of string	M	Array of SPINE entity types that are available for the given device. See Clause 5.

4.3.3 Entity

For the entity information level, this path is available:

— Path “/entities”

Access to information through this path depends on the authorization scope. In particular, a command like “GET /entities” will only return information on those entities where the authorized requester was granted

sufficient access rights. Access to an entity SHALL NOT be given if there are no sufficient access rights for any of its parent entities or its device.

Within the given API, each entity is uniquely identified by the tuple (deviceId, entityId). The entity properties of a successful response are shown in Table 3.

With a GET command on this path it is possible to get an overview on available entity types and available use cases and feature object types or to query for particular properties like an entity type. The deviceId and entityId are essential parameters in queries of several paths if information of a particular entity of a particular device is required.

Table 3 — Properties of “entity” information

Element name	Type	M/O/C	Brief explanation
deviceId	string	M	SPINE-IoT device identifier. SHALL be unique within the API.
entityId	integer	M	SPINE-IoT entity identifier of a given device. The tuple (deviceId, entityId) SHALL be unique within the API.
parentEntityId	integer	O	SPINE-IoT entity identifier of the parent entity. Absent for a device's top-level entities.
entityType	string	M	SPINE entity type. See Clause 5.
usecasesOverview.	array of object	O	Overview of available Use Cases.
usecasesOverview. name	string	M	The (technical) Use Case discovery name (defined per Use Case).
usecasesOverview. usecaseMajorVersion	string	M	The major version of the Use Case. If the detailed version is “1.2.0”, the major version is “v1” (without quotation marks). The usecaseMajorVersion is also used as parameter in the path “/usecaseInterfaces/{usecaseName}/{usecaseMajorVersion}”.
usecasesOverview. actor	string	M	A (technical) Use Case actor name (defined per Use Case).
featuresOverview.	object	M	Overview of available features.
featuresOverview. featureObjTypes	array of string	M	Array of available SPINE-IoT feature object types.
featuresOverview. featureTypes	array of string	M	Array of available SPINE feature types. See Clause 5.

As can be seen from Table 3, a parent-child-relationship between two entities (see Figure 2 is expressed with the property parentEntityId.

Before a server (the provider of this API) removes a particular entity, it SHALL remove all of the entity's child entities and features.

NOTE A client can be informed about the removal of entities and features via the subscription mechanism, see 4.6.

4.3.4 Feature

For the feature information level, this path is available:

— Path “/features”

Access to information through this path depends on the authorization scope. In particular, a command like “GET /features” will only return information on those features where the authorized requester was granted

sufficient access rights. Access to a feature SHALL NOT be given if there are no sufficient access rights for its entity.

Within the given API, each feature is uniquely identified by the tuple (deviceId, entityId, featureObjType, featureType). However, a feature might comprise more than one instance of a featureObjType in case of list-like data. This will be explained in more detail below.

SPINE-IoT applies basically the same feature object type/feature type concept as SPINE:

- The feature object type defines a particular collection of properties and their types.
- The feature type defines a basic set of rules on a collection of feature object types.

Due to the different nature of the underlying OpenAPI, SPINE-IoT feature object types are modelled a bit differently in comparison to SPINE.

With a GET command on this path it is possible to get an overview on available feature object types and their data or to query for the data of particular kinds of features. The deviceId, entityId, featureObjType, and featureType are essential parameters in queries if information of a particular featureObjType-featureType is required.

A feature can comprise a single object or it can comprise a list of objects of the same type, which is in both cases the featureObjType. More precisely, for each featureObjType it is specified whether a feature can contain just one instance or a list of instances of the featureObjType (this will be detailed subsequently).

The properties of a single featureObjType instance of a successful response on a GET command are shown in Table 4. In general, a GET on /features returns an array of featureObjType instances. For a feature with a list of featureObjType instances, Table 4 will occur in the response for each of these instances, with the “data” property carrying the respective instance.

Table 4 — Properties of “feature” information of a single featureObjType instance

Element name	Type	M/O/C	Brief explanation
deviceId	string	M	SPINE-IoT device identifier. SHALL be unique within the API.
entityId	integer	M	SPINE-IoT entity identifier of a given device. The tuple (deviceId, entityId) SHALL be unique within the API.
featureObjType	string	M	SPINE-IoT feature object type. This property determines the content type of property “data”.
featureType	string	M	SPINE feature type. See Clause 5. This property determines basic rules that are applied to the given feature.
data	oneOf ... (see text)	M	The content type of “data” SHALL correspond exactly to the feature object type denoted by the property “featureObjType”. The feature definitions are described in EN 50631-3-1 (see YAML models for SPINE-IoT).

Table 5 shows the feature object types that are supported in this document.

Table 5 — Supported featureType-featureObjType combinations

featureType value (string)	featureObjType value (string)
PowerSequences	powerSequenceAlternativesRelation
PowerSequences	powerSequenceNode
PowerSequences	powerSequence
PowerSequences	powerTimeSlot

The modification of a feature via POST command is described in 4.7.

On list-based features: As mentioned above, a feature may contain a list of featureObjType instances. A featureObjType permits being used this way only if it defines so-called identifiers. These identifiers permit unique identification of the list items. Each featureObjType specifies its own set of identifiers. For a given featureObjType, this permits also GET commands on a featureObjType specific path, with the possibility to query for individual identifiers (e.g. sequenceId and slotNumber), which is not considered further in this subclause (for more details, see YAML models in EN 50631-3-1). With regard to POST commands on the path /features (see 4.7) or callbacks from a feature (see 4.6.3), these identifiers are used to express modifications on single featureObjType identifiers.

4.4 Use Case model

4.4.1 General

Like SPINE (see Clause 5), SPINE-IoT models a Use Case through features and provides a possibility to discover available Use Cases. As a Use Case is based upon one or more features of an entity (dotted lines in Figure 3), a Use Case instance belongs to the respective entity (dashed lines in Figure 3).

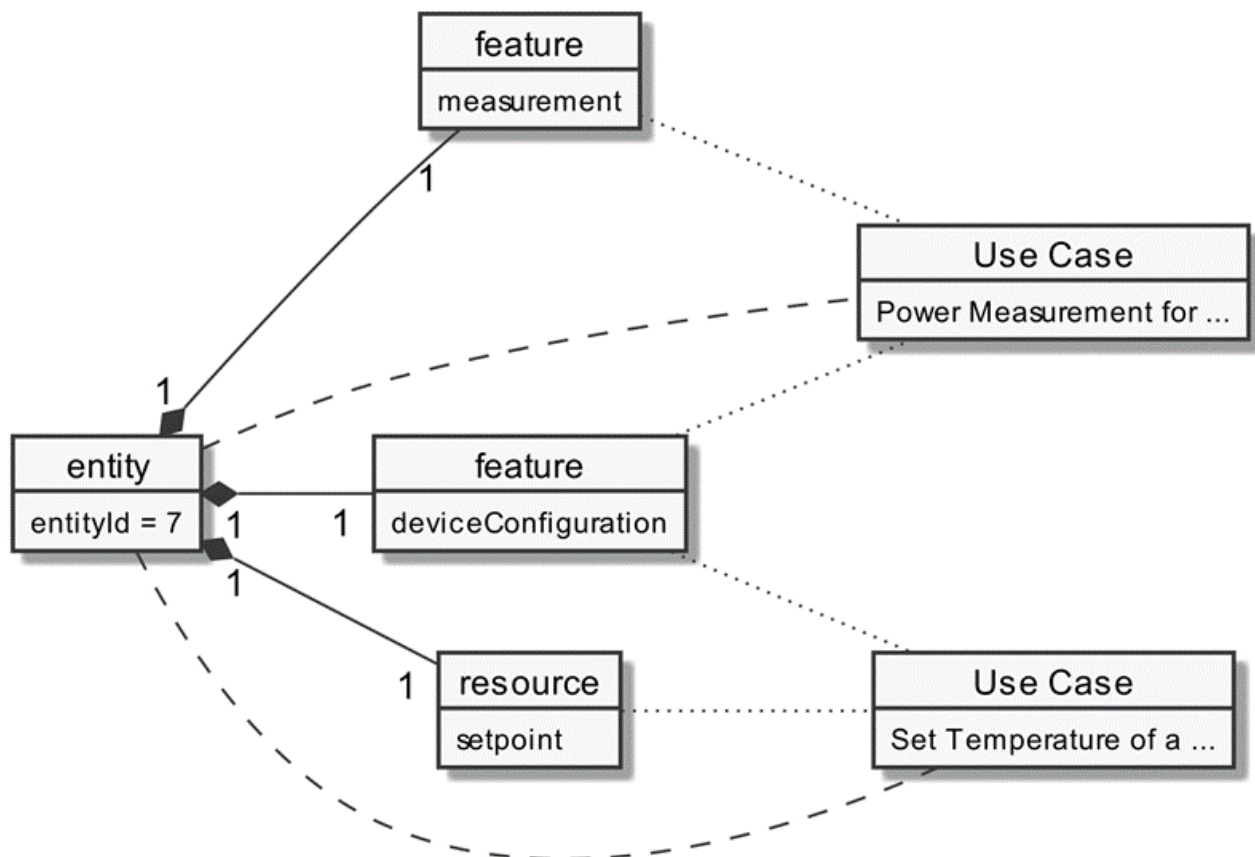


Figure 3 — Use Case instance examples

SPINE-IoT provides two basic paths for Use Cases:

- Path “/usecases”

This path can be used to discovery available Use Cases.

- Path “/usecaseInterfaces”

This path can be used to access a Use Case's data and functionality more easily in comparison to the direct access via the path “/features”.

4.4.2 Use Case information and instances

Use Case information is defined as detailed information on the support of a Use Case and the availability and location (address) of a Use Case instance. This includes cases where a device can support a Use Case, but the required entity is currently not available.

A Use Case instance is defined as an entity with required features holding the device's data for this particular instance of the Use Case.

The following descriptions will explain these terms in more detail.

For the Use Case information level, this path is available:

- Path “/usecases”

Access to information through this path depends on the authorization scope. In particular, a command like “GET /usecases” will only return information on those Use Cases where the authorized requester was granted sufficient access rights. Access to a Use Case information SHALL NOT be given if there are no sufficient access rights for its related features.

Figure 3 shows two Use Case instances on an entity with entityId = 7. In some cases, a Use Case instance might not be available all the time. A typical example is a charging station where currently no electric vehicle is plugged in. Nevertheless it is beneficial for a charging station to express which Use Cases it might provide once an electric vehicle is plugged. An API client can then decide to continue listening for the appearance of a Use Case instance. SPINE-IoT supports such situations with a Use Case information on a parent entity. In Figure 4, a Use Case information is announced for entityId = 1, but with the additional information that the Use Case is not available on this entity itself (isOnThisEntity = false).

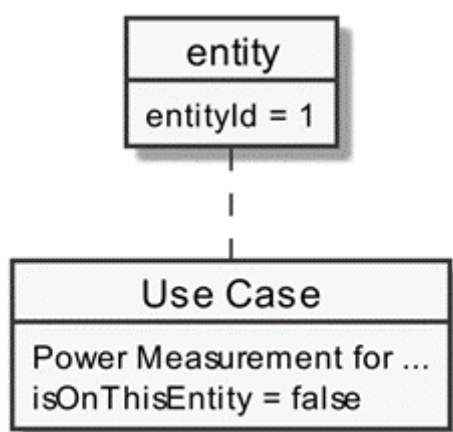


Figure 4 — Announcement of potential Use Case instance

As soon as there is an entity providing a real Use Case instance, another Use Case information is announced. In Figure 5, this instance is announced for entityId = 7 (with isOnThisEntity = true).

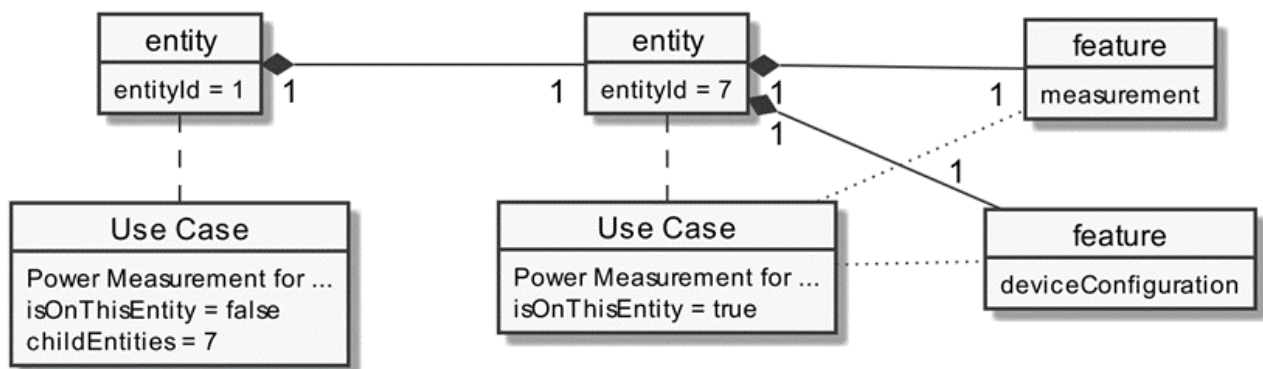


Figure 5 — Potential Use Case with real instance

Within the given API, each Use Case information is uniquely identified by the tuple (deviceId, entityId, usecase.name, usecase.usecaseMajorVersion, usecase.actor). The Use Case information properties of a successful response are shown in Table 6.

With a GET command on this path it is possible to get an overview on available Use Cases or to query for a particular Use Case information. The deviceId, entityId, usecase.name, and usecase.usecaseMajorVersion are essential parameters in queries to get an overview with some details on the support of Use Cases that are of interest.

Table 6 — Properties of “Use Case” information

Element name	Type	M/O/C	Brief explanation
deviceId	string	M	SPINE-IoT device identifier. SHALL be unique within the API.
entityId	integer	M	SPINE-IoT entity identifier of a given device. The tuple (deviceId, entityId) SHALL be unique within the API.
usecase.	object	M	
usecase.name	string	M	The (technical) Use Case discovery name (defined per Use Case).
usecase.usecaseVersion	string	M	The detailed version number of the Use Case. Example: “1.2.0”.
usecase.usecaseMajorVersion	string	M	The major version of the Use Case. If the detailed version is “1.2.0”, the major version is “v1” (without quotation marks). The usecaseMajorVersion is also used as parameter in the path “/usecaseInterfaces/{usecaseName}/{usecaseMajorVersion}”.
usecase.actor	string	M	A (technical) Use Case actor name (defined per Use Case).
usecase.scenarioSupport	array of integer	M	The list of the supported scenarios of this Use Case.

Element name	Type	M/O/C	Brief explanation
usecase. isOnThisEntity	boolean	M	If set to “true”, the entity of this Use Case information hosts the Use Case data. In this case, the property childEntities is not relevant. If isOnThisEntity is “false”, this Use Case information just serves to announce that one or more existing or potentially upcoming child entities may host Use Case data. In this case, these Use Cases' entities are listed in the property childEntities.
usecase. childEntities	array of integer	M	See description of property isOnThisEntity.

With this information, a client is able to get the addresses of Use Case instances.

Within the given API, each Use Case instance is uniquely identified by the tuple (deviceId, entityId, usecase.name, usecase.usecaseMajorVersion), where entityId has to be set to a value where required resources are available. With regard to Table 5, this means entityId is

- either the property entityId of the Use Case information if “usecase. isOnThisEntity” is true,
- or an item of the property “usecase. childEntities” of the Use Case information if “usecase. isOnThisEntity” is false.

This can be used to access the Use Case instance's data via the path “/features” (see 4.3.4) or via a Use Case interface (see 4.4.3).

4.4.3 Use Case interface

For Use Case specific interfaces, this path is available:

- Path “/usecaseInterfaces/{usecaseName}/{usecaseMajorVersion}”

The path parameter usecaseName corresponds to the property “usecase. name” of Table 6. The path parameter usecaseMajorVersion corresponds to the property “usecase. usecaseMajorVersion” of Table 6.

Access to information through this path depends on the authorization scope. In particular, a command like “GET /usecaseInterfaces/{usecaseName}/{usecaseMajorVersion}” will only return features of those Use Case instances where the authorized requester was granted sufficient access rights. Access to a Use Case instance SHALL NOT be given if there are no sufficient access rights for its related features.

A GET command on such a path retrieves the features of the requested Use Case instances. Together with the query parameters deviceId and entityId, the response of the GET command would contain only the features of a single Use Case instance.

Many Use Cases require the use of more than one feature. With the path “/features”, several GET commands may then be required to retrieve all data. With a Use Case interface path, on the other hand, the same data can be requested with a single GET command.

A Use Case may define particular POST commands. Together with the query parameters deviceId and entityId, the POST command would affect only the features of a single Use Case instance.

4.5 Binding

4.5.1 General

In many cases, a device needs to request a binding at another device to get the permission for controlling a particular part of the other device. This is primarily relevant to change data at the other device, e.g. a limitation for the device's power consumption. The binding mechanism can be used to limit the number of devices that are given the permission for certain changes. With regard to Use Cases, there is frequently only one actor (or at least a very limited number) permitted to execute certain scenarios with another actor.

4.5.2 Binding information and instances

Bindings can be requested (i.e. created), renewed, deleted and listed at this path:

— Path “/bindings”

Requesting, renewing and deleting bindings through this path depends on the authorization scope. Bindings can only be requested, renewed and deleted on those features where the authorized requester was granted sufficient access rights. Requesting, renewing and deleting bindings SHALL NOT be permitted by the API if there are no sufficient access rights for any of the requested features or Use Case interfaces.

A binding can be requested for features or for Use Case interfaces. A binding on a feature is rather unspecific as it does not express for which purpose the binding is requested. In contrast, a binding on a Use Case interface can express the relationship and responsibilities of two actors in the given Use Case.

With a POST command on this path a binding is requested. The POST command takes a request body according to Table 7.

Table 7 — Request body for POST /bindings

Element name	Type	M/O/C	Brief explanation
	oneOf bindingRequestFeatures bindingRequestUsecaseInterf aces	M	To create a binding on features, use type bindingRequestFeatures. To create a binding on usecaseInterfaces, use type bindingRequestUsecaseInterfaces.

With the type bindingRequestFeatures, the POST command contains information to bind on features. The type is shown in Table 8.

Table 8 — Properties of bindingRequestFeatures

Element name	Type	M/O/C	Brief explanation
features.	object	M	A binding to one or more features is requested. The request is about those features that match ALL of the set child properties of “features”.
features. deviceId	string	O	SPINE-IoT device identifier. SHALL be unique within the API.
features. deviceType	string	O	SPINE device type. See Clause 5.
features. entityId	integer	O	SPINE-IoT entity identifier of a given device. The tuple (deviceId, entityId) SHALL be unique within the API.
features. entityType	string	O	SPINE entity type. See Clause 5.
features. featureObjType	string	O	SPINE-IoT feature object type.
features. featureType	string	O	SPINE feature type. See Clause 5.

With the type bindingRequestUsecaseInterfaces, the POST command contains information to bind on usecaseInterfaces. The type is shown in Table 9.

Table 9 — Properties of bindingRequestUsecaseInterfaces

Element name	Type	M/O/C	Brief explanation
usecaseInterfaces.	object	M	A binding to one or more Use Case interfaces is requested. The request is about those Use Case interfaces that match ALL of the set child properties of “usecaseInterfaces”.
usecaseInterfaces. deviceld	string	O	SPINE-IoT device identifier. SHALL be unique within the API.
usecaseInterfaces. entityld	integer	O	SPINE-IoT entity identifier of a given device. The tuple (deviceld, entityld) SHALL be unique within the API.
usecaseInterfaces. usecaseName	string	M	The (technical) Use Case discovery name (defined per Use Case).
usecaseInterfaces. usecaseMajorVersion	string	M	The major version of the Use Case. If the detailed version is “1.2.0”, the major version is “v1” (without quotation marks). The usecaseMajorVersion is also used as parameter in the path “/usecaseInterfaces/{usecaseName}/{usecaseMajorVersion}”.
usecaseInterfaces. actor	string	M	A (technical) Use Case actor name (defined per Use Case).
usecaseInterfaces. requester.	object	O	Information on the sender of the binding request.
usecaseInterfaces. requester. actor	string	M	The Use Case actor name (defined per Use Case) of the requester.
usecaseInterfaces. requester. usecaseVersion	string	M	The detailed version number of the Use Case supported by the requester. Example: “1.2.0”.
usecaseInterfaces. requester. deviceld	string	O	SPINE-IoT device identifier of the requester. SHALL be unique within the API. This property is only relevant if a Use Case defines that both actors need to provide an API (i.e. if both actors need to provide data and functionality that the respective other actor needs to access).
usecaseInterfaces. requester. entityld	integer	O	SPINE-IoT entity identifier of the requester. The tuple (deviceld, entityld) SHALL be unique within the API. This property is only relevant if the requester's deviceld property is relevant.

If the API accepts the binding request, it responds with a success code and the created bindingId according to Table 10. The created binding MAY expire at the time stated in the property “expires”. The duration from the creation of the binding to its expiration is determined by the server and SHOULD be at least 24h. Subsequently it will be described how a client can renew a binding.

Table 10 — Response body upon accepted binding request

Element name	Type	M/O/C	Brief explanation
bindingId	string	M	Identifier of the created binding. SHALL be unique within the API.
expires	date-time	O	The binding will be released (deleted) by the server at this time. A client can renew this binding only before this time is reached. If expires is absent, the server SHOULD NOT remove the binding autonomously.

With a PATCH command on this path, a specific binding or all of the requester's bindings can be renewed. The PATCH command takes a request body according to either Table 11 or Table 12.

Table 11 — Renew all bindings

Element name	Type	M/O/C	Brief explanation
renewAll	boolean, only "true" allowed	M	The client requests for the renewal of all of its bindings.

Table 12 — Renew a single bindingId

Element name	Type	M/O/C	Brief explanation
renewBindingId	string	M	The client requests for the renewal of its bindingId as set in renewBindingId.

If the PATCH command is accepted, the server responds with an array of items with properties as shown in Table 10, containing the bindingIds where the expiry has been updated. The duration from the renewal of the binding(s) to their expiration is determined by the server and SHOULD be at least 24h.

With a DELETE command on this path, a specific binding or all of the requester's bindings can be deleted. The DELETE command takes a request body according to Table 13. At least one of the properties SHALL be set.

Table 13 — Request body for the deletion of one or all bindings

Element name	Type	M/O/C	Brief explanation
bindingId	string	O	Identifier of the binding that shall be deleted. SHALL be unique within the API.
all	boolean	O	If set to "true", all bindings of the requester shall be deleted. In this case, the property bindingId SHALL be ignored if present.

With a GET command on this path it is possible to get an overview on all bindingId values that are still valid. The response is an array of items with properties as shown in Table 10.

4.6 Subscription

4.6.1 General

The subscription mechanism enables a client of the API to receive changes at the server unsolicitedly. This is typically more efficient than frequent polling by the client.

4.6.2 Subscription management

Subscriptions can be requested (i.e. created), renewed, deleted and listed at this path:

— Path "/subscriptions"

Requesting, renewing and deleting subscriptions through this path depends on the authorization scope. Subscriptions can only be requested, renewed and deleted on the information where the authorized requester was granted sufficient access rights. Requesting, renewing and deleting subscriptions SHALL NOT be permitted by the API if there are no sufficient access rights for any of the requested information or Use Case interfaces.

A subscription can be requested for information of the device model (4.3 - devices, entities, features) or the Use Case model (4.4 - Use Case information, Use Case interfaces). It is possible to subscribe for different kinds of data with a single subscription.

With regard to a given Use Case instance, a client may subscribe to the relevant features directly. Alternatively, the client may subscribe to a particular Use Case instance (via Use Case interface) to receive changes of the related features.

With a POST command on this path a subscription is requested. The POST command takes a request body according to Table 14.

Table 14 — Request body for POST /subscriptions

Element name	Type	M/O/C	Brief explanation
callbackUrl	string (URI)	M	The URI where subscribed data will be sent (see 4.6.3).
devices.	object	O	A subscription to information on one or more devices is requested. The request is about those devices that match ALL of the set child properties of “devices”.
devices. deviceId	string	O	SPINE-IoT device identifier. SHALL be unique within the API.
devices. deviceType	string	O	SPINE device type. See Clause 5.
entities.	object	O	A subscription to information on one or more entities is requested. The request is about those entities that match ALL of the set child properties of “entities”.
entities. deviceId	string	O	SPINE-IoT device identifier. SHALL be unique within the API.
entities. deviceType	string	O	SPINE device type. See Clause 5.
entities. entityId	integer	O	SPINE-IoT entity identifier of a given device. The tuple (deviceId, entityId) SHALL be unique within the API.
entities. entityType	string	O	SPINE entity type. See Clause 5.
features.	object	O	A subscription to one or more features is requested. The request is about those features that match ALL of the set child properties of “features”.
features. deviceId	string	O	SPINE-IoT device identifier. SHALL be unique within the API.
features. deviceType	string	O	SPINE device type. See Clause 5.
features. entityId	integer	O	SPINE-IoT entity identifier of a given device. The tuple (deviceId, entityId) SHALL be unique within the API.
features. entityType	string	O	SPINE entity type. See Clause 5.
features. featureObjType	string	O	SPINE-IoT feature object type.
features. featureType	string	O	SPINE feature type. See Clause 5.

Element name	Type	M/O/C	Brief explanation
usecases.	object	O	A subscription to information on one or more Use Cases is requested. The request is about those Use Case information that match ALL of the set child properties of "usecases".
usecases. deviceId	string	O	SPINE-IoT device identifier. SHALL be unique within the API.
usecases. entityId	integer	O	SPINE-IoT entity identifier of a given device. The tuple (deviceId, entityId) SHALL be unique within the API.
usecases. usecaseName	string	O	The (technical) Use Case discovery name (defined per Use Case).
usecases. usecaseMajorVersion	string	O	The major version of the Use Case. If the detailed version is "1.2.0", the major version is "v1" (without quotation marks). The usecaseMajorVersion is also used as parameter in the path "/usecaseInterfaces/{usecaseName}/{usecaseMajorVersion}".
usecaseInterfaces.	object	O	A subscription to one or more Use Case interfaces is requested. The request is about those Use Case interfaces that match ALL of the set child properties of "usecaseInterfaces".
usecaseInterfaces. deviceId	string	O	SPINE-IoT device identifier. SHALL be unique within the API.
usecaseInterfaces. entityId	integer	O	SPINE-IoT entity identifier of a given device. The tuple (deviceId, entityId) SHALL be unique within the API.
usecaseInterfaces. usecaseName	string	O	The (technical) Use Case discovery name (defined per Use Case).
usecaseInterfaces. usecaseMajorVersion	string	O	The major version of the Use Case. If the detailed version is "1.2.0", the major version is "v1" (without quotation marks). The usecaseMajorVersion is also used as parameter in the path "/usecaseInterfaces/{usecaseName}/{usecaseMajorVersion}".
usecaseInterfaces. actor	string	O	A (technical) Use Case actor name (defined per Use Case).

If the API accepts the subscription request, it responds with a success code and the created subscriptionId according to Table 15. The created subscription MAY expire at the time stated in the property "expires". The duration from the creation of the subscription to its expiration is determined by the server and SHOULD be at least 24h. Subsequently it will be described how a client can renew a subscription.

Table 15 — Response body upon accepted subscription request

Element name	Type	M/O/C	Brief explanation
subscriptionId	string	M	Identifier of the created subscription. SHALL be unique within the API.
expires	date-time	O	The subscription will be released (deleted) by the server at this time. A client can renew this subscription only before this time is reached. If expires is absent, the server SHOULD NOT remove the subscription autonomously.

With a PATCH command on this path, a specific subscription or all of the requester's subscriptions can be renewed. The PATCH command takes a request body according to either Table 16 or Table 17.

Table 16 — Renew all subscriptions

Element name	Type	M/O/C	Brief explanation
renewAll	boolean, only "true" allowed	M	The client requests for the renewal of all of its subscriptions.

Table 17 — Renew a single subscriptionId

Element name	Type	M/O/C	Brief explanation
renewSubscriptionId	string	M	The client requests for the renewal of its subscriptionId as set in renewSubscriptionId.

If the PATCH command is accepted, the server responds with an array of items with properties as shown in Table 15, containing the subscriptionIds where the expiry has been updated. The duration from the renewal of the subscription(s) to their expiration is determined by the server and SHOULD be at least 24h.

With a DELETE command on this path, a specific subscription or all of the requester's subscriptions can be deleted. The DELETE command takes a request body according to Table 18. At least one of the properties SHALL be set.

Table 18 — Request body for the deletion of one or all subscriptions

Element name	Type	M/O/C	Brief explanation
subscriptionId	string	O	Identifier of the subscription that shall be deleted. SHALL be unique within the API.
all	boolean	O	If set to "true", all subscriptions of the requester shall be deleted. In this case, the property subscriptionId SHALL be ignored if present.

With a GET command on this path it is possible to get an overview on all subscriptionId values that are still valid. The response is an array of items with properties as shown in Table 15.

4.6.3 Callbacks

The content of a callback depends on the subscribed information and on the information changes at the server. A server may combine information changes of several subscriptions into a single callback.

A callback contains an array of change descriptions. The following types of change descriptions are defined:

- device change description
- entity change description
- feature change description

— Use Case information change description

Each of these change descriptions expresses whether information was created, replaced (updated) or deleted (see description of type `end pointNotifyChangeType`).

A feature change description can originate from a subscription using the “feature” or “usecaseInterfaces” property in Table 14. This is because a GET command on /features as well as a GET command on a Use Case interface both lead to a response with feature information.

The “device change description” contains information according to a subscription's “devices” property. The possible content of a “device change description” is described in Table 19.

Table 19 — Properties of a “device change description” callback

Element name	Type	M/O/C	Brief explanation
change	<code>end pointNotifyChangeType</code>	M	Indicates whether “device” expresses the creation or replacement or deletion of a “device” information.
device.	object	M	The created/replaced/deleted “device” information. The child properties of “device” are identical to Table 2. The “device” information is uniquely identified by the property “deviceId”.

The “entity change description” contains information according to a subscription's “entities” property. The possible content of a “entity change description” is described in Table 20.

Table 20 — Properties of an “entity change description” callback

Element name	Type	M/O/C	Brief explanation
change	<code>end pointNotifyChangeType</code>	M	Indicates whether “entity” expresses the creation or replacement or deletion of an “entity” information.
entity.	object	M	The created/replaced/deleted “entity” information. The child properties of “entity” are identical to Table 3. The “entity” information is uniquely identified by the properties tuple (“deviceId”, “entityId”).

The “feature change description” contains information according to a subscription's “features” property. The possible content of a “feature change description” is described in Table 21. As a feature can contain a list of `featureObjType` instances, a feature change description expresses changes per `featureObjType` instance.

Table 21 — Properties of a “feature change description” callback

Element name	Type	M/O/C	Brief explanation
change	end pointNotifyChangeType	M	Indicates whether “feature” expresses the creation or replacement or deletion of a featureObjType instance of a feature.
feature.	object	M	A created/replaced/deleted featureObjType instance of a feature. The child properties of “feature” are identical to Table 4. The feature is uniquely identified by the properties tuple (“deviceId”, “entityId”, “featureObjType”, “featureType”). The particular featureObjType instance is uniquely identified by its identifiers.

See Table 5 for supported combinations of featureObjType and featureType.

The “Use Case information change description” contains information according to a subscription's “usecases” property. The possible content of a “Use Case information change description” is described in Table 22.

Table 22 — Properties of a “Use Case information change description” callback

Element name	Type	M/O/C	Brief explanation
change	end pointNotifyChangeType	M	Indicates whether “usecase” expresses the creation or replacement or deletion of a “usecase” information.
usecase.	object	M	The created/replaced/deleted “usecase” information. The child properties of “usecase” are identical to Table 6. The “usecase” information is uniquely identified by the properties tuple (“deviceId”, “entityId”, “usecase. name”, “usecase. usecaseMajorVersion”).

On the type “endpointNotifyChangeType”: This type defines two enumeration values (strings):

— createReplace

This string denotes that the change description contains either a new (created) uniquely identifiable information (device information or entity information, etc.) or the current value of the uniquely identifiable information (i.e. the new value replaces the previous one).

— delete

This string denotes that the change description expresses that the contained uniquely identifiable information has been deleted at the server.

4.7 Requesting feature changes

4.7.1 General

While 4.3.4 focuses on the GET command to get feature information, this subclause describes the POST command for the path “/features”. A POST on /features can be used by a client to create, replace, patch or delete a featureObjType instance of a feature. Notifications of feature changes are described in 4.6.3.

4.7.2 Requesting changes information and instances

A Use Case interface may also provide POST commands. Though these commands also modify one or more features, POST commands on a Use Case interface can be designed in a Use Case specific manner to provide a simpler possibility for the respective modification of features. However, this is described per Use Case interface and will not be considered further within this subclause.

A POST command on the path “/features” contains an array of change requests. Each change request has properties as described in Table 23. As a feature can contain a list of featureObjType instances, a feature change request expresses changes per featureObjType instance.

Table 23 — Properties of a “feature” change request

Element name	Type	M/O/C	Brief explanation
deviceId	string	M	SPINE-IoT device identifier. SHALL be unique within the API.
entityId	integer	M	SPINE-IoT entity identifier of a given device. The tuple (deviceId, entityId) SHALL be unique within the API.
featureObjType	string	M	SPINE-IoT feature object type. This property determines the content type of property “data”.
featureType	string	M	SPINE feature type. See Clause 5. This property determines basic rules that are applied to the given feature.
data.	oneOf featureModifyCreateReplace featureModifyDelete featureModifyPatchUnset	M	To create a new or replace an existing featureObjType instance, use type featureModifyCreateReplace. To delete an existing featureObjType instance, use type featureModifyDelete. To patch an existing featureObjType instance, use type featureModifyPatchUnset.

With the type featureModifyCreateReplace, the POST command contains a featureObjType information that is either new for the server or that is a replacement for an existing featureObjType instance (i.e. the new value replaces the previous one). The type is shown in Table 24.

Table 24 — Properties of type featureModifyCreateReplace

Element name	Type	M/O/C	Brief explanation
data. createReplace	oneOf ... (see text)	M	The content type of “data. createReplace” SHALL correspond exactly to the feature object type denoted by the property “featureObjType”. The feature definitions are described in EN 50631-3-1 (see YAML models for SPINE-IoT). The content expresses either a new featureObjType instance or the replacement of an existing featureObjType instance. The particular featureObjType instance of the given feature is uniquely identified by its identifiers.

With the type featureModifyDelete, the POST command contains information on a featureObjType instance that is to be deleted at the server. The type is shown in Table 25.

Table 25 — Properties of type featureModifyDelete

Element name	Type	M/O/C	Brief explanation
data. delete	oneOf ... (see text)	M	The content type of “data. delete” SHALL correspond exactly to the feature object type denoted by the property “featureObjType”. The feature definitions are described in EN 50631-3-1 (see YAML models for SPINE-IoT). The contained identifiers determine which featureObjType instance is to be deleted at the server. Other properties of “data. delete” SHALL NOT be considered for the deletion of the instance.

With the type featureModifyPatchUnset, the POST command contains a featureObjType information that expresses the adjustment of one or more selected properties of the respective featureObjType instance. This means other properties of a featureObjType instance (those that are not “selected”) remain unchanged by the POST command. However, a server may need to adjust further properties by itself upon the execution of the POST command (this depends on the server's application owning the featureObjType instance, which is beyond the scope of this subclause). The type permits to alter (patch) the value of properties as well as the deletion (unset) of properties. The type is shown in Table 26.

Table 26 — Properties of type featureModifyPatchUnset

Element name	Type	M/O/C	Brief explanation
data. patch	oneOf ... (see text)	M	The content type of “data. patch” SHALL correspond exactly to the feature object type denoted by the property “featureObjType”. The feature definitions are described in EN 50631-3-1 (see YAML models for SPINE-IoT). The contained identifiers determine which featureObjType instance is to be adjusted at the server. The other properties within “data. patch” are set accordingly by the server in its featureObjType instance.
data. unset	oneOf ... (see text)	O	The content type of “data. unset” SHALL correspond exactly to the feature object type denoted by the property “featureObjType”. The feature definitions are described in EN 50631-3-1 (see YAML models for SPINE-IoT). The contained identifiers determine which featureObjType instance is to be adjusted at the server. The other properties within “data. unset” are deleted by the server in its featureObjType instance, regardless of the values of these properties in “data. unset”.

The following rules apply to all kinds of change requests:

- A server SHALL silently discard unknown properties of the received featureObjType information.

NOTE This preserves compatibility with API versions that just introduce new properties of a known featureObjType.

- For properties that are not discarded according to the previous rule, a server SHALL reject the whole POST command if any of the properties of the received featureObjType information cannot be applied by the server.

With the exception of discarded properties, this means a POST command shall only be accepted completely or rejected completely. The server shall not apply a partial application of the POST command.

5 SPINE Protocol

5.1 General

The SPINE Protocol serves as neutral message framework to model some typical mechanisms of common communication protocols. This permits to model interactions between SPINE devices and local energy management system.

The next subclauses discuss a general “SPINE device architecture” including the concept of addressing, a general neutral message format (“SPINE datagram”), and so-called “communication modes”. Afterwards, some further mechanisms (like detailed discovery, binding, subscription, etc.) are discussed. This can be seen as “functional commissioning” between SPINE devices, i.e. the management of useful relations between functionalities of SPINE devices.

5.2 Architecture overview

5.2.1 General rules

In theory, the SPINE data model permits arbitrary combinations of SPINE entities, SPINE features, SPINE classes etc. However, the definition of interoperable binding (see 5.5.4) and subscription (see 5.5.5) mechanisms requires imposing some restrictions, i.e. the definition of an architecture. Thus, the following rules apply:

1. A device consists of entities. An entity consists of (sub-)entities or features. For each address level (device level, entity level, feature level), the SPINE protocol defines an addressing scheme. Remark: This is explained in more detail in 5.2.3.
2. Each device SHALL implement a so-called primary NodeManagement instance with information on itself. The device SHALL offer this information on its entity 0 (entity of the device with the entity address = 0) at feature 0 (feature of the entity 0 with the feature address = 0). This information SHOULD contain information about all entities of the device and all features of these entities. The so-called “entityType” (see 5.5.2) of entity 0 SHALL be “DeviceInformation”. The so-called “featureType” (see 5.5.2) of feature 0 of entity 0 SHALL be “NodeManagement”.
3. This document does not consider any non-primary NodeManagement instance (i.e. NodeManagement instances on different addresses than entity 0 at feature 0 are not considered).
4. Each NodeManagement instance is a feature in general (see above). Thus, NodeManagement instances include proper information on itself or other NodeManagement instances as well, according to the rules given above.
5. On each feature there SHALL be at maximum one class implemented with regard to the features primary functionality. I.e. it is NOT permissive to offer more than one class on a single feature (see also 5.3.3).
6. A feature on a device is assigned a “functional role”. This role is EITHER “server” OR “client” OR “special”. Please note a “functional role” is independent from any “connection role” (i.e. a role typically used in communications technologies like TCP).
7. The functional role “server” is used if the device is the “owner” of data of the corresponding feature (see also 5.3.3.3). I.e. the device can notify changes or send this data as reply upon request. It may also accept “write” operations to perform a data change. In most cases, the feature role as “server” also includes the capability of a device to operate autonomously, i.e. to execute its server feature tasks even if no feature with role “client” sends data to control the server's feature tasks.

8. The functional role “client” is used to receive information provided by a “server” and to use the server's features accordingly (example: configure the server features using “write” operations, provided this is offered by the server feature).
9. The functional role “special” is reserved for specific features. It SHALL ONLY be used for features explicitly mentioned in official SPINE specifications. In this document the primary NodeManagement instance SHALL have the role “special”.
10. The concepts for binding and subscription respect the role assignment. Details are given in the corresponding sections.

NOTE The rules above only describe the architecture. Whether all or just reduced information of a device's NodeManagement instances is shared with another device and whether the communication between the features of two devices is restricted (or even blocked) or not is discussed separately (especially using the “trust level” concept).

5.2.2 Common data types

5.2.2.1 General

The majority of data types use data types defined by W3C [5]. In this document these data types are identified by the namespace prefix “xs”: (“xs:boolean”, e.g.).

This document defines also some own data types which are based upon W3C data types. In this document these data types are mentioned without namespace prefix. Details on their definition can be found in the following.

5.2.2.2 ElementTagType

Used for elements that have no content (empty tag). A complexType, containing no elements.

If used as type for an element, the element SHALL have no further content in this SPINE edition.

5.2.2.3 LabelType

A default type used for all “label” elements. A simpleType (restricted string).

The string-length SHOULD NOT be longer than 256 characters. If it is longer, the sender SHALL consider the possibility that the receiver will shorten the string to 256 characters.

5.2.2.4 DescriptionType

A default type used for all “description” elements. A simpleType (restricted string).

The string-length SHOULD NOT be longer than 4 096 characters. If it is longer, the sender SHALL consider the possibility that the receiver will shorten the string to 4 096 characters.

5.2.2.5 SpecificationVersionType

The SPINE specification version.

A simpleType (restricted string), with the pattern

`[1-9][0-9]*\.[0-9]+\.[0-9]+`

allowing version strings like “1.0.0”, “2.5.3”, etc. The maximum string length is limited to 128 characters.

5.2.2.6 EnumExtendType

Used for extending enumeration with vendor specific ones. Some enumerations may be extended by the vendors with own values. A simpleType (restricted string), with the pattern `_(i:[1-9][0-9]*|n:[a-zA-Z0-9-]+)_\p{Cc}\p{Cf}\p{Z}+`

The maximum string length is limited to 256 characters.

5.2.2.7 RoleType

If a role has to be stated, this type is used. A simpleType (restricted string), containing the enumeration values client, server or special.

5.2.2.8 FeatureGroupType

Use for connecting related features. A simpleType (restricted string), with the pattern

`(#[1-9][0-9]*)*`

allowing feature group strings like “#1”, “#1#5”, etc. The maximum string length is limited to 128 characters.

5.2.2.9 DeviceTypeType

Union of:

- DeviceTypeEnumType (see 5.2.2.10)
- EnumExtendType (see 5.2.2.6)

5.2.2.10 DeviceTypeEnumType

The type of the device.

5.2.2.11 EntityTypeType

Union of:

- EntityTypeEnumType (see 5.2.2.12)
- EnumExtendType (see 5.2.2.6)

5.2.2.12 EntityTypeEnumType

The type of the entity.

5.2.2.13 FeatureTypeType

Union of:

- FeatureTypeEnumType (see 5.2.2.14)
- EnumExtendType (see 5.2.2.6)

5.2.2.14 FeatureTypeEnumType

The type of the feature.

5.2.2.15 FunctionType

Union of:

- FunctionEnumType (see 5.2.2.16)
- EnumExtendType (see 5.2.2.6)

5.2.2.16 FunctionEnumType

A list of all SPINE functions.

5.2.2.17 PossibleOperationsClassifierType

If the element partial is used, filtering the further content is permitted. Otherwise only full operations (no filtering) are allowed. A complexType, containing the element partial (type: ElementTagType, see 5.2.2.2).

5.2.2.18 PossibleOperationsReadType

If read operations are permitted, this element is set. Elements from the base type can be used. A complexType with the base type PossibleOperationsClassifierType (see 5.2.2.17).

5.2.2.19 PossibleOperationsWriteType

If write operations are permitted, this element is set. Elements from the base type can be used. A complexType with the base type PossibleOperationsClassifierType (see 5.2.2.17).

5.2.2.20 PossibleOperationsType

Defines which operations on a function are allowed, as shown in Figure 6. A complexType, containing the elements read (type: PossibleOperationsReadType, see 5.2.2.18) and write (type: PossibleOperationsWriteType, see 5.2.2.19)

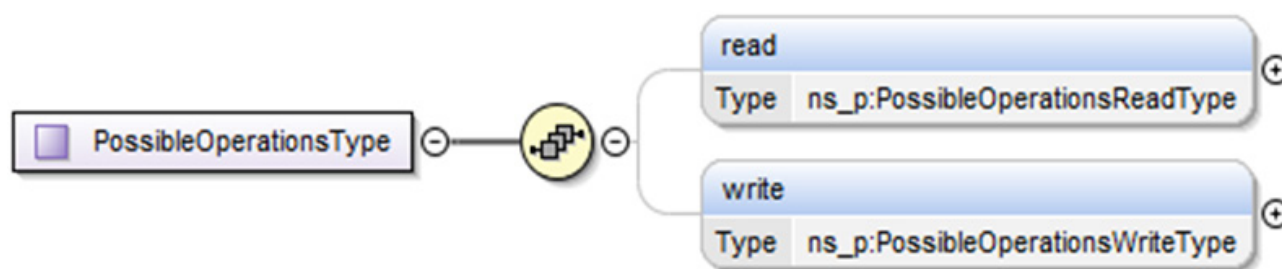


Figure 6 — PossibleOperationsType

5.2.2.21 MaxResponseDelayType

Used to represent the maxResponseDelay. A simpleType (xs:duration).

5.2.2.22 AbsoluteOrRelativeTimeType

The commonly used type for time information in the SPINE data model. Allows the modelling of absolute (xs:dateTime) or relative (xs:duration) times. A union with the two members (allowed types) xs:duration and xs:dateTime.

xs:dateTime is used to model absolute times.

xs:duration is used to model relative times. If not stated otherwise a duration of 0 s (e.g. "PT0S", or "PT0H", etc.) equals "now", while negative durations denote times in the past and positive values denote times in the future relative to "now".

5.2.2.23 AddressDeviceType

Used to address a (physical) device. A simpleType (restricted string), containing the device address part, with the pattern

$d: _ (i: [1-9] [0-9]^* | n: [a-zA-Z0-9-]^+) _ [^\backslash p\{Cc\} \backslash p\{Cf\} \backslash p\{Z\}]^+$

5.2.2.24 AddressEntityType

Used to address a logical device. A simpleType (unsignedInt), containing an entity address part.

5.2.2.25 AddressFeatureType

Used to address a specific feature. A simpleType (unsignedInt), containing the feature address part.

5.2.2.26 DeviceAddressType

Address type including only the device part, as shown in Figure 7. A complexType with the element device (type: AddressDeviceType, see 5.2.2.23).

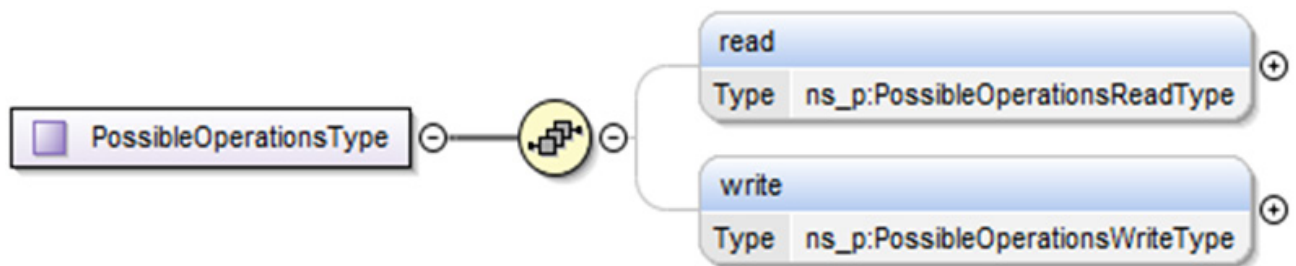


Figure 7 — DeviceAddressType

5.2.2.27 EntityAddressType

Address type including the device part and a list of entities, as shown in Figure 8. A complexType with the base type DeviceAddressType (see 5.2.2.26) and a further list of element entity (type: AddressEntityType, see 5.2.2.24).

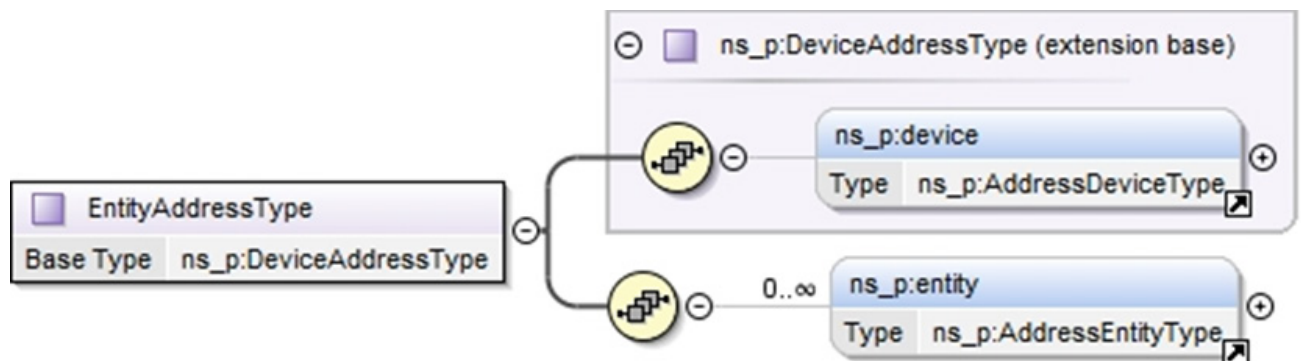


Figure 8 — EntityAddressType

5.2.2.28 FeatureAddressType

Address type including the device part, a list of entities and the feature part, as shown in Figure 9. A complexType with the base type EntityAddressType (see 5.2.2.27) and a further element feature (type: AddressFeatureType, see 5.2.2.25).

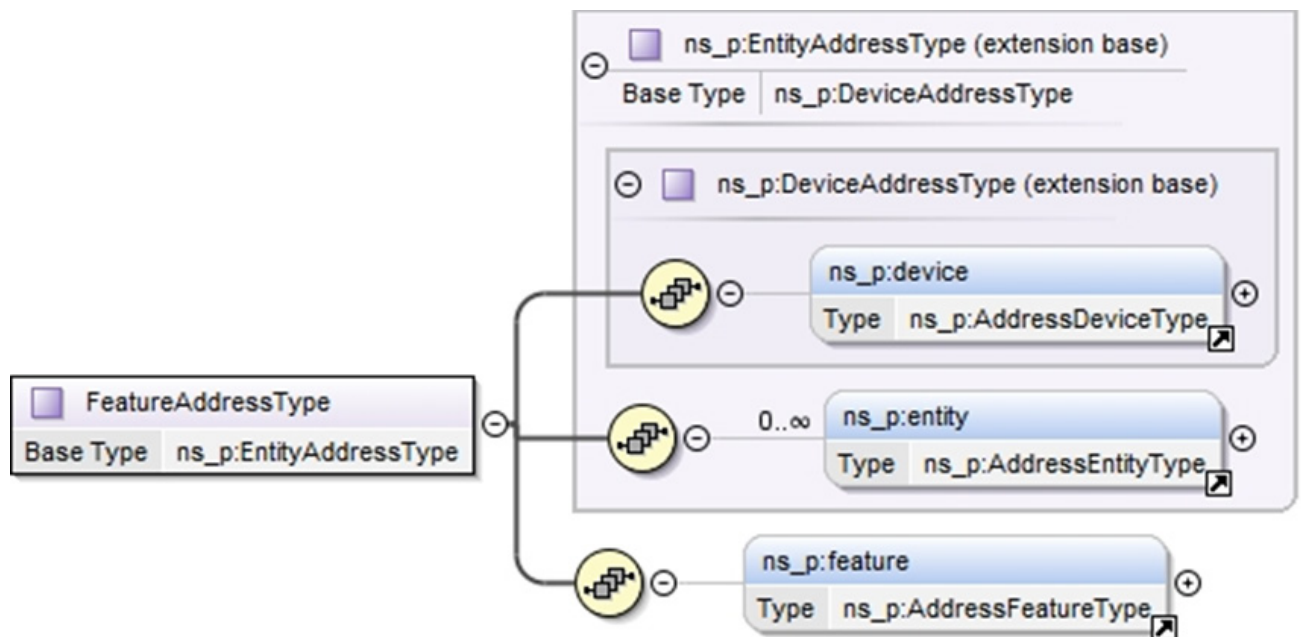


Figure 9 — FeatureAddressType

5.2.3 Address level details

The concept of the miscellaneous address levels as described in 5.2.1.

The following example defines a device with name “someDevice” with three top-level entities (entities directly below the device “someDevice”) with entity addresses 0, 1, 4 (remark: “entity 0” describes a SPINE entity with the entity address = 0, feature 0 describes a SPINE feature with the feature address = 0):

```
"someDevice"
  +- entity 0          (child of "someDevice")
  |   +- feature 0
  +- entity 1
  |   +- entity 4 (child of "someDevice"/entity 1)
  |   |   +- feature 7 (*1)
  |   +- entity 5 (child of "someDevice"/entity 1)
  |   |   +- feature 1 (*2)
  |   |   +- feature 7 (*2)
  |   +- feature 1 (child of "someDevice"/entity 1)
  |   +- feature 4 (child of "someDevice"/entity 1)
  +- entity 4          (child of "someDevice")
  |   +- feature 1 (child of "someDevice"/entity 4)
(*1): child of "someDevice"/entity 1/entity 4
(*2): child of "someDevice"/entity 1/entity 5
```

In this example, the top-level entity 1 contains two features (1, 4), but also two other (sub-)entities. This shows that entities can be nested.

Starting at the top, the full address paths are (note that this is an example to describe the SPINE address level concept and that the following lines DO NOT state valid SPINE addresses; the correct usage of SPINE addresses is described in 5.3):

1. device “someDevice”
2. device “someDevice” / entity 0
3. device “someDevice” / entity 0 / feature 0
4. device “someDevice” / entity 1
5. device “someDevice” / entity 1 / entity 4
6. device “someDevice” / entity 1 / entity 4 / feature 7
7. device “someDevice” / entity 1 / entity 5
8. device “someDevice” / entity 1 / entity 5 / feature 1
9. device “someDevice” / entity 1 / entity 5 / feature 7
10. device “someDevice” / entity 1 / feature 1
11. device “someDevice” / entity 1 / feature 4
12. device “someDevice” / entity 4
13. device “someDevice” / entity 4 / feature 1

Only the full address path is unique. I.e. feature 7 of { device “someDevice” / entity 1 / entity 4 } differs to feature 7 of { device “someDevice” / entity 1 / entity 5 }. Similarly, entity 4 of { device “someDevice” / entity 1 } differs to entity 4 of { device “someDevice” }.

It shall now be explained how this is modelled in the SPINE XSDs and XMLs. In the XSDs an element “entity” is often defined with attributes

```
minOccurs="0" maxOccurs="unbounded"
```

The upper bound of this cardinality means an “entity” tag can occur arbitrarily often at the given position in an XML (representing arbitrary depth of nested entities). However, a device that complies with this and previous versions of the SPINE specification only can silently discard messages where an entity list comprises more than 15 “entity” items. This means an implementation SHOULD avoid to implement any of its entities with more than 15 “entity” items as it is likely that a communication partner will ignore such entities. The lower bound “0” of this cardinality means there may be no “entity” element at all. Please note that the architecture itself always requires the presence of at least one entity. But certain message definitions that contain address elements with this generic address type may well permit the absence of “entity” elements for specific purposes (among others, the deletion of all so-called “bindings” between two devices omits all “entity” elements in a specific address element).

The i-th occurrence of “entity” within an address instance corresponds to the i-th entity level (depth). The entity level “i+1” is a child of entity level “i”. I.e. an XML part with full address path for { device “someDevice” / entity 1 / entity 4 / feature 7 } is represented as follows:

```
<device>someDevice</device>
<entity>1</entity>
<entity>4</entity>
<feature>7</feature>
```

Within the message description tables, the possibility of multiple “entity” tags (representing nested entities) is specified as follows:

... entity (list)

As already explained above, multiple entity tags in an address instance always describe “nested entities”, i.e. entities as child elements of other entities. Such multiple entity tags DO NOT describe “parallel” entities (entities on the same hierarchical level). One address instance can only describe one single address path and is not used to describe several parallel address paths of a device.

With regard to the so-called “restricted function exchange” (see 5.3.3.4) it needs to be emphasized that a SPINE address as information unit shall always be seen as an indivisible information unit that needs to be transmitted completely (with the exception that the “device” part might be omitted in some cases). This means that a SPINE address with a list of entity tags does neither match the definition of “identifiable list entries” nor the definition of “non-identifiable list entries” of 5.3.3.4.1.

NOTE A SPINE address might also be used for other purposes than an information unit – see especially 5.3.3.4.7.3 for its use as so-called selectors.

5.3 SPINE Datagram

5.3.1 Introduction

5.3.1.1 General information

The ISO-OSI layer model defines an application layer, among others. Interaction with another application is considered an end-to-end connection. With regard to the SPINE data model the “functions” could be exchanged on this level.

In order to model the exchange between two end points dynamically the SPINE data model defines an element “datagram” consisting of “header” and “payload”, as described in Figure 10.

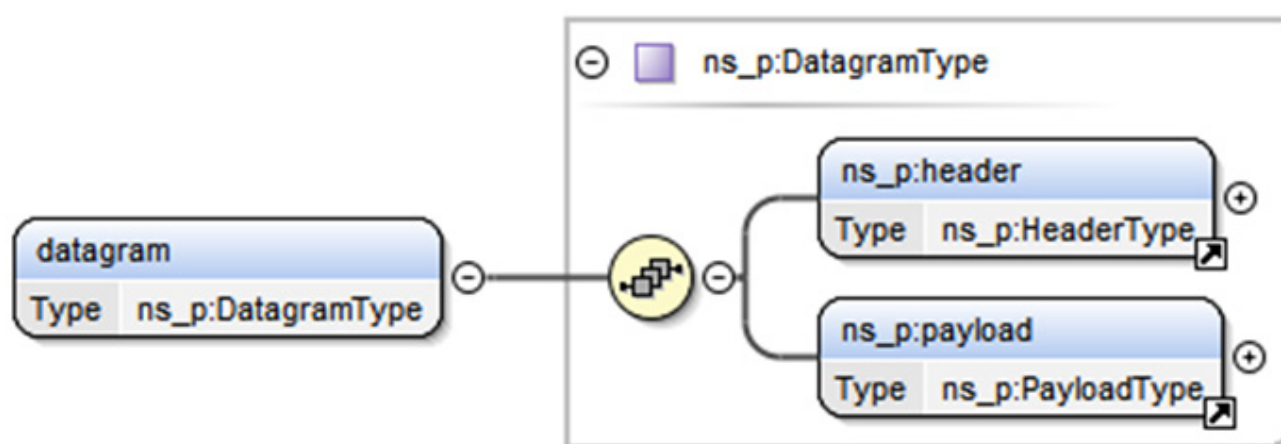


Figure 10 — SPINE datagram

5.3.1.2 Structure

The structure of the SPINE datagram SHALL be set as shown in Table 27.

Table 27 — Structure of the SPINE datagram

Element name	M/O/NV/C	Brief explanation
datagram	M	The root element of the SPINE datagram SHALL always be present.
datagram. header	M	The header element SHALL be present. For sub-elements of the header see 5.3.2.7.
datagram. payload	M	The payload element SHALL be present. For sub-elements of the payload see 5.3.3.2.

The notation “a.b” (for example “datagram.header”) is used here to show the hierarchical structure of a SPINE Datagram. It is used to define that “b” is a child element of “a”.

5.3.2 Header

5.3.2.1 General information

The SPINE header contains information on the version of the applied data model, addresses of end points, etc, as described in Figure 11. This technical solution defines the SPINE Protocol Specification version 1.3.0.

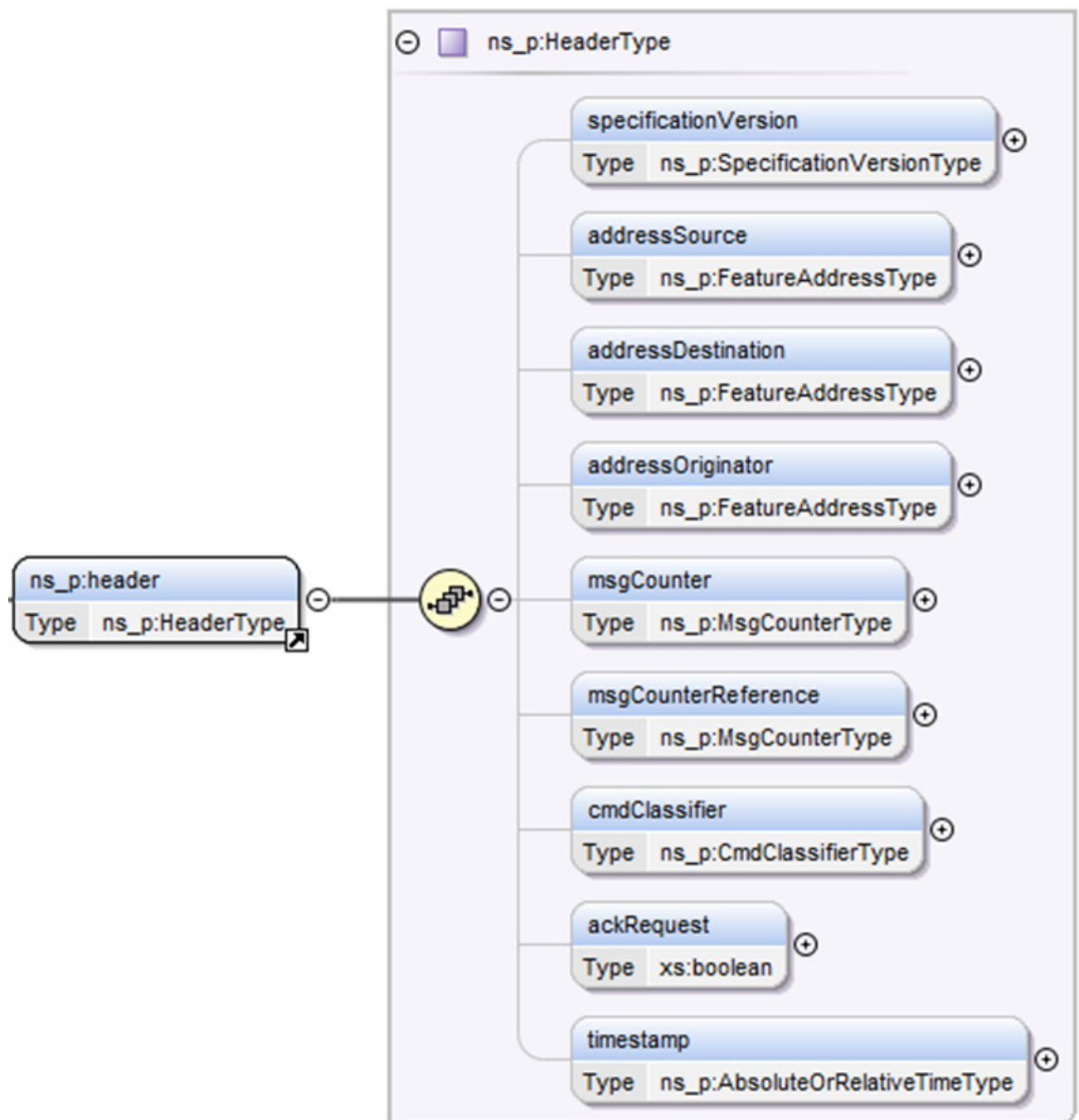


Figure 11 — SPINE header

5.3.2.2 Address information

5.3.2.2.1 addressSource and addressDestination

The elements `addressSource` and `addressDestination` are defined with their child elements and purpose in Table 29. `addressSource` corresponds to the unique address path (see 5.2.3) of a feature where a SPINE message was created. `addressDestination` corresponds to the unique address path of the receiving feature. Subsequently some rules on the child elements of `addressSource` and `addressDestination` are given.

If a device creates a message it SHALL set its entity and feature address parts into `addressSource`. The recipient's entity and feature address parts SHALL be set in `addressDestination`.

The "device" address parts of `addressSource` and `addressDestination` may be omitted in some cases. When to set the "device" address parts is defined subsequently. For this, we consider a communication between the

SPINE devices “A” and “B”. The “common rules” (see below) apply in any case. Furthermore, additional rules apply dependent on the communication mode:

Common rules on the use of “device” address elements:

In general, the use of “addressSource. device” and “addressDestination. device” REQUIRES to ensure the uniqueness of all “device” values. I.e. device “A”’s value of “device” SHALL be different to device “B”’s value of “device”. This is also in accordance with 5.5.2.2.5.2.

We assume a message is transmitted from device “A” to device “B”:

1. If “addressSource. device” is present and not empty it SHALL be identical to device “A”’s own value of “device”. The message SHALL be considered illegal if the value differs.
2. If “addressDestination. device” is present and not empty it SHALL be identical to device “B”’s own value of “device”. The message SHALL be considered illegal if the value differs.

Additional rules in case of “simple communication mode”:

The following rules apply in case of a “simple communication mode” (see 5.4.2). In this mode, two devices are considered being directly connected to each other. We assume a message is transmitted from device “A” to device “B”:

Within the header the “device” elements of addressSource and addressDestination SHOULD be set to the correct value. The absence of “device” in “addressSource” SHALL be treated as if “device” in “addressSource” was set to device “A”’s own value of “device” (i.e. the “device” address part of device “A”). The absence of “device” in “addressDestination” SHALL be treated as if “device” in “addressDestination” was set to device “B”’s own value of “device” (i.e. the “device” address part of device “B”).

Additional rules in case of “enhanced communication mode”:

The following rules apply in case of an “enhanced communication mode” (see 5.4.3). In this mode, both “addressSource. device” and “addressDestination. device” SHALL always be set in the header.

5.3.2.2.2 addressOriginator

As explained in Table 29 the element “addressOriginator” can be used to model information of a “forwarded” XML and permits preserving the address of the original submitter. The use of addressOriginator is optional. However, a brief example shall demonstrate a typical use of this element:

We assume device “A” created a message “X” and submitted it to device “B”. We also assume device “B” serves as some kind of “data warehouse” for all messages it received from miscellaneous devices. With its “data warehouse” service it can provide an overview of messages from all its connected devices. We also assume these messages are available on one or more features of device “B”. If a device “C” reads on such a feature of device “B”, device “B” shall set its own feature address into “addressSource” of its response to device “C”. In order to keep the information where the functional content of the message originally stemmed from (device “A”) the element “addressOriginator” needs to be set properly.

If “addressOriginator” is used as described above, the elements “addressSource. device”, “addressDestination. device” and “addressOriginator. device” SHALL be set.

5.3.2.3 Message counter

5.3.2.3.1 General

A message counter (msgCounter, msgCounterReference) serves for the identification of a message. This is especially important if replies are delivered in a different order than the corresponding requests.

5.3.2.3.2 msgCounter

If a device creates a message it SHALL assign msgCounter a value that does not conflict with any other of its recently created messages (i.e. it SHALL be a virtually unique value). In general, the value SHALL NOT collide with any of the device’s previously created messages where the device still expects proper responses. The msgCounter value SHALL be ascending and restart from 0 once the largest possible number ($2^{64}-1$) was

used. The msgCounter values MAY skip some numbers (e.g. after a message “X” with msgCounter “10” a message “Y” with msgCounter “15” is sent).

If a SPINE device “A” receives a message “X” from SPINE device “B” with a msgCounter less or equal than the last msgCounter received from device “B”, “A” SHALL process the message “X” as usual. Afterwards, device “A” SHALL use the unexpectedly low msgCounter value as the last msgCounter received from device “B”. If device “A” receives a message with unexpectedly low msgCounter value from device “B”, it MAY report this to the user in order to report that the communication partner may have a problem or unexpected condition (e.g. factory reset); however, such a report is only useful if the underlying communications technology preserves the order of messages.

Implementation advice: A best practice to keep msgCounter values unique even in case of power failures is as follows: A device keeps a “stored msgCounter value” in a non-volatile memory. When a device is turned on, it first increases the “stored msgCounter value” by 1 000 and uses the result for its next message. From then on, every 1 000th created message the device copies the msgCounter value into the “stored msgCounter value”.

Please note: A device MAY assign its messages in the msgCounter field values that are unique across all its communication partners. But this kind of uniqueness is not required by this document. Rather, this kind of uniqueness may be easier for implementation in embedded devices whereas web servers may scale better with values that are communication partner specific.

5.3.2.3.3 msgCounterReference

If the created message serves as reply or comparable reaction to a previously received message, the device SHALL take the value of msgCounter from the received message and set it into msgCounterReference of its own message.

The element msgCounterReference SHALL NOT be set if the sent-out message does not relate to a received message.

5.3.2.4 Message classifiers

The element cmdClassifier conveys information on the kind of operation associated with the given message “M”.

This subclause also uses the terms “dedicated reply”, “dedicated data”, or “dedicated response”. The SPINE specification defines a number of (sub-)classes with (SPINE) functions. A device “A” may be the “owner” of a specific function instance (i.e. it has the role “server” at the function’s feature address). Another device “B” may request to get a (full or specifically curtailed) copy of this (SPINE) function. A response is called “dedicated” if this request can be fulfilled, i.e. the reply conveys indeed a proper copy of this (SPINE) function.

Table 28 shows permitted values for cmdClassifier and the relation to the kind of message “M”. The receipt of a message “M” may lead to the return of a related “acknowledgement message” (see 5.3.2.6), denoted as “N”. The scope of “N” is also shown in Table 28.

Table 28 — cmdClassifier values and kind of messages for a message “M” and the scope of related acknowledgement messages

cmdClassifier message “M”	of	Kind of message “M”	Scope of related acknowledgement message “N” (see 5.3.2.6.2)
read		Initial	Application error
write		Initial	Application success or error
call		Initial	Application success or error
reply		Response	Transmission success or error
notify		Response	Transmission success or error
result		Acknowledgement	Not applicable

We assume device “A” sends an “initial message” to device “B” (address level details like “feature” are just left out to simplify the explanation). Dependent on the received “initial message” device “B” creates a “response message” for device “A”. I.e. a “response message” is always related to an “initial message”.

Furthermore, “response messages” are only used in case the received “initial message” was processed successfully by the recipient.

An “acknowledgement message” can be used to indicate whether an “initial message” or “response message” was processed or transmitted successfully or not (see 5.3.2.6.2).

The different values of cmdClassifier are used for the following operations:

read

This denotes a “read operation” from the sender (addressSource) to the recipient (addressDestination). The recipient is considered the owner of the proper information (i.e. function). The recipient SHALL respond with a dedicated “reply” if the read operation is valid and can be processed regularly, i.e. without any failure. Otherwise the recipient SHALL respond with an “acknowledgement message” (see 5.3.2.6.2) where “errorNumber” is set to a different value than “0”. Please note that the obligation to create a “reply” or “error indication” is independent from the value of the element “ackRequest” of the header. More precisely, the acknowledgement request (see 5.3.2.6.1) of a received “read operation” SHALL NOT be evaluated.

reply

This denotes a “reply operation” from the owner of the (assumed) proper information (addressSource) to the recipient (addressDestination). A reply SHALL be created according to the rules given for cmdClassifier value “read”. It SHALL NOT be used in any other case. The recipient of a “reply operation” SHALL evaluate the acknowledgement request of the message according to 5.3.2.6.1. Please note that a proper acknowledgement message just denotes a transmission success or error of the “reply operation”.

notify

This denotes a notification (“notify operation”) from the owner of the proper information (addressSource) to the recipient (addressDestination). In contrast to a dedicated “reply” upon a “read” message, a “notify” message is created autonomously by the owner of the proper information, i.e. without the need of a received “read” message. A typical example for the creation of a “notify” message is the notification of a value change in a feature that is subscribed by the recipient. The recipient of a “notify operation” SHALL evaluate the acknowledgement request of the message according to 5.3.2.6.1. Please note that a proper acknowledgement message just denotes a transmission success or error of the “notify operation”.

write

This denotes a replacement or modification instruction (“write operation”, “full” or “restricted”) from the sender (addressSource) to the recipient (addressDestination). The recipient is considered the owner of the information (SPINE class function) that shall be replaced or modified according to the instruction. The recipient of a “write operation” SHALL evaluate the acknowledgement request of the message according to 5.3.2.6.1.

call

This denotes an instruction (“call operation”) from the sender (addressSource) to the recipient (addressDestination) to trigger a specific action at the recipient. A “call operation” can be used to exchange information where the “ownership concept” of the other classifiers usually does not apply. The recipient of a “call operation” SHALL evaluate the acknowledgement request of the message according to 5.3.2.6.1.

result

This message classifier is used for so-called “application acknowledgement messages” (also called “result messages”; see 5.3.2.6.1) to indicate the general success or error with regard to a transmitted message. The recipient of a “result message” SHALL NOT evaluate the acknowledgement request of the received “result message”. A “result message” SHALL NOT be created as any kind of response to a received “result message”.

5.3.2.5 Acknowledgement concept

5.3.2.5.1 Acknowledgement request

A message “M” denotes the kind of its “acknowledgement request” with the element “datagram. header. ackRequest” (see Table 29): The value “true” indicates “acknowledgement message is required” whereas the value “false” indicates “acknowledgement message is NOT required”.

The recipient of the message “M” SHALL submit an acknowledgement message (also referred to as “result message” as described below) (see 5.3.2.6.2) if all of the following conditions are fulfilled:

1. The received message “M” belongs to an operation that **REQUIRES** the evaluation of the acknowledgement request according to 5.3.2.5.
2. The received message “M” does not belong to an operation that forbids the evaluation of the acknowledgement request according to 5.3.2.5.
3. The received message “M” does not belong to an operation that forbids the creation of an acknowledgement message according to 5.3.2.5.
4. The received message “M” indicates “acknowledgement message is required”, i.e. the value of “ackRequest” is set to “true”.

5.3.2.5.2 Acknowledgement message

An acknowledgement message “N” is related to a received message “M” and indicates whether the received message “M” could be processed or received successfully (positive acknowledgement) or not (negative acknowledgement, i.e. error indication). More precisely, the following scopes of “N” can be assigned:

1. Application success:

Positive acknowledgement: The recipient of “M” received and evaluated and processed “M” successfully.

2. Application error:

Negative acknowledgement: The recipient of “M” encountered a problem with “M” or an error occurred with the transmission of “M” to the recipient of “M”.

3. Transmission success:

Positive acknowledgement: The recipient of “M” confirms it received “M”.

4. Transmission error:

Negative acknowledgement: An error occurred with the transmission of “M” to the recipient of “M”.

Which scope applies is defined in 5.3.2.5.

As the result classifier is used for acknowledgement messages, such messages are also called “result messages”.

A “result message” is composed as follows:

Element “cmdClassifier” SHALL be set to “result”. The remaining header elements are set as if the message is a regular reply to the related received message. Furthermore, the element “payload” SHALL contain a “resultData” function. The element “errorNumber” of “resultData” expresses the kind of the message and SHALL be present and set as follows:

1. A value of “0” SHALL be used for “positive acknowledgement”.
2. Every other value denotes a “negative acknowledgement”.

In addition, the element “description” of “resultData” MAY be present and filled with a readable information.

In case a “result message” cannot be sent within time according to “defaultMaxResponseDelay” or “maxResponseDelay” described in 5.3.2.6.3, refer to 5.3.2.6.3.

Please note: This subclause does not specify the circumstances when an acknowledgement message is to be sent or not. For this, see 5.3.2.5 and 5.3.2.6.1.

5.3.2.5.3 Delayed application response

For each feature server a device needs some time to generate a proper response (which could be a message with “reply” classifier or “result” classifier) upon a received request. To have an interoperable concept for

devices to know how long to wait for a response, a timeout mechanism with “maximum response delay” is used.

The duration specified by “maximum response delay” only relates to the immediate access to the communication interface of a device. This means that latencies from communication channels are not covered by “maximum response delay”.

NOTE Each communication channel has some latency. This latency depends on the kind of the communications technology itself as well as on the environment (especially wireless technologies often suffer from poor coverage or too many participants sharing the available bandwidth). As specified above, this kind of latency is independent from the “maximum response delay”.

There are two timeout levels to derive the “maximum response delay” for a given server feature:

1. By default, the “maximum response delay” is equal to defaultMaxResponseDelay. The defaultMaxResponseDelay SHALL be 10 s.
2. If the maxResponseDelay element within the feature description of the detailed discovery (see 5.5.2.3) is set AND the value of the maxResponseDelay element is larger than zero seconds, the value of the maxResponseDelay element SHALL be applied for this feature as “maximum response delay” (i.e. instead of defaultMaxResponseDelay).

Recommendations on the value of maxResponseDelay:

1. If a feature server will usually or frequently need more time than defaultMaxResponseDelay to generate proper replies, it SHOULD set the maxResponseDelay element in the feature description of the detailed discovery (see 5.5.2.3) to a duration that the feature server can almost always fall short of for the generation of proper replies.
2. If a feature server will almost always need less time than defaultMaxResponseDelay to generate proper replies, it MAY set the maxResponseDelay element in the feature description of the detailed discovery (see 5.5.2.3) to a duration that the feature server can almost always fall short of for the generation of proper replies.

Rules on the use of “maximum response delay”:

1. An implementation that complies with this or a subsequent version of the specification SHALL be able to handle response delays of at least defaultMaxResponseDelay.
2. A feature client MAY use “maximum response delay” for the detection of a response-timeout even if “maximum response delay” is shorter than defaultMaxResponseDelay.
3. An implementation SHOULD consider the communications technology specific latency carefully before it begins with the detection of a timeout to an expected response.

5.3.2.6 Time information in “timestamp”

The header’s (optional) element “timestamp” uses the type AbsoluteOrRelativeTimeType. In case of absolute times the UTC zone shall be applied. A valid example is “2016-04-28T19:43:14.3Z” (in contrast to “2016-04-28T17:43:14.3-02:00” or even the bare local time “2016-04-28T17:43:14.3”).

The application of relative times (i.e. according to the type “xs:duration”) is of limited use. It is usually only useful for devices that “collect” their data/messages over a certain period and send them only at specific times (e.g. battery powered devices; these often also have no real time clock/UTC setup). In this case, the (negative) relative time is relative to “now” and indicates when the message was created in the past.

5.3.2.7 Structure

The structure of the SPINE header SHALL be set as shown below.

Table 29 — Structure of the SPINE header

Element name	Type	M/O/NV/C	Brief explanation
datagram. header		M	The header element SHALL be present.
datagram. header. specificationVersion	SpecificationVersion Type (see 5.2.2)	M	The version of the SPINE data model applicable to the XML. SHALL be present.
datagram. header. addressSource	FeatureAddressType (see 5.2.2)	M	The address of the submitter of this XML. SHALL be present.
datagram. header. addressSource. device	AddressDeviceType; see “Device address” in 5.5.2.2.5.2	O	Device part of the source address. SHOULD be present in simple communication mode, SHALL be present in all other communication modes. If the element is present, it SHALL not be empty. See 5.3.2.2.1 for details.
datagram. header. addressSource. entity (list)	AddressEntityType (see 5.2.2)	M	Entity part(s) of the source address. SHALL be present. See also 5.2.3, especially concerning restrictions on the supported depth.
datagram. header. addressSource. feature	AddressFeatureType (see 5.2.2)	M	Feature part of the source address. SHALL be present.
datagram. header. addressDestination	FeatureAddressType	M	The address of the receiver of this XML. SHALL be present.
datagram. header. addressDestination. device	AddressDeviceType; see “Device address” in 5.5.2.2.5.2	O	Device part of the destination address. SHOULD be present in simple communication mode, SHALL be present in all other communication modes. If the element is present, it SHALL not be empty. See 5.3.2.2.1 for details.
datagram. header. addressDestination. entity (list)	AddressEntityType (see 5.2.2)	M	Entity part(s) of the destination address. SHALL be present. See also 5.2.3, especially concerning restrictions on the supported depth.
datagram. header. addressDestination. feature	AddressFeatureType (see 5.2.2)	M	Feature part of the destination address. SHALL be present.
datagram. header. addressOriginator	FeatureAddressType (see 5.2.2)	O	Can be used to model information of a “forwarded” SPINE message. The element would contain the address of the original submitter. MAY be present. See also 5.3.2.2.2.
datagram. header. addressOriginator. device	AddressDeviceType; see “Device address” in 5.5.2.2.5.2	M	Device part of the originator address. SHALL be present.
datagram. header. addressOriginator. entity (list)	AddressEntityType (see 5.2.2)	M	Entity part(s) of the originator address. SHALL be present. See also 5.2.3, especially concerning restrictions on the supported depth.
datagram. header. addressOriginator. feature	AddressFeatureType (see 5.2.2)	M	Feature part of the originator address. SHALL be present.
datagram. header. msgCounter	xs:unsignedLong	M	The message number of the submitter of this SPINE message. SHALL be present.

Element name	Type	M/O/NV/C	Brief explanation
datagram. header. msgCounterReference	xs:unsignedLong	O	The message number of the related SPINE message. SHALL NOT be set if the message does not relate to another message. Otherwise it SHALL be present and set to the message number of the related message.
datagram. header. cmdClassifier	CmdClassifierType, see 5.3.2.4	M	The so-called “classifier” associated to the given SPINE function. This denotes for which kind of operation a function is used (read, write, notify, etc.). SHALL be present. See 5.3.2.4.
datagram. header. ackRequest	xs:Boolean	O	Indicates the kind of “acknowledgement request” of the message. The value “true” indicates that an explicit acknowledgement message for this message is requested. May be present. If absent, the default value “false” applies. See 5.3.2.5.1 for details.
datagram. header. timestamp	AbsoluteOrRelativeTimeType (see 5.2.2)	O	The timestamp of the creation of this SPINE message. May be present. See 5.3.2.6.

5.3.3 Payload

5.3.3.1 General information

Within the element “payload”, a single “command” can be placed (broadly speaking; the data model is also prepared for future extensions, but this is not considered further in detail in this document); see Figure 12 It permits or even requires the presence of additional elements to express/identify a functionality. Details are discussed in the subsequent sections.

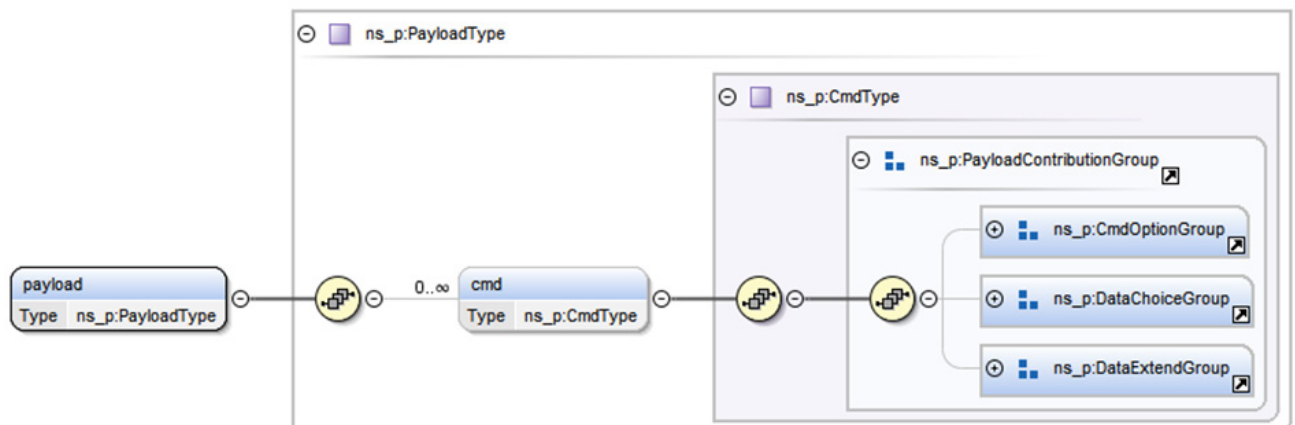


Figure 12 — SPINE payload

5.3.3.2 Elements and usage

Within the protocol specification, the elements of the SPINE payload SHALL be set as shown in Table 30.

Table 30 — Elements of the SPINE payload

Element name	Type	M/O/NV/C	Brief explanation
datagram. payload		M	The payload element SHALL be present.
datagram. payload. cmd		1..unbounded	Each “cmd” instance can take information for one function. In this document just one (the first) occurrence of “cmd” within “payload” SHALL be used (i.e. although the data model permits the occurrence of multiple “cmd” instances in theory, just one is considered).
datagram. payload. cmd. function	FunctionType (see 5.2.2)	O	Contains the function name of element “datagram.payload.cmd.<FUNCTION>”. SHALL be present if datagram.payload.cmd.filter is present. SHALL be absent otherwise. See 5.3.3.4.
datagram. payload. cmd. filter		0..unbounded	Identifies the content that shall be restricted (see 5.3.3.4). Although the model permits multiple occurrences, this document permits two occurrences at maximum. SHALL ONLY be present if at least one child element is present.
datagram. payload. cmd. filter. filterId	xs:unsignedInt	O	Reserved for future use.
datagram. payload. cmd. filter. cmdControl		O	Specifies the kind of function/data restriction. SHALL be present for specified function/data restrictions only. SHALL be absent otherwise. See 5.3.3.4. In this document, exactly one of the possible child elements SHALL be present if datagram.payload.cmd.filter.cmdControl is present.
datagram. payload. cmd. filter. cmdControl. delete		O	Denotes the restriction is a “delete” operation.
datagram. payload. cmd. filter. cmdControl. partial		O	Denotes the restriction is a “partial” operation.
datagram. payload. cmd. filter. <SELECTORS>		0..unbounded	“<SELECTORS>” is a placeholder for SPINE class specific “Selectors” definitions. E.g. for a SPINE function “X” there might be a specific “XSelectors” defined. Please look at the SPINE data model for a list of all possible entries. If present, it SHALL be an instance of the specific “selectors” of the function “datagram.payload.cmd.<FUNCTION>”. I.e. with the function name of “...cmd.<FUNCTION>” abbreviated with “X”, the “Selectors” name in “<SELECTORS>” SHALL be “XSelectors”. Can occur multiple times. All occurrences shall be interpreted as logical “OR” operation.

Element name	Type	M/O/NV/C	Brief explanation
datagram. payload. cmd. filter. <ELEMENTS>		O	“<ELEMENTS>” is a placeholder for SPINE class specific “Elements” definitions. E.g. for a SPINE function “X” there might be a specific “XElements” defined. Please look at the SPINE data model for a list of all possible entries. If present, it SHALL be an instance of the specific “Elements” of the function “datagram. payload. cmd. <FUNCTION>”. I.e. with the function name of “...cmd.<FUNCTION>” abbreviated with “X”, the “Elements” name in “<ELEMENTS>” SHALL be “XElements”.
datagram. payload. cmd. <FUNCTION>		M	“<FUNCTION>” is a placeholder for exactly one SPINE function. SHALL be present.
datagram. payload. cmd. manufacturerSpecificExtension	xs:hexBinary	O	Can be used to extend a given function with proprietary data. Please note it shall only be used together with a function. MAY be present.
datagram. payload. cmd. lastUpdatedAt	AbsoluteOrRelativeTimeType (see 5.2.2)	O	Can be used in case an implementation caches data of the actual node. MAY be present.

On each feature there SHALL be at maximum one class implemented with regard to the feature's primary functionality.

The general XSD-based SPINE payload definition permits the presence of multiple “cmd” instances and therefore multiple functions in theory. However, within this version of the protocol specification each instance of a “payload” element SHALL contain exactly one “cmd” instance.

5.3.3.3 Ownership

The combination of functions and classifiers/operations of the SPINE data model should be applied considering the concept of “ownership”. The concept of “ownership” is directly linked to the SPINE role concept. The “ownership” of data are always defined on SPINE feature level.

The “owner” of data (functions) may

- notify its own data,
- reply a request (i.e. received read operation) with its own data,
- update its own data upon request (i.e. upon received write operation).

This, of course, only provided that these actions are supported by the owner.

Especially the write operation should be considered. As example we assume device “A” with a client feature “AA” submits a write operation with data “X” to device “B” with server feature “BB”. Then, data “X” shall be considered to be owned and “known” by feature “BB” of device “B”. I.e. device “A” should not expect device “B” considers “X” as data owned by feature “AA” of device “A”.

Call operations and acknowledgement messages (with classifier “result”) have a different scope, hence do not belong to the concept of ownership.

5.3.3.4 Restricted function exchange with cmdOptions

5.3.3.4.1 Overview

The restricted function exchange (RFE) concept is used to exchange just a certain restricted part of a function instead of the full function. The following elements of “datagram.payload.cmd” are subsequently called “cmdOptions”:

1. datagram.payload.cmd.function
2. datagram.payload.cmd.filter
3. datagram.payload.cmd.filter.cmdControl

The cmdOption “datagram.payload.cmd.function” SHALL be used and include the correct function name if and only if any of the other cmdOptions is used.

If full functions are requested or exchanged (“full function exchange”), cmdOptions are NOT used.

Please note the cmdOption “datagram.payload.cmd.function” just serves as some kind of preface in order to introduce which function to operate on in some cases. I.e. it DOES NOT contain function data.

The basic difference between restricted and full functions is determined by the absence or presence of cmdOptions.

Please note: The following subsections show cmdOptions combinations that also contain the case of full function exchange for comparison.

In general, the support of restricted function exchange is optional unless a featureType or Use Case requires the support explicitly. Even then, the featureType or Use Case may support only specific kinds of restricted function exchange. Consider also 5.3.3.4.9.

Let us assume “T” is a data model definition (including proper presence indications for each element) for a full function exchange. An example for “T” is the SPINE function “smartEnergyManagementPsData” of the feature type “SmartEnergyManagementPs”. Subclause 5.3.3.4 defines general requirements of “restricted function exchange” and it defines for specific cases which elements are required at least. For example, a “notify” with “cmdControl” set to “partial” can be used to convey a <FUNCTION> that contains added or modified function parts. In this case added and modified elements are required, whereas unchanged elements (even if the definition of “T” designates them as “mandatory”) are not required (unless rules of 5.3.3.4 impose specific requirements as in case of, e.g. identifiers). Of course, for each required (child) element its respective parent elements are always mandatory.

Remark: A featureType may specify one or more specifically restricted variants of “T”. For example, the feature type “SmartEnergyManagementPs” defines the “primary use” of the SPINE function “smartEnergyManagementPsData” to express energy management related information of a device (conveyed in a “read” or “notify” operation). Additionally, the featureType also defines variants of this SPINE function to shift a functionality of the device or select a different option (these variants use “write” operations). These variants can be seen as extracts or simplifications of the “primary use” of the function. They are especially useful to describe dedicated changes of the device's SPINE function.

On the cardinality of “filter”:

As shown in Table 30, the element “datagram.payload.cmd.filter” may occur more than one time. This cardinality is used to permit a restricted function exchange with two “filter” parts within one message: One “filter” part with sub-element “cmdControl” set to “delete” and another “filter” part with sub-element “cmdControl” set to “partial”. As brief example we take one of the permitted “write” cmdOptions combinations of 5.3.3.4.2 where a “delete” as well as a “partial” part are used (functional details on these parts are explained in 5.3.3.4.2). We assume some “xyzData” list entries of the function “xyzListData” are deleted and some list entries are partially modified for this example. The “payload” part could then look like this (with some content replaced by “...” to improve readability):