March 2023

Specification of the Asset Adminitration Shell

Part 1: Metamodel

SPECIFICATION

IDTA Number: 01001

Specification of the Asset Administration Shell. Part 1: Metamodel

Chapter 1. Preamble

1.1. Editorial Notes

This document (version 3.0) was produced from June 2022 to January 2023 by the joint sub working group "Asset Administration Shell" of the working group "Reference Architectures, Standards and Norms" of the Plattform Industrie 4.0 and the working group "Open Technology" of the Industrial Digital Twin Association (IDTA). It is the first release published by the Industrial Digital Twin Association.

A major change consists in splitting the overall document into four parts: Part 1 (this document) covers the core metamodel of the Asset Administration Shell, Part 5 covers the AASX package exchange format, Part 3 is a series that covers the predefined data specifications and Part 4 covers the security metamodel. Another major change is that the mapping rules for the different supported exchange formats (XML, JSON and RDF) are moved to the github repositories themselves that also contain the schemata. Part 2, the API Specification, was defined in a separate document from the very beginning.

Version 3.0RC02 was produced from November 2020 to May 2022 by the sub working group "Asset Administration Shell" of the joint working group of the Plattform Industrie 4.0 working group "Reference Architectures, Standards and Norms" and the "Open Technology" working group of the Industrial Digital Twin Association.

Version 3.0RC01 of this document, published in November 2020, was produced from November 2019 to November 2020 by the sub working group "Asset Administration Shell" of the Plattform Industrie 4.0 working group "Reference Architectures, Standards and Norms".

The second version V2.0 of this document was produced from August 2018 to November 2019 by the sub working group "Asset Administration Shell" of the Platform Industrie 4.0 working group "Reference Architectures, Standards and Norms". Version 2.0.l was published in May 2020.

The first version of this document was produced September 2017 to July 2018 by a joint working group with members from ZVEI SG "Models and Standards" and the Plattform Industrie 4.0 working group "Reference Architectures, Standards and Norms ". The document was subsequently validated by the platform’s working group "Reference Architectures, Standards and Norms".

For better readability the abbreviation "I4.0" is consistently used for "Industrie 4.0" in compound terms. The term "Industrie 4.0" continues to be used when standing on its own.

This specification is versioned using Semantic Versioning 2.0.0 (semver) and follows the semver specification [36].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be

interpreted as described in BCP 14 RFC2119 RFC8174[1]:

- MUST word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- MUST NOT This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.
- SHOULD This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

1.2. Scope of this Document

The aim of this document is to define selected specifications of the structure of the Administration Shell to enable the meaningful exchange of information about assets and I4.0 components between partners in a value creation network.

This part of the document focuses on how such information needs to be processed and structured. In order to define these specifications, the document formally stipulates some structural principles of the Administration Shell. This part does not describe technical interfaces of the Administration Shell or other systems to exchange information, protocols, or interaction patterns.

This document focuses on:

- a metamodel for specifying information of an Asset Administration Shell and its submodels,
- exchange format for the transport of information from one partner in the value chain to the next,
- identifiers,
- access control,

- an introduction to the need of mappings to suitable technologies used in different life cycle phases of a product: XML, JSON, RDF, AutomationML, and OPC UA.

This document presumes some familiarity with the concept of the Asset Administration Shell. Some of the concepts are described in Annex A for convenience sake. The concepts are being standardized as IEC standard IEC 63278 series [44]. The main stakeholders addressed in this document are architects and software developers aiming to implement a digital twin using the Asset Administration Shell in an interoperable way. Additionally, the content can also be used as input for discussions with international standardization organizations and further initiatives. Please consult the continuously updated reading guide [38] for an overview of documents on the Asset Administration Shell. The reading guide gives advice on which documents should be read depending on the role of the reader.

1.3. Structure of the Document

All clauses that are normative have "(normative)" as a suffix in the heading of the clause.

Clause 2 provides terms and definitions as well as abbreviations, both for abbreviations used in the document and for abbreviations that may be used for elements of the metamodel defined in this document.

Clause 3 gives a short introduction into the content of this document.

Clause 4 summarizes relevant, existing content from the standardization of Industry 4.0; i.e. it provides an overview and explains the motives, but is not absolutely necessary for an understanding of the subsequent definitions.

Clause 5 is the main normative part of the document. It stipulates structural principles of the Administration Shell in a formal manner to ensure an exchange of information using Asset Administration Shells. A UML diagram has been defined for this purpose.

Clause 6 explains how to define predefined data specifications, including those for defining concept descriptions.

Clause 7 provides information on the exchange of information compliant to this specification in existing data formats like XML, AutomationML, OPC UA information models, JSON, or RDF.

Clause 8 summarizes the content and gives an outlook on future work.

Annex A contains additional background information on Asset Administration Shell, while a mapping to the requirements can be found in Annex B.

Annex C defines the grammar language used in the specification. Annex E contains information about UML, while Annex D provides the tables used to specify UML classes etc. as used in this specification.

Annex H describes metamodel changes compared to previous versions. Annex F provides some hints for

modelers, and Annex G shows selected metamodel diagrams including all inherited attributes for developers.

The bibliography can be found in Annex I.

1.4. Working Principles

The work is based on the following principle: keep it simple but do not simplify if it affects interoperability.

To create a detailed specification of the Administration Shell according to the scope of Part 1 result papers published by Plattform Industrie 4.0, the trilateral cooperation between France, Italy, and Germany, as well as international standardization results were analyzed and taken as source of requirements for the specification process. As many ideas as possible from the discussion papers were considered. See Annex A for more information.

The partners represented in the Plattform Industrie 4.0 and the Industrial Digital Twin Association (IDTA) and associations such as ZVEI, VDMA, VDI/ VDE and Bitkom, ensure that there is broad sectoral coverage of process, hybrid, and factory automation and in terms of integrating information technology (IT) and operational technology (OT).

Design alternatives were intensively discussed within the working group. An extensive feedback process of this document series is additionally performed within the working groups of Plattform Industrie 4.0 and IDTA.

Guiding principle for the specification was to provide detailed information, which can be easily implemented also by small and medium-sized enterprises.

[1] https://www.ietf.org/rfc/rfc2119.txt

Chapter 2. Terms, Definitions and Abbreviations

2.1. Terms and Definitions

> Please note: the definitions of terms are only valid in a certain context. This glossary applies only within the context of this document.

If available, definitions were taken from IEC 63278-1 DRAFT, July 2022.

access control

protection of system resources against unauthorized access; a process by which use of system resources is regulated according to a security policy and is permitted by only authorized entities (users, programs, processes, or other systems) according to that policy

- [SOURCE: IEC TS 62443-1-1]

application

software functional element specific to the solution of a problem in industrial-process measurement and control

> Note 1 to entry: an application can be distributed among resources and may communicate with other applications.

- [SOURCE: IEC TR 62390:2005-01, 3.1.2]

asset

physical, digital, or intangible entity that has value to an individual, an organization, or a government

> Note 1 to entry: an asset can be single entity, a collection of entities, an assembly of entities, or a composition of entities.

> EXAMPLE 1: examples for physical entities are equipment, raw material, parts components and pieces, supplies, consumables, physical products, and waste.

EXAMPLE 2: examples for digital assets are process definitions, business procedures, or actual states.

EXAMPLE 3: a software license is an example of an intangible asset.

- [SOURCE: IEC 63278-1, based on IEV 741-01-04, editorial changes]

attribute

data element of a *property*, a relation, or a class in information technology

- [SOURCE: ISO/IEC Guide 77-2, ISO/IEC 27460, IEC 61360]

Asset Administration Shell (AAS)

standardized digital representation of an asset

Note 1 to entry: Asset Administration Shell and Administration Shell are used synonymously.

- [SOURCE: IEC 63278-1, note added]

class

description of a set of objects that share the same *attributes*, *operations*, methods, relationships, and semantics

- [SOURCE: IEC TR 62390:2005-01, 3.1.4]

capability

implementation-independent potential of an Industrie 4.0 component to achieve an effect within a domain

Note 1 to entry: capabilities can be orchestrated and structured hierarchically.

Note 2 to entry: capabilities can be made executable via services.

Note 3 to entry: the impact manifests itself in a measurable effect within the physical world.

- [SOURCE: Glossary Industrie 4.0, minor changes]

coded value

value that can be looked up in a dictionary and can be translated

- [SOURCE: ECLASS [1]]

component

product used as a constituent in an assembled product, *system*, or plant

- [SOURCE: IEC 63278-1; IEC 61666:2010, 3.6, editorial changes]

concept

unit of knowledge created by a unique combination of characteristics

- [SOURCE: EC 63278-1; IEC 61360-1:2016, 3.1.8; ISO 22274:2013, 3.7]

digital representation

information and services representing an entity from a given viewpoint

> EXAMPLE 1: examples of information are properties (e.g. maximum temperature), actual parameters (e.g. actual velocity), events (e.g. notification of status change), schematics (electrical), and visualization information (2D and 3D drawings).

> EXAMPLE 2: examples of services are providing the history of the configuration data, providing the actual velocity, and providing a simulation.

> EXAMPLE 3: examples of viewpoints are mechanical, electrical, or commercial characteristics.

- [SOURCE: IEC 63278-1, editorial changes]

digital twin

digital representation, sufficient to meet the requirements of a set of use cases

> Note 1 to entry: in this context, the entity in the definition of digital representation is typically an asset.

- [SOURCE: IIC Vocabulary IIC:IIVOC:V2.3:20201025, adapted (an asset, process, or system was changed to an asset)]

explicit value

commonly used *concept*, like numbers (e.g. 109, 25) which do not need lookup in dictionaries

- [SOURCE: ECLASS]

identifier (ID)

identity information that unambiguously distinguishes one entity from another one in a given domain

> Note 1 to entry: there are specific identifiers, e.g. UUID Universal unique identifier, IEC 15418 (GS1).

- [SOURCE: Glossary Industrie 4.0]

instance

concrete, clearly identifiable component of a certain *type*

> Note 1 to entry: an individual entity of a type, for example a device, is obtained by defining specific property values.

> Note 2 to entry: in an object-oriented view, an instance denotes an object of a class (of a type).

- [SOURCE: IEC 62890:2016, 3.1.16 65/617/CDV, editorial changes]

instance asset

specific *asset* that is uniquely identifiable

> EXAMPLE 1: examples of instance assets are material, a product, a part, a device, a machine, software, a control system, a production system.

- [SOURCE: IEC 63278-1, editorial changes]

operation

executable realization of a function

> Note 1 to entry: the term method is synonymous to operation.

Note 2 to entry: an operation has a name and a list of parameters [ISO 19119:2005, 4.1.3].

- [SOURCE: Glossary Industrie 4.0, editorial changes]

ontology

collection of concepts, where each concept is constituted by an identifier, name, description, and additional entities and where relationships between concepts can be described without restriction

- [SOURCE: IEC 63278-1]

property

defined characteristic suitable for the description and differentiation of products or components

Note 1 to entry: the concept of type and instance applies to properties.

Note 2 to entry: this definition applies to properties as described in IEC 61360/ ISO 13584-42.

Note 3 to entry: the property types are defined in dictionaries (like IEC component data dictionary or ECLASS), they do not have a value. The property type is also called data element type in some standards.

Note 4 to entry: the property instances have a value and are provided by the manufacturers. A property instance is also called property-value pair in certain standards.

Note 5 to entry: properties include nominal value, actual value, runtime variables, measurement values, etc.

Note 6 to entry: a property describes one characteristic of a given object.

Note 7 to entry: a property can have attributes such as code, version, and revision.

Note 8 to entry: the specification of a property can include predefined choices of values.

- [SOURCE: according to ISO/IEC Guide 77-2] as well as [SOURCE: according to Glossary Industrie 4.0]

qualifier

well-defined element associated with a *property* instance or *submodel element*, restricting the value statement to a certain period of time or use case

> Note 1 to entry: qualifiers can have associated values.

- [SOURCE: according to IEC 62569-1]

service

Demarcated scope of functionality which is offered by an entity or organization via interfaces

> Note 1 to entry: one or multiple operations can be assigned to one service.

- [SOURCE: Glossary Industrie 4.0]

smart manufacturing

manufacturing that improves its performance aspects with integrated and intelligent use of processes and resources in cyber, physical and human spheres to create and deliver products and services, which also collaborates with other domains within enterprises' value chains

> Note 1 to entry: performance aspects include agility, efficiency, safety, security, sustainability, or any other performance indicators identified by the enterprise.

> Note 2 to entry: in addition to manufacturing, other enterprise domains can include engineering, logistics, marketing, procurement, sales, or any other domains identified by the enterprise.

- [SOURCE: IEC TR 63283-1:2022 ED1]

Submodel

container of SubmodelElements defining a hierarchical structure consisting of SubmodelElements

- [SOURCE: IEC 63278-1]

SubmodelElement

elements in a Submodel

- [SOURCE: IEC 63278-1]

Submodel template

container of Submodel template elements defining a hierarchical structure consisting of Submodel template elements

> Note 1 to entry: a Submodel template is a specific kind of concept.

- [SOURCE: IEC 63278-1]

Submodel template element

elements in a Submodel template

> Note 1 to entry: a Submodel template element is a specific kind of concept.

- [SOURCE: IEC 63278-1]

system

interacting, interrelated, or interdependent elements forming a complex whole

- [SOURCE: IEC 63278-1; IEC TS 62443-1-1:2009, 3.2.123]

technical functionality

functionality of the *Administration Shell* that is exposed by an application programming interface (API) and that is creating added value to the respective *assets(s)*

> Note 1 to entry: can consist of single elements, which are also known as functions, operations, methods, skills.

- [SOURCE: according to [18]]

template

specification of the common features of an object in sufficient detail that such object can be instantiated using it

> Note 1 to entry: object can be anything that has a type.

- [SOURCE: according to ISO/IEC 10746-2]

type

hardware or software element which specifies the common *attributes* shared by all instances of the type

- [SOURCE: IEC TR 62390:2005-01, 3.1.25]

type asset

(abstract) representation of a set of instance assets with common characteristics and features

> Note 1 to entry: the set of instance assets may exist or may not exist. Examples of type assets are type of material, a product type, a type of a part, a device type, a machine type, a type of software, a type of control system, a type of production system.

- [SOURCE: IEC 63278-1]

variable

software *entity* that may take different values, one at a time

- [SOURCE: IEC 61499-1]

2.2. Abbreviations Used in this Document

Abbreviation	Description
AAS	Asset Administration Shell
AASX	Package file format for the Asset Administration Shell
AML	AutomationML
API	Application Programming Interface
BITKOM	Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V.
BLOB	Binary Large Object
CDD	Common Data Dictionary
GUID	Globally unique identifier
I4.0	Industrie 4.0
ID	identifier

Abbreviation	Description
IDTA	Industrial Digital Twin Association
IEC	International Electrotechnical Commission
IRDI	International Registration Data Identifier
IRI	Internationalized Resource Identifier
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
OPC	Open Packaging Conventions (ECMA-376, ISO/IEC 29500-2)
OPC UA	OPC Unified Architecture
PDF	Portable Document Format
RAMI4.0	Reference Architecture Model Industrie 4.0
RDF	Resource Description Framework
REST	Representational State Transfer
RFC	Request for Comment
SOA	Service Oriented Architecture
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
UTC	Universal Time Coordinated
VDE	Verband der Elektrotechnik, Elektronik und Informationstechnik e.V.
VDI	Verein Deutscher Ingenieure e.V.
VDMA	Verband Deutscher Maschinen- und Anlagenbau e.V.
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
ZIP	archive file format that supports lossless data compression
ZVEI	Zentralverband Elektrotechnik- und Elektronikindustrie e. V.

2.3. Abbreviations of Metamodel

The following abbreviations are not used in the document but may be used as abbreviations for the elements in the metamodel defined in this document.

1. Elements with Allowed Identifying Values

Abbreviation	**Description**
AAS	Asset Administration Shell
Cap	Capability
CD	Concept Description
DE	DataElement
DST	DataSpecification Template
InOut	inoutputVariable
In	inputVariable
Prop	Property
MLP	MultiLanguageProperty
Range	Range
Ent	Entity
Evt	Event
File	File
Blob	Blob
Opr	Operation
Out	outputVariable
Qfr	Qualifier
Ref	ReferenceElement
Rel	RelationshipElement
RelA	AnnotatedRelationshipElement
SM	Submodel
SMC	SubmodelElementCollection

SME	SubmodelElement
SML	SubmodelElementList

[1] In IEC61360:2017, this refers to a "term" of a value list

Chapter 3. Introduction

This document specifies the information metamodel of the Asset Administration Shell.

The general concept and the structure of the Asset Administration Shell is described in IEC 63278-1 (see Figure 1).

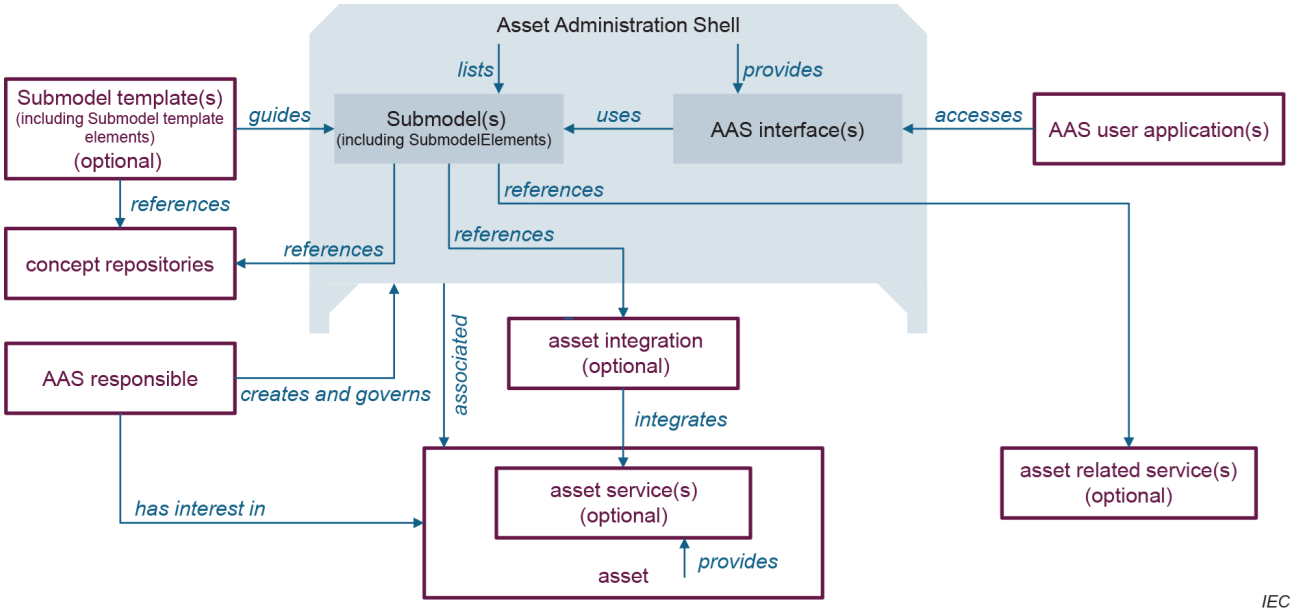


Figure 1. Asset Administration Shell and Related Roles (Source: IEC 63278-1)

These are the main specifics and roles defined for the Asset Administration Shell:

- an Asset Administration Shell has an association to an asset,
- an Asset Administration Shell provides an interface or several interfaces,
- an Asset Administration Shell lists one or several submodels,
- an Asset Administration Shell responsible creates and governs the Asset Administration Shell,
- an Asset Administration Shell user application accesses the information of the Asset Administration Shell via IT interface(s).
- a Submodel template guides the creation of a submodel following the template,
- a Submodel template may reference concept dictionaries and ontologies,
- concept dictionaries and ontologies define the common vocabulary as basis for interoperability,
- submodels may reference the asset services provided by an asset via an asset integration; further services related to the asset can be referenced.

This document specifies a technology-neutral specification of the information metamodel of the Asset Administration Shell in UML. It serves as the basis for deriving several different formats for exchanging Asset Administration Shells, e.g. for XML, JSON, RDF, AutomationML, and OPC UA information models.

Figure 2 shows the different ways of exchanging information Asset Administration Shells. This part of the "Asset Administration Shell in Detail" series is the basis for all of these types of information exchange.

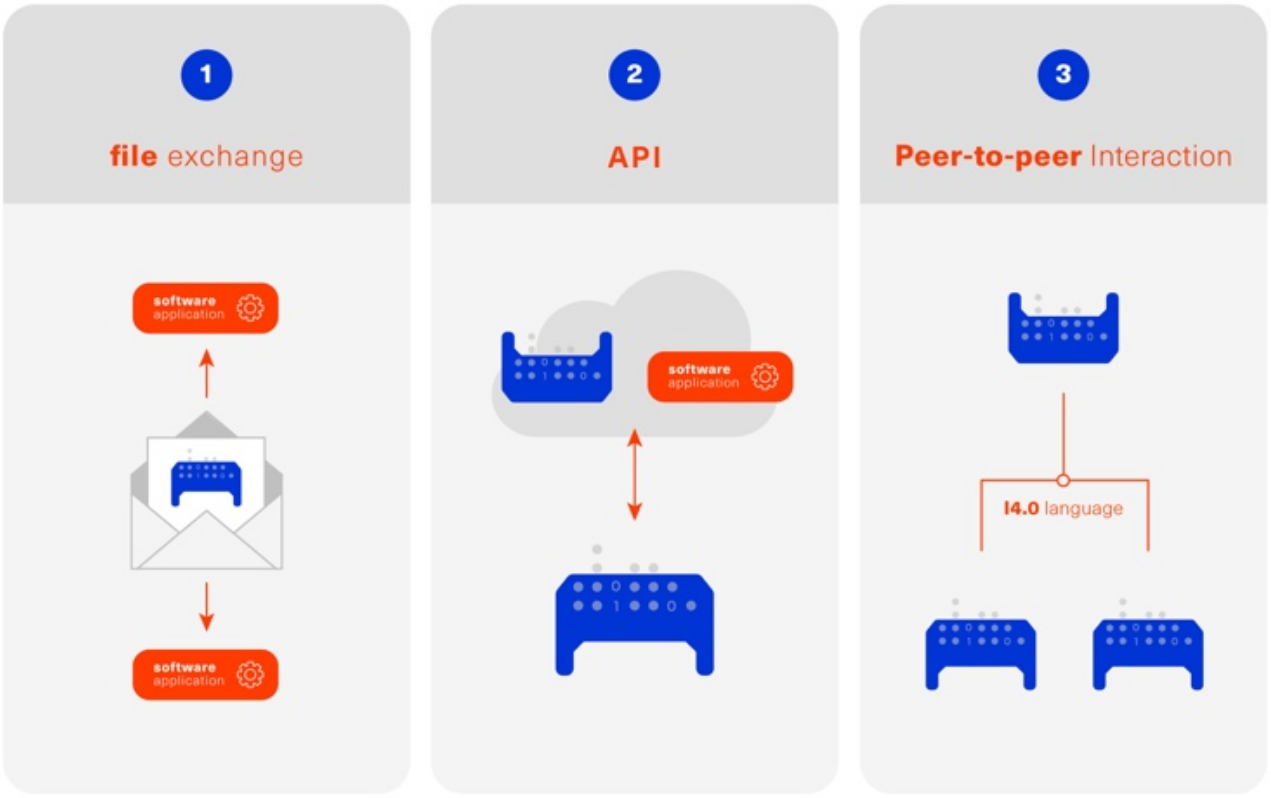


Figure 2. Types of Information Exchange via Asset Administration Shells

File exchange (1) is described in detail in Part 5 of this document series.

The API (2) based on the information metamodel specified in this document is specified in Part 2 of the document series "Details of the Asset Administration Shell" [37].

The I4.0 language (3) is based on the information metamodel specified in this document [47].

Chapter 4. General

4.1. Introduction

Before specifying the information metamodel of the Asset Administration Shell, some general topics relevant for the Asset Administration Shell are explained.

Subclause 4.2 describes some general aspects of handling type and instance assets.

Subclause 4.3 explains the very important aspects of identification in the context of the Asset Administration Shell.

Subclause 4.4 provides matching strategies for semantic identifiers and references.

Subclause 4.5 explains the difference between submodel instances and templates.

Subclause 4.6 discusses aspects of event handling.

4.2. Types and Instances

4.2.1. Life Cycle with Type Assets and Instance Assets

Industry 4.0 utilizes an extended understanding of assets, comprising elements such as factories, production systems, equipment, machines, components, produced products and raw materials, business processes and orders, immaterial assets (such as processes, software, documents, plans, intellectual property, standards), services, human personnel, etc..

The RAMI4.0 model [3] defines a generalized life cycle concept derived from IEC 62890. The basic idea is to distinguish between possible types and instances for all assets within Industry 4.0. This makes it possible to apply the type/instance distinction for all elements such as material type/material instance, product type/product instance, machine type/ machine instance, etc. Business-related information is handled on the 'business' layer of the RAMI4.0 model. The business layer also covers order details and workflows, again for both type and instance assets.

> Note: to distinguish asset 'type' and asset 'instance', the term 'asset kind' is used in this document. The three different relationship classes between assets, especially type assets and instance assets, explained below show why the distinction is so important. The attribute "derivedFrom" in the metamodel is used to explicitly state a relationship between assets that are being derived from one another. Other relationships are not explicitly supported by the metamodel of the Asset Administration Shell, but they can be modelled via the "RelationshipElement" submodel element type.

Table 1 gives an overview of the different life cycle phases and the role of type assets and instance assets as well as their relationship in these phases.

This important relationship should be maintained throughout the life of the instance assets. It makes it possible to forward updates from the type assets to the instance assets, either automatically or on demand.

Table 1. Life Cycle Phases and Roles of Type and Instance Assets

Asset Kind	Life Cycle Phase	Description
Type asset	Development	Valid from the ideation/conceptualization to the first prototypes/test. The 'type' of an asset is defined; distinguishing properties and functionalities are defined and implemented. All (internal) design artefacts associated with the type asset are created, such as CAD data, schematics, embedded software.
	Usage/ Maintenance	Ramping up production capacity. The 'external' information associated to the asset is created, such as technical data sheets, marketing information. The selling process starts.
Instance asset	Production	Instance assets are created/produced, based on the type asset information. Specific information about production, logistics, qualification, and test are associated with the instance assets.
	Usage/ Maintenance	Usage phase by the purchaser of the instance assets. Usage data is associated with the instance asset and might be shared with other value chain partners, such as the manufacturer of the instance asset. Also included: maintenance, re-design, optimization, and de-commissioning of the instance asset. The full life cycle history is associated with the asset and might be archived/shared for documentation.

The second class of relationships are feedback loops/information within the life cycle of the type asset and instance asset. For product assets, for example, information on usage and maintenance of product instances may be used to improve product manufacturing as well as the design of the (next) product type.

The third class of relationships are feedforward/information exchange with assets of other asset classes. For example, sourcing information from business assets can influence design aspects of products; or the design of the products affects the design of the manufacturing line.

Note: the NIST model [49] provides an illustration of the second/third class of relationships.

A fourth class of relationships consists between assets of different hierarchy levels. For example, these could be the (dynamic) relationships between manufacturing stations and currently produced products. They could be also the decomposition of production systems in physical, functional, or safety hierarchies. In this class of relationships, automation equipment is seen as a complex, interrelated graph of automation devices and products, performing intelligent production and self-learning/optimization tasks.

Details and examples for composite I4.0 Components can be found in [12]. A composite I4.0 Component is the combination of a complex asset and its Asset Administration Shell. The hierarchy, typically a Bill of Material (BOM) but also any other relationship between different assets, can be represented in one of its submodels.

Note: for submodels representing the Bill of Material of a complex asset, the metamodel not only provides the possibility to define relationships (via the submodel element "RelationshipElement", see above), it also explicitly supports the representation of another asset (via the submodel element "Entity"). The term "Entity" is chosen as superordinate concept in this context and refers to either an asset or another item that is not an asset but may be part of a more complex item or asset.

4.2.2. Asset Administration Shells Representing Type Assets and Instance Assets

An Asset Administration Shell either represents a type asset or an instance asset. Typically, there is a relationship between instance assets and a type asset. However, not every instance asset is required to have a corresponding type asset.

gives an example of how to handle type assets and their derived instance assets. The attribute "assetKind" indicates whether the Asset Administration Shell (denoted by the ": AAS" UML notation for a class instance) represents a type asset or an instance asset. Additionally, attributes are added to show that the attributes of type asset and instance assets typically differ from each other.

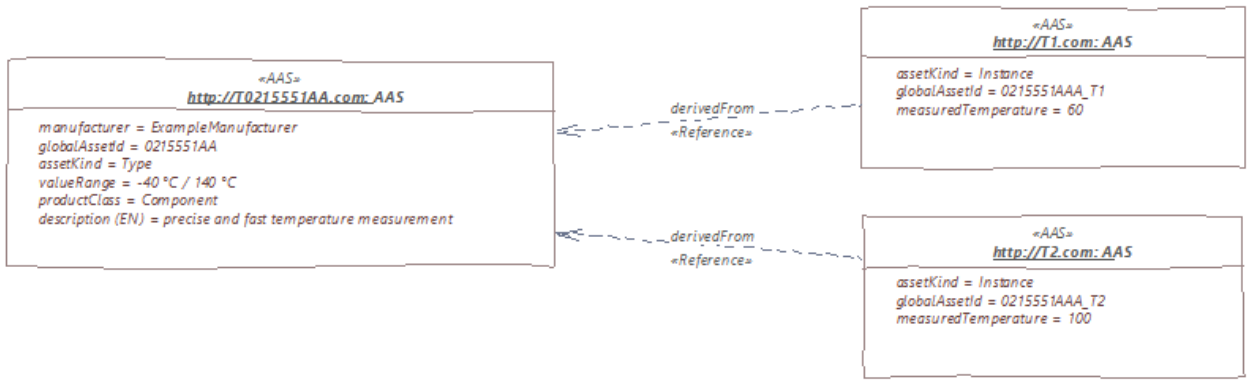


Figure 3. Example: Asset Administration Shells for Type and Instance Assets

Note 1: the example is simplified for ease of understanding and only roughly complies with the metamodel as specified in Clause 5. The ID handling is simplified as well: the names of the classes correspond to the unique global identifier of the Asset Administration Shells.

Note 2: in the context of Plattform Industrie 4.0, types and instances typically refer to "type assets" and "instance assets". When referring to types or instances of an Asset Administration Shell, this is explicitly denoted as "Asset Administration Shell types" and "Asset Administration Shell instances" to not mix them up. Asset Administration Shell types are synonymously used with the term "Asset Administration Shell template".

Note 3: please refer to 2 for the IEC definition of types and instances. Within the scope of this document, there is no full equivalency between these definitions and the type/instance concepts of object-oriented programming (OO).

There shall be a concrete type asset of a temperature sensor and two uniquely identifiable physical temperature sensors of this type. The intention is to provide a separate Asset Administration Shell for the type asset as well as for every single instance asset.

In the example, the first sensor has the unique ID "0215551AAA_T1" and the second sensor has the unique ID "0215551AAA_T2". "0215551AAA_T1" and "0215551AAA_T2" are the global asset IDs of the two assets, i.e. sensors. The Asset Administration Shell for the first sensor has the unique URI "http://T1.com" and the Asset Administration Shell for the second sensor has the unique URI "http://T2.com". The asset kind of both is "Instance". The example shows that the measured temperature at operation time of the two sensors is different: for T1 it is 60 °C, for T2 it is 100 °C. For the time-being we ignore the relationship "derivedFrom" of the two Asset Administration Shells "http://T1.com " and "http://T2.com" with Asset Administration Shell "http://T0215551AA.com".

Note 1: even though the HTTP scheme is used for the identifier, please be aware that these identifiers are logical ones. Identifiers do not have to be URLs. At the same time, URLs used as identifiers do not have to refer to accessible content.

Note 2: the physical unit can be obtained by the semantic reference of the element "measuredTemperature". This is not shown in the example for simplicity reasons.

These two instance assets share a lot of information on the type asset (in this example a sensor type), for which an own Asset Administration Shell is created. The unique ID for this Asset Administration Shell is "http://T0215551AA.com", the unique ID of the sensor type is "0215551AA". The asset kind is "Type" and not "Instance". The information shared by all instances of this temperature sensor type is the product class (="Component"), the manufacturer (="ExampleManufacturer"), the English Description (=”precise and fast temperature measurement"), and the value range ("-40 °C / 140 °C").

Now the two Asset Administration Shells of the two instance assets may refer to the Asset Administration Shell of the type asset "0215551AA" using the relationship attribute "derivedFrom".

Note 1: in the UML sense, "attribute" refers to the property or characteristic of a class (instance).

Note 2: if a specific type asset exists, it typically exists in time before the respective instance assets.

Note 3: the term Asset Administration Shell is used synonymously with the term Asset Administration Shell instance. An Asset Administration Shell may be realized based on an Asset Administration Shell type. Asset Administration Shell types are out of the scope of this document.

Note 4: in public standardization, the Asset Administration Shell types might be standardized. However, it is much more important to standardize the property types (called property definitions or concept descriptions) or other submodel element types as well as complete submodel types because these can be reused in different Asset Administration Shells.

Note 5: in the domain of the Internet of Things (IoT), instance assets are typically denoted as "Things" whereas type assets are denoted as "Product".

4.2.3. Asset Administration Shell Types and Instances

In the previous clause, type assets and instance assets were explained. The obvious question now is how to harmonize Asset Administration Shells and Asset Administration Shell types. The example in shows that the attributes "globalAssetId" and "assetKind" as well as the global Asset Administration Shell identifier (*id*, represented as name of the class) are present for all Asset Administration Shells. However, if there is no standard, the semantics of "id", "globalAssetId" or "kind" are not clear, although they are the same for all Asset Administration Shells. It is also not clear, which of the attributes are mandatory and which are specific for the asset (type or instance), as illustrated in .

This is the purpose of this document: the definition of a metamodel that defines which attributes are mandatory and which are optional for all Asset Administration Shells. The Plattform Industrie 4.0 metamodel for Asset Administration Shells is defined in Clause 5.

Note 1: the metamodel of the Asset Administration Shell is suitable for type assets or instance assets. An alternative approach could have been to define two metamodels, one for type assets and one for instance assets. However, the large set of similarities led to the decision of only one metamodel.

Note 2: the metamodel itself does not require the existence of mandatory submodels. This is another step of standardization similar to the standardization of submodels of the Asset Administration Shell type level.

Note 3: an Asset Administration Shell type shall be realized based on the metamodel of an Asset Administration Shell as defined in this document. This metamodel is referred to as "Asset Administration Shell Metamodel".

Note 4: it is not mandatory to define an Asset Administration Shell type before defining an Asset Administration Shell (instance). An Asset Administration Shell instance that does not realize an Asset Administration Shell type shall be realized based on the metamodel of an Asset Administration Shell as defined in this document.

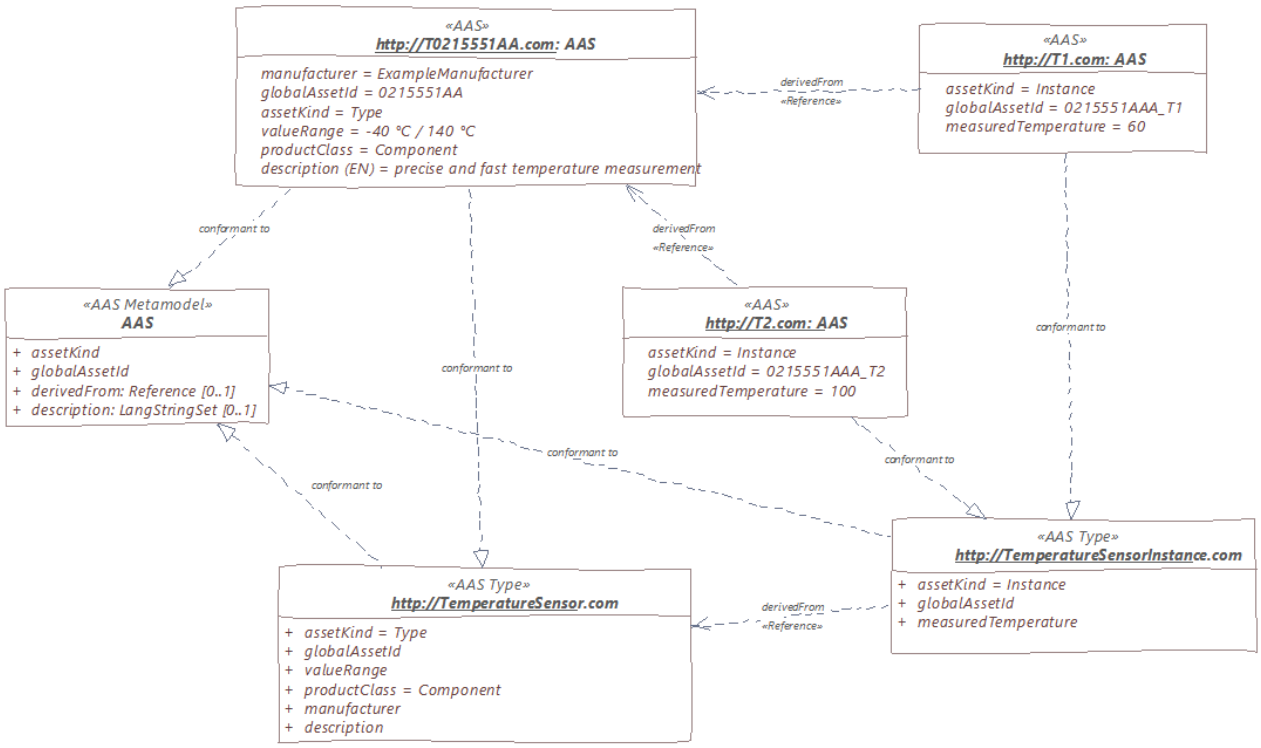


Figure 4. Example: Asset Administration Shell, Asset Administration Shell Types and Instances

4.3. Identification of Elements

4.3.1. Overview

According to [4], identifiers are needed for the unique identification of many different elements within the domain of smart manufacturing. They are a fundamental element of a formal description of the Administration Shell. Identification is especially required for

- Asset Administration Shells,
- assets,
- submodel instances and submodel templates,
- property definitions/concept descriptions in external repositories, such as ECLASS or IEC CDD.

Identification will take place for two purposes

- to uniquely distinguish all elements of an Administration Shell and the asset it is representing, and
- to relate elements to external definitions, such as submodel templates and property definitions, in order to bind semantics to this data and the functional elements of an Administration Shell.

4.3.2. Identifiers for Assets and Administration Shells

In the domain of smart manufacturing, the assets need to be uniquely identified worldwide [4] [20] by the

means of identifiers (IDs). The Administration Shell also has a unique ID (see Figure 5).

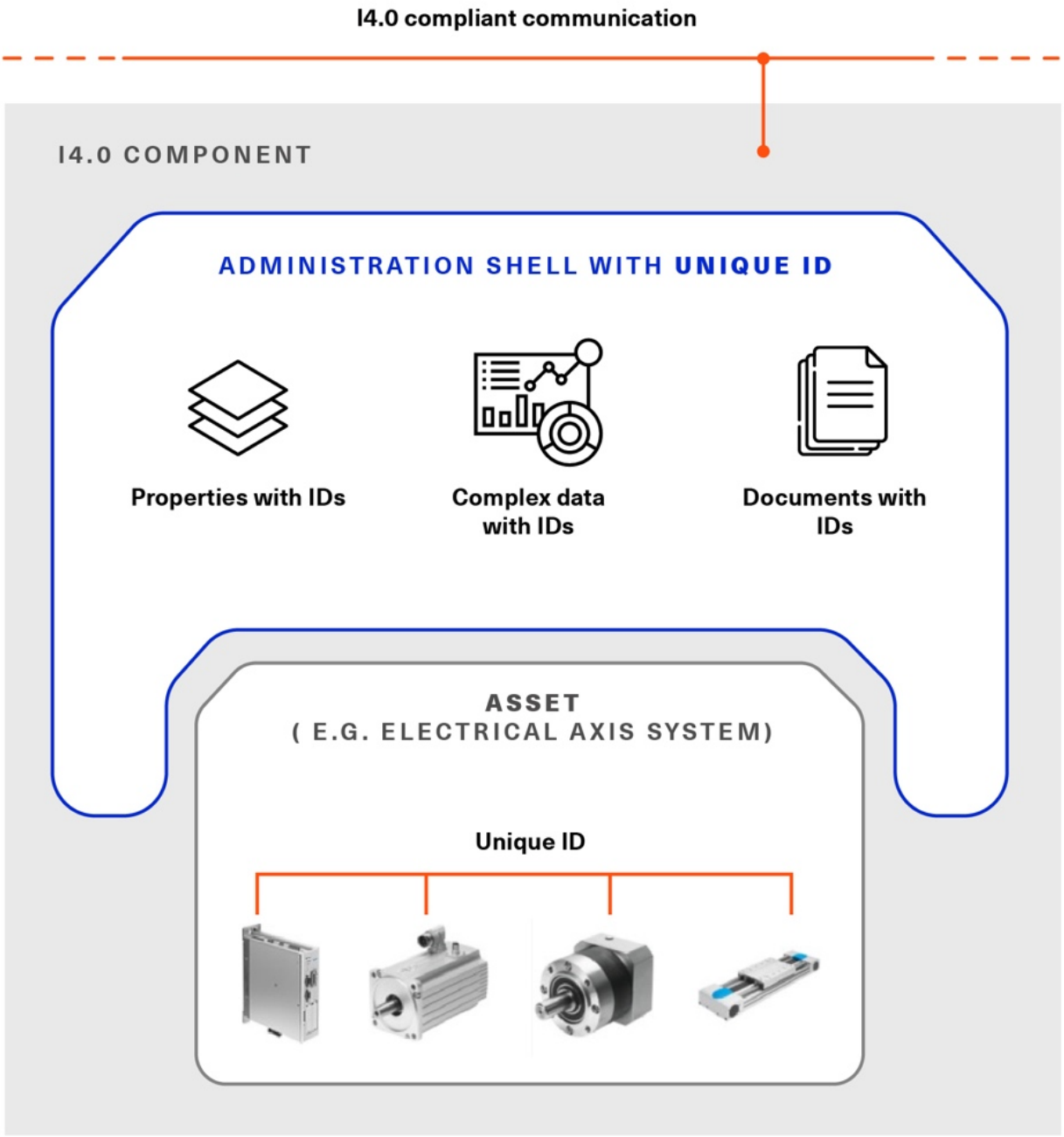


Figure 5. Unique Identifier for Administration Shell and Asset (Modified Figure from [4])

An Administration Shell represents exactly one asset, with a unique asset ID. In a batch-based production, the batches will become the assets and will be described by a respective Administration Shell. If a set of assets shall be described by an Administration Shell, a unique ID for the composite asset needs to be created [13].

The ID of the asset needs to comply with the restrictions for global identifiers according to [4][20]. If the asset features further identifications like serial numbers and alike, they are not to be confused with the unique global identifiers of the asset itself[1].

4.3.3. What Type of Identifiers Exist?

In [4][20], two standard-conforming global identification types are defined:

- **IRDI** – ISO29002-5, ISO IEC 6523 and ISO IEC 11179-6 [20] as an identifier scheme for properties and classifications. They are created in a process of consortium-wise specification or international standardization. To this end, users come together and feed their ideas into the consortia or standardization bodies. Properties in ISO, IEC help to safeguard key commercial interests. Repositories like ECLASS and others make it possible to standardize a relatively large number of identifiers in an appropriately short time.
- **IRI** – IRI (RFC 3987) or URI and URL according to RFC 3986 as identification of assets, Administration Shells and other (probably not standardized, but globally unique) properties and classifications.

The following is also permitted:

- **Custom** – internal custom identifiers such as UUIDs/GUIDs (universally unique identifiers/globally unique identifiers), which a manufacturer can use for all sorts of in-house purposes within the Administration Shell.

This means that the IRIs/URIs/URLs and internal custom identifiers can represent and communicate manufacturer-specific information and functions in the Administration Shell and the 4.0 infrastructure just as well as standardized information and functions. One infrastructure can serve both purposes.

CLSID are URIs for GUIDs. They start with a customer specific schema. Hence, Custom should really only be used if the customer-specific identifier is no IRDI nor IRI.

Besides the global identifiers, there are also identifiers that are unique only within a defined namespace, typically its parent element. These identifiers are also called local identifiers. For example, properties within a submodel have local identifiers.

Besides absolute URIs there are also relative URIs.

See also DIN SPEC 91406 [31] for further information on identification.

4.3.4. Which Identifiers for Which Elements?

Not every identifier is applicable for every element of the UML model representing the Asset Administration Shell. Table 2 therefore gives an overview on the different constraints and recommendations on the various entities, which implement "Identifiable" or "HasSemantics". Attributes relate to the metamodel in Clause 5.1 and Clause 5.3.

See Annex A for more information on how to create new identifiers and best practices for creating URI identifiers.

Table 2. Elements with Allowed Identifying Values

Elements with identifying values	Attribute	Allowed identifiers (recommended or typical)	Remarks
AssetAdministrationShell	id	IRI (URL)	mandatory Typically, URLs will be used.
	idShort	string	optional[2]
	displayName	multi language string	optional
AssetInformation	globalAssetId	IRI	recommended As soon as the Asset Administration Shell is "released" for production or operation, a globalAssetId should be assigned. An Asset ID may be retrieved e.g., by a QR code on the asset, by an RFID for the asset, from the firmware of the asset, or from an asset database. IEC 61406 (formerly DIN SPEC 91406) defines the format of such Asset IDs.
	specificAssetId	IRI, Custom	recommended An asset typically may be represented by several different identification properties like for example the serial number, its RFID code etc. They are used for lookup of Asset Administration Shells in cases the globalAssetId is not available. However, they do not need to be globally unique.
Submodel with kind = Template	id	IRDI, IRI (URI)	mandatory IRDI, if the defined submodel is standardized and has been assigned an IRDI.

Elements with identifying values	Attribute	Allowed identifiers (recommended or typical)	Remarks
	idShort	string	recommended Typically used as idShort for the submodel of kind Instance as well
	displayName	multi language string	recommended Typically used as displayName for the submodel of kind Instance as well
	semanticId	IRDI, IRI (URI)	recommended The semantic ID might refer to an external semantic model defining the semantics of the submodel.
	supplementalSemanticId	IRDI, IRI (URI)	optional
Submodel with kind = Instance	id	IRI (URI), Custom	mandatory
	idShort	string	recommended Typically, the idShort or English short name of the submodel template that is referenced via semanticId.
	displayName	multi language string	optional
	semanticId	IRDI, IRI (URI)	recommended Typically, the semanticId is an external reference to an external standard defining the semantics of the submodel.
	supplementalSemanticId	IRDI, IRI (URI)	optional
SubmodelElement	idShort	string	mandatory Typically, the English short name of the concept definition that is referenced via semanticId.

Elements with identifying values	Attribute	Allowed identifiers (recommended or typical)	Remarks
	displayName	multi language string	optional If no display name is defined in the language requested by the application, the display name is selected in the following order, if available: • the preferred name in the requested language of the concept description defining the semantics of the element, • if there is a default language list defined in the application, the corresponding preferred name in the language is chosen according to this order, • the English preferred name of the concept description defining the semantics of the element, • the short name of the concept description, • the idShort of the element.
	semanticId	IRDI, IRI (URI), Custom	recommended link to a *ConceptDescription* or the concept definition in an external repository via a global ID
	supplementalSemanticId	IRDI, IRI (URI)	optional

Elements with identifying values	Attribute	Allowed identifiers (recommended or typical)	Remarks
ConceptDescription	id	IRDI, IRI, Custom	mandatory *ConceptDescription* needs to have a global ID. If the concept description is a copy from an external dictionary like ECLASS or IEC CDD, it may use the same global ID as it is used in the external dictionary.
	idShort	string	recommended e.g. same as English short name
	displayName	multi language string	optional
	isCaseOf	IRDI, IRI (URI)	optional links to the concept definition in an external repository, which the concept description is a copy from, or that it corresponds to
Qualifier	semanticId	IRDI, IRI (URI), Custom	recommended Links to the qualifier type definition in an external repository IRDI, if the defined qualifier type is standardized and has been assigned an IRDI.

4.3.5. Usage of Short ID for Identifiable Elements

The Administration Shell fosters the use of worldwide unique identifiers to a large degree. However, in some cases, this may lead to inefficiencies. Example: a property, which is part of a submodel, which in turn is part of an Administration Shell, each of which is identified by global identifiers [4].

In an application featuring a resource-oriented architecture (ROA), a worldwide unique resource locator (URL) might be composed of a series of segments, which do not need to be globally unique, see Figure 6.

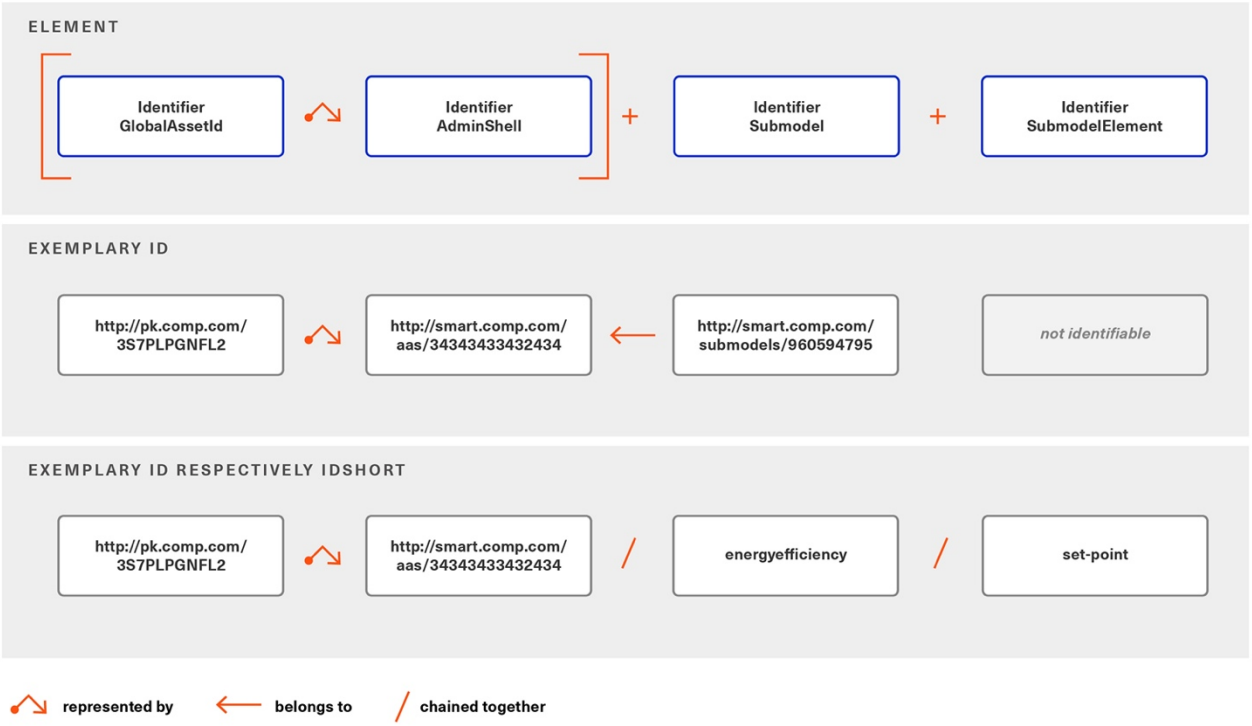


Figure 6. Motivation of Exemplary Identifiers and idShort

To allow such efficient addressing by the chaining of elements by an API of an Administration Shell, *idShort* is provided for a set of classes of the metamodel. It inherits from the abstract class *Referable*, in order to refer to such dependent elements (see 5.1).

Before accessing concrete data provided via a submodel, an application typically checks if the submodel provides the required data, i.e. the semantics of the submodel is checked for suitability. A so-called *semanticId* should be defined for this submodel as well as the submodel element. This semantic ID helps to easily find the semantic definition of the submodel (see Clause 5.3.2).

4.4. Matching Strategies

4.4.1. Matching Strategies for Semantic Identifiers

When comparing two elements, different use cases should be considered in order to define how these two elements are semantically related. This clause gives first hints on the aspects to consider when dealing with matching semantic identifiers. For example, semantic references including context information as represented in IRDI-Path in ECLASS are not yet considered. Sometimes a concept description is derived from another concept description or is identical to or at least compatible with another concept description. The metamodel foresees an attribute "isCaseOf" for such semantic IDs. However, these are not considered in the matching strategies described in this clause.

Exact Matching (identical semanticIds) – DEFAULT

- With exact matching, two semantic IDs need to be string-identical.
  - Example: Property with idShort "ManufacturerName" + semanticId 0173-1#02-AAO677#002 and Property with idShort "Herstellername" + semanticId 0173-1#02-AAO677#002 have exactly equal semantics.

Intelligent Matching (compatible semanticIds)

- Ignore Versioning
  - With intelligent matching, different versions of a concept definition may be matched. For example, if semantic versioning is used to version the concept description, then upward or backward compatible versions can be matched.
  - Example: property with idShort "ManufacturerName" + semanticId 0173-1#02-AAO677#002 and Property with idShort "Herstellername" + semanticId 0173-1#02-AAO677#003 have equal semantics.

> Note: to compare two semantic IDs, knowledge about versioning needs to be available. In the example above, two IRDIs from ECLASS are compared. ECLASS rules ensure that the semantics is always backward compatible for new versions; a new IRDI would be created for breaking changes.

- Consider Semantic Mappings
  - Existing semantic mapping information can be considered for intelligent matching. Semantic mappings may exist within one and the same dictionary, but also between different dictionaries and ontologies.
  - Example: 0112/2///61360_4#AAE530 for nominal capacity of a battery in dictionary IEC CDD and 0173-1#02-AAI048#004 in ECLASS have equal semantics[3] [4].

> Note: this example does not represent an existing semantic mapping; it is only a candidate.

> Note: this example does not represent an existing semantic mapping; it is only a candidate.

- Consider Domain Knowledge
  - With intelligent matching, domain knowledge available in machine-readable form may be taken into account, such as an "is-a"-relationship between two concept definitions.
  - Example: a Hammer drill (0173-1#01-ADS698#010) and a percussion drill (0173-1#01-ADS700#010) are drills for mineral material (0173-1#01-ADN177#005) and are compatible with a request or constraints asking for drills for mineral material.

4.4.2. Matching Algorithm for References

Clause 4.4.1 has discussed matching strategies for semantic identifiers. This clause explains matching strategies based on the reference concept (see Clause 5.3.10) in more detail and covers other kinds of identifying elements.

For example, the string serialization of references as defined in Clause 7.2.3 is used for easier understanding.

Exact matching of two references

- An external reference A matches an external reference B if all values of all keys are identical.

> Note: it is unlikely that a fragment value is identical to a global reference value; it will reference something different.

A model reference A matches a model reference B if all values of all keys are identical.

> Note: the key type can be ignored since the fragment keys are always unique (e.g. all idShorts of submodel elements in a submodel or all submodel elements in a submodel element list or collection).

- An external reference A matches a model reference B and vice versa if all values of all keys are identical.

> Note: since identifiables of the Asset Administration Shell are globally unique, model references are special cases of global references. The only difference is the handling of key types that are predefined for Asset Administration Shell elements. Other key types could be predefined, e.g. for IRDI paths etc. However, so far only generic key types are supported.

> Note: if the key types are not identical although all key values follow the correct order of the key chain, then at least one of the references is buggy and a warning should be issued.

The definition of XML Schema is used for matching

- "(Of string or names:) Two strings or names being compared must be identical.
- Characters with multiple possible representations in ISO/IEC 10646 (e.g. characters with both precomposed and base+diacritic forms) match only if they have the same representation in both strings.
- No case folding is performed.

- (Of strings and rules in the grammar:) A string matches a grammatical production if it belongs to the language generated by that production."

Examples for matching external references[5]:

(GlobalReference)0173-1#01-ADS698#010, (GlobalReference)0173-1#01-ADS700#010

matches

(GlobalReference 0173-1#01-ADS698#010, (FragmentReference)0173-1#01-ADS700#010

Examples for non-matching external references:

(GlobalReference)https://example.com/aas/1/1/1234859590, (FragmentReference)Specification, (FragmentReference)Bibliography

does not match

(GlobalReference)https://example.com/aas/1/1/1234859590, (FragmentReference)Specification, (FragmentReference)Bibliographie

Examples for matching model references:

Although these two model references would match according to the matching rules, other rules are violated, i.e. that the ID of the submodel is unique. If the ID of a submodel is unique, it is not possible that there are two direct submodel element children with the same name (here: Specification). It is also not possible two different versions of the same submodel are compared here, because we would then assume that the ID also contains the version information (see Clause 5.3.2.2). The matching algorithm would still identify these two model references as matching although one of them is buggy.

(Submodel)https://example.com/aas/1/1/1234859590, (File)Specification

(Submodel)https://example.com/aas/1/1/1234859590, (Blob)Specification

Examples for matching model and external references:

(Submodel)https://example.com/aas/1/1/1234859590

is identical to

(GlobalReference)https://example.com/aas/1/1/1234859590

(Submodel)https://example.com/aas/1/1/1234859590, (File)Specification (FragmentReference)Bibliography

is identical to

(GlobalReference)https://example.com/aas/1/1/1234859590, (FragmentReference)Specification, (FragmentReference)Bibliography

4.5. Submodel Instances and Templates

4.5.1. Can New or Proprietary Submodels be Formed?

It is in the interest of Industry 4.0 for as many submodels as possible, including free and proprietary submodels, to be formed (see [4], "Free property sets"). A submodel can be formed at any time for a specific Administration Shell of an asset. The provider of the Administration Shell can form in-house identifiers for the type and instance of the submodel in line with Clause 4.3. All I4.0 systems are called on to ignore submodels and properties that are not individually known. Hence, it is always possible to deposit proprietary – e.g. manufacturer-specific or user-specific – information, submodels, or properties in an Administration Shell.

> Note: it is the intention of the Administration Shell to include proprietary information, e.g. to link to company-wide identification schemes or information required for company-wide data processing. This way, a single infrastructure can be used to transport standardized and proprietary information at the same time. New information elements can also be conveyed and introduced (and standardized at a later stage).

4.5.2. Creating a Submodel Instance Based on an Existing Submodel Template

A public specification of a submodel template (e.g. via publication by Plattform Industrie 4.0) should be available to instantiate an existing submodel template. In special cases, a submodel can also be instantiated from a non-public submodel template, such as a manufacturer specification.

In November 2020, the first two submodel templates for the Asset Administration Shell were published, one for a nameplate [40] and one for generic technical data [39]. Others followed and will follow. Please see [45] for an overview of registered submodel templates.

The identifiers of concept definitions to be used as semantic references are already predefined in each submodel template. An instantiation of such a submodel merely requires the creation of properties with a semantic reference to the property definition and an attached value. The same applies to other subtypes of submodel elements.

The only thing that cannot be defined in the template itself is the unique ID of the submodel instance itself (it is not identical to the ID of the submodel template), as well as the property values, etc. Templates also define cardinalities, for example whether an element is optional or not. Submodel element lists typically contain more than one element: the template contains an exemplary element template; the other elements can be created by copy/paste from this template.

4.6. Events

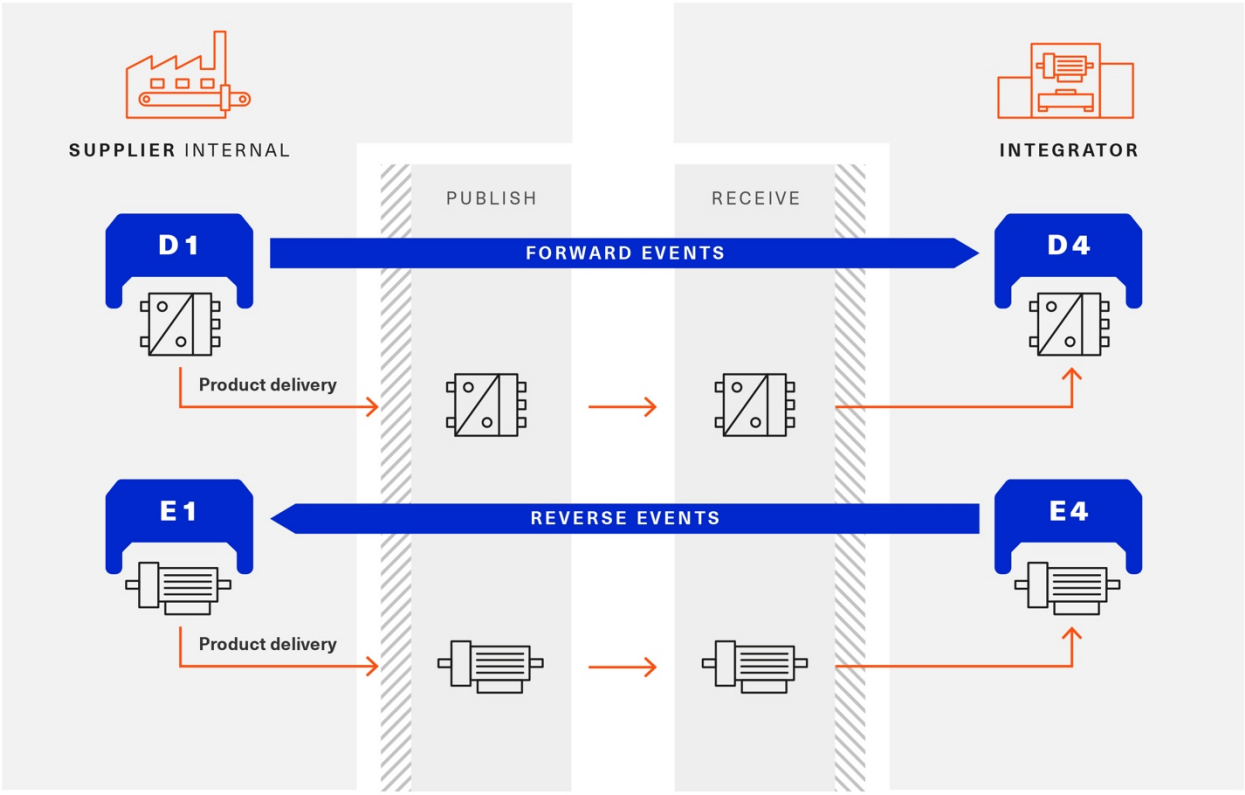
4.6.1. Overview

Events are a very versatile mechanism of the Asset Administration Shell. The following subclauses describe some use cases for events. They summarize different types of events to depict requirements, introduce a *SubmodelElement* “*Event*" to enable declaration of events of an Asset Administration Shell. Further, the general format of event messages is specified.

> Note: the concept of event is still in the experimental phase. Please be aware that backward compatibility cannot be ensured for future versions of the metamodel.

4.6.2. Brief Use Cases for Events Used in Asset Administration Shells

- Event use cases are briefly outlined in the following (see also):



- An integrator has purchased a device. Later in time, the supplier of the device provides a new firmware. The integrator wants to detect the offer of a new firmware and wants to update the firmware after evaluating its suitability ("forward events"). A dependent Asset Administration Shell ("D4") detects events from a parent or type Asset Administration Shell ("D1"), which is described by the *derivedFrom* relation. An illustration of the use case is given in Figure 7.

- An integrator/operator operates a motor purchased from a supplier. During operation, condition monitoring incidents occur. Both parties agree on a business model providing availability. The supplier wants to monitor device statuses which are located further in the value chain ("reverse events"). An illustration of the use case is given in Figure 7.

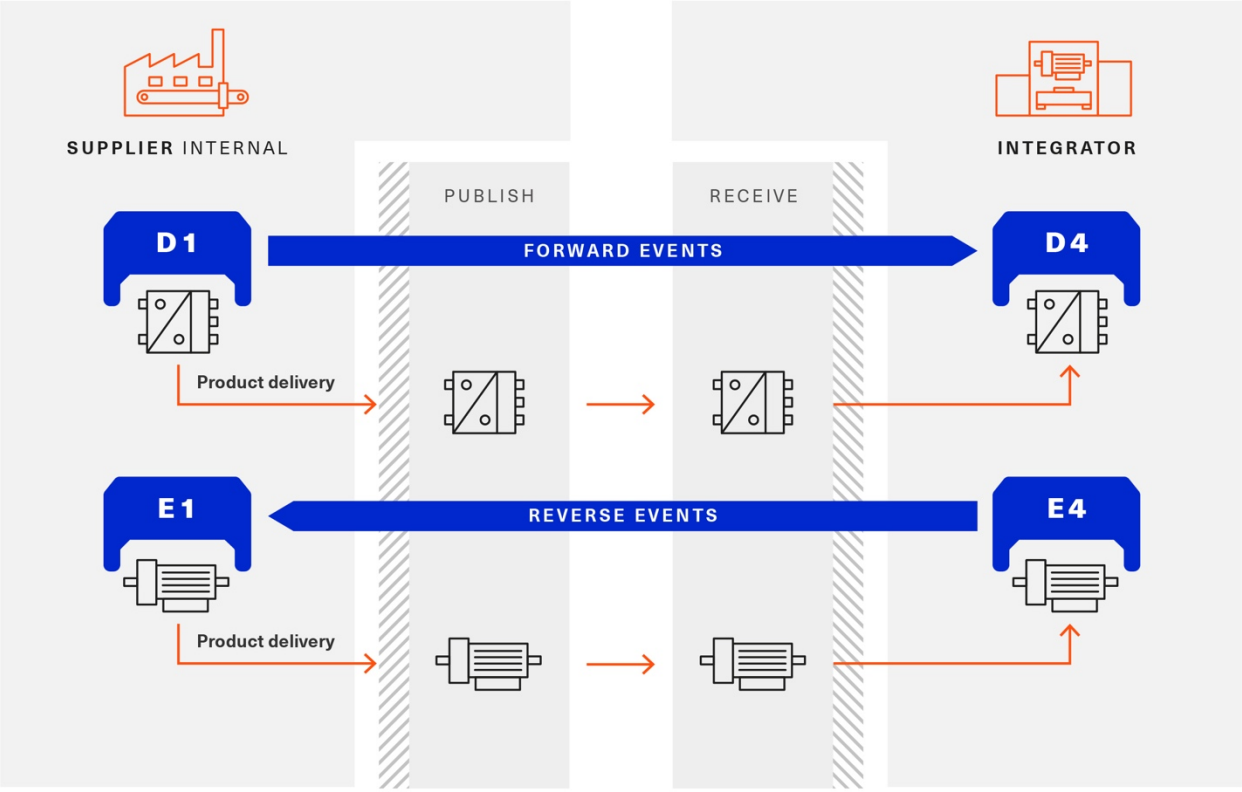


Figure 7. Forward and Reverse Events

- An operator is operating a certain I4.0 component over time. Changes that occasionally occur to these I4.0 components from different systems shall be tracked for documentation and auditing purposes. This can be achieved by recording events over time. An illustration of the use case is given in Figure 8.

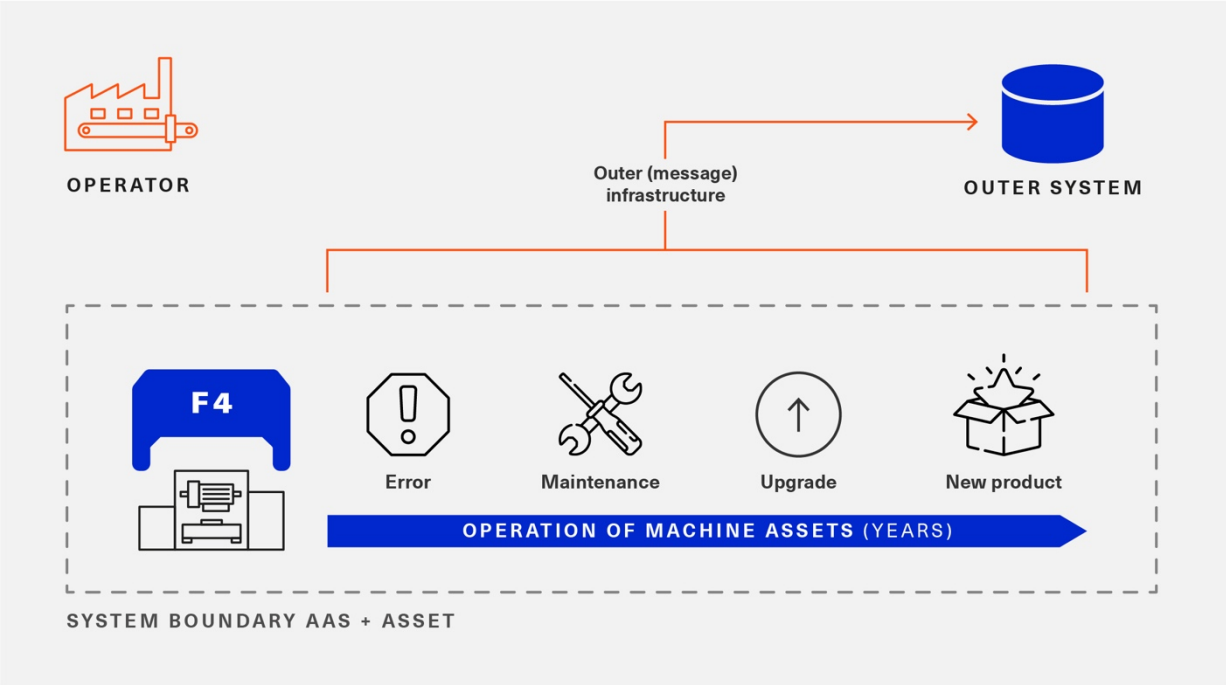


Figure 8. Tracking of Changes via Events

- An operator is operating different I4.0 components, which are deployed to manufacturer clouds. The operator wants to integrate data from these components, according to DIN SPEC 92222. For this purpose, information needs to be forwarded to the operator cloud ("value push"). An illustration of the use case is given in Figure 9.

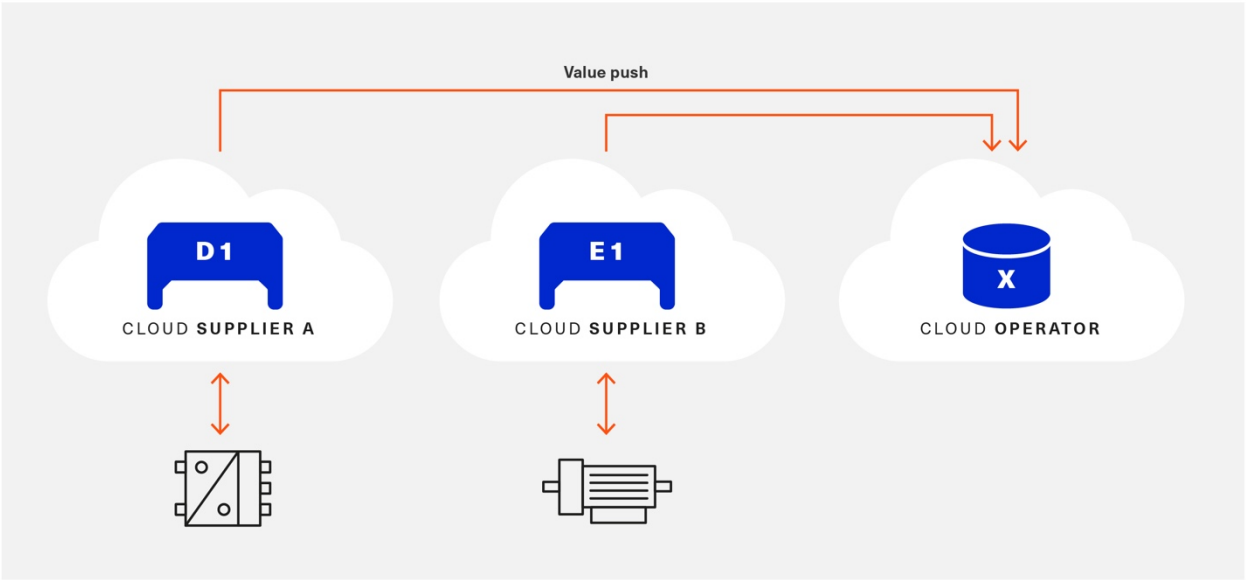


Figure 9. Value Push Events Across Clouds

4.6.3. Input and Output Directions of Events

We can distinguish between incoming and outgoing events. See Table 3 for more information on the event directions.

Table 3. Directions of Events

Direction	Description
Out	The event is monitoring the *Referable* it is attached to. An outer message infrastructure, e.g. by OPC UA, MQTT or AMQP, will transport these events to other Asset Administration Shells, further outer systems and users.
In	The software entity, which implements the respective *Referable*, can handle incoming events. These incoming events will be delivered by an outer message infrastructure, e.g. OPC UA, MQTT or AMQP, to the software entity of the *Referable*.

4.6.4. Types of Events

The uses cases described in Clause 4.6.2 need different types of events. Each event type is identified by a *semanticId* and features a specialized payload.

Table 4 gives an overview of types of events. The possible directions of an event are described in Clause 4.6.3.

Table 4. Types of Events

Group	Direction	Motivation / Conditions
Structural changes of the Asset Administration Shell	Out	• CRUD[6] of Submodels, Assets, SubmodelElements, etc.
	In	• Detect updates on parent/type/*derivedFrom* Asset Administration Shell
Updates of properties and dependent attribute	Out	• update of values of SubmodelElements • time-stamped updates and time series updates • explicit triggering of an update event
Operation element of Asset Administration Shell	Out	• monitoring of (long-lasting) execution of *OperationElement* and updating events during execution
Monitoring, conditional, calculated events	Out	• e.g. when voiding some limits (e.g. stated by Qualifiers with expression semantics)
Infrastructure events	Out	• Booting, shutdown, out of memory, etc. of software entity of respective Referable (Asset Administration Shell, Submodel)
Repository events	In/ Out	• Change of semantics of IRDIs (associated concept definition)

Group	Direction	Motivation / Conditions
Security events	Out	• logging events • access violations, unfitting roles and rights, denial of service, etc.
Alarms and events	Out	• alarms and events management analog to distributed control systems (DCS)

Custom Event Types

Custom event types can be defined by using a proprietary, but worldwide unique, semanticId for this event type. Such customized events can be sent or received by the software entity of the respective referable, based on arbitrary conditions, triggers, or behavior. While the general format of the event messages needs to comply with this specification, the payload might be completely customized.

Event Scopes

Events can be stated with an *observableReference* to the *Referables* of Asset Administration Shell, *Submodel*s, and *SubmodelElements.* These *Referables* define the scope of the events, which are to be received or sent.
Table 5 describes the different scopes of an event.

Table 5. Event Scopes

Event attached to …	Scope
AssetAdministrationShell	This event monitors/represents all logical elements of an Administration Shell, such as *AssetAdministrationShell, AssetInformation, Submodels*.
Submodel	This event monitors/represents all logical elements of the respective *Submodel* and all logical dependents.
SubmodelElementList and SubmodelElementCollection and Entity	This event monitors/represents all logical elements of the respective *SubmodelElementCollection, SubmodelElementList or Entity* and all logical dependents (value or statement resp.).
SubmodelElement (others)	This event monitors/represents a single atomic *SubmodelElement*, e.g. a data element which might include the contents of a *Blob* or *File*.

[1] Such additional asset identifiers are contained in *AssetInformation/specificAssetIds*.

[2] Note: in version V1.0 of this specification, idShort was optional for identifiables. This changed in V2.0: idShort was set to mandatory for all referables. With V3.0, idShort was again made optional.

[3] Semantic mapping files are also used in ECLASS between ECLASS Classic and ECLASS Advanced: https://eclass.eu/support/technical-specification/data-model/basic-advanced-mapping

[4] This is the format used for semantic mapping in ECLASS: https://eclass.eu/fileadmin/Redaktion/pdf-

Dateien/Wiki/ECLASSXML_3.0/ECLASS_XML/mapping.xsd

[5] The example only contains arbitrary IRDIs and does not represent a real-world example.

Chapter 5. The Information Metamodel of the Asset Administration Shell (normative)

5.1. Introduction

This clause specifies the information metamodel of the Asset Administration Shell.

An overview of the metamodel of the Asset Administration Shell is given in Subclause 5.1; Subclause 5.3 describes the classes and all their attributes in detail.

The legend of the UML diagrams and the table specification of the classes is explained in Annex D and Annex E. Readers familiar with UML may skip the first clause in Annex E. It is however recommended to have a look at the specifics used in this modelling, especially those on dealing with model references.

> Note: an xmi representation of the UML model can be found in the repository "aas-specs" in the github project admin-shell-io **[51]**: https://github.com/admin-shell-io/aas-specs/tree/master/schemas/xmi

5.2. Overview Metamodel of the Asset Administration Shell

This clause gives an overview of the main classes of the Asset Administration Shell (AAS) metamodel.

Figure 10 shows the main classes to describe a single Asset Administration Shell.

An Asset Administration Shell represents exactly one asset (*AssetAdministrationShell/assetInformation*). Type assets and instance assets are distinguished by the attribute "*AssetInformation/assetKind*". See Clause 5.3.4 for details.

> Note: the UML modelling uses so-called abstract classes for denoting reused concepts like "HasSemantics", "Qualifiable" etc.

In case of an Asset Administration Shell of an instance asset, a reference to the Asset Administration Shell representing the corresponding type asset or another instance asset it was derived from may be added (*AssetAdministrationShell/derivedFrom*). The same holds true for the Asset Administration Shell of an type asset. Types can also be derived from other types.

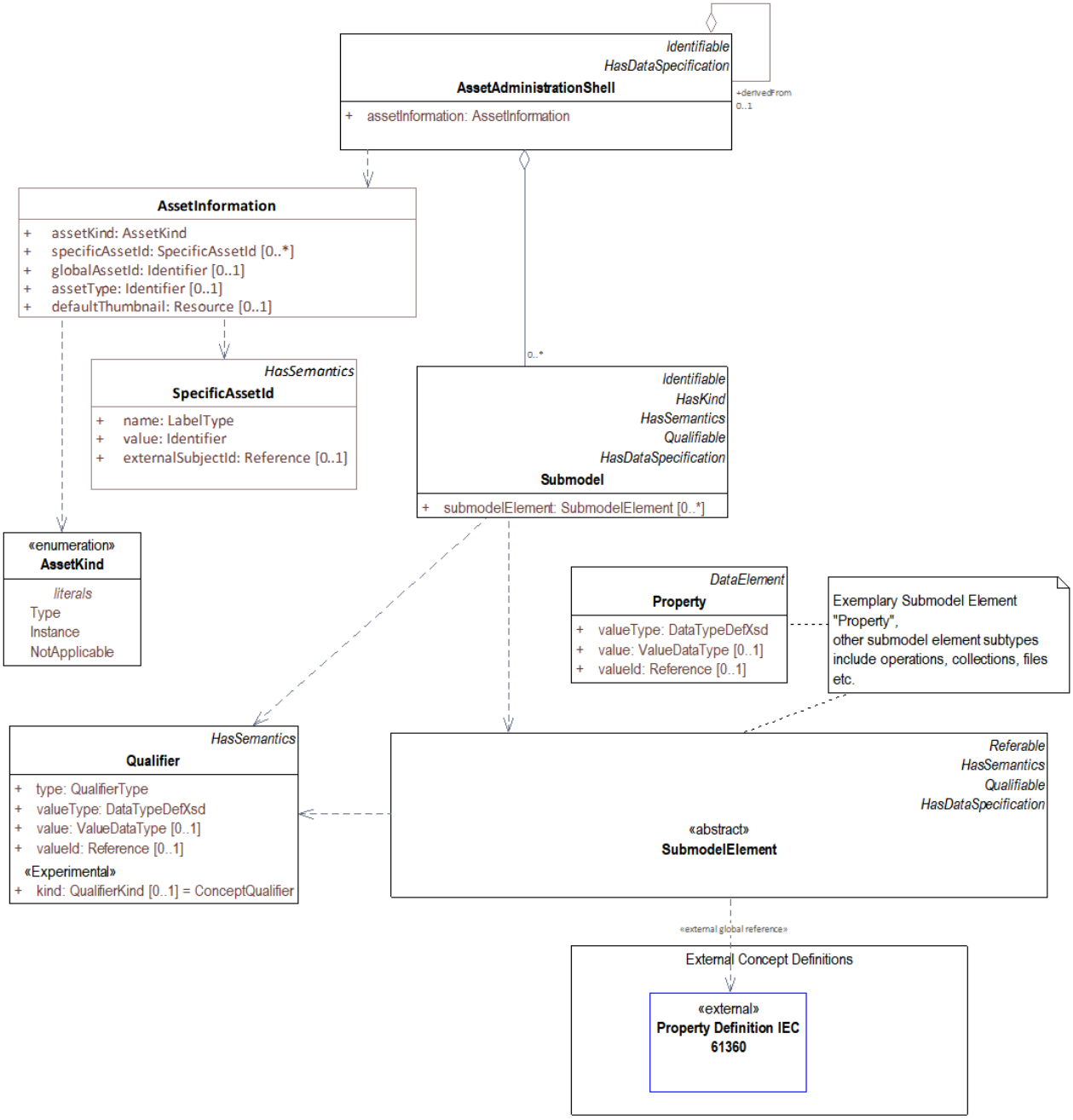


Figure 10. Overview Metamodel of the Asset Administration Shell

An asset may typically be represented by several different identification properties like the serial number, the manufacturer part ID or the different customer part IDs, its RFID code, etc. Such external identifiers are defined as specific asset IDs, each characterized by a user-defined name, a value, and the user domain (tenant, subject in Attribute Based Access Control; *AssetInformation/specificAssetId*). See Clause 5.3.4 for details. Additionally, a global asset identifier should be assigned to the asset (*AssetInformation/globalAssetId*) in the production and operation phase.

Asset Administration Shells, submodels and concept descriptions need to be globally uniquely identifiable (*Identifiable*). Other elements like properties only need to be referable within the model and thus only require a local identifier (*idShort* from *Referable*). For details on identification, see Clause 4.3; details on *Identifiable* and *Referable* are provided in Clause 5.3.2.7 and Clause 5.3.2.10.

*Submodel*s consist of a set of submodel elements. Submodel elements may be qualified by a so-called *Qualifier*. There might be more than one qualifier per *Qualifiable*. See Clause 5.3.2.8 and Clause 5.3.2.9 for details.

There are different subtypes of submodel elements like properties, operations, lists, etc. See Clause 5.3.7 for details. A typical submodel element is shown in the overview figure: a property is a data element that has a value of simple type like string, date, etc. Every data element is a submodel element (not visible in the figure but implicitly the case, since *DataElement* is inheriting from *SubmodelElement*). For details on properties, see Clause 5.3.7.12.

Every submodel element needs a semantic definition *(semanticId* in *HasSemantics*) to have a well-defined meaning. The submodel element might either refer directly to a corresponding semantic definition provided by an external reference (e.g. to an ECLASS or IEC CDD property definition) or it may indirectly reference a concept description (*ConceptDescription*). See Clause 4.4.1 for matching strategies, and Clause 5.3.2.6 for details.

A concept description may be derived from another property definition of an external standard or another concept description (*ConceptDescription/isCaseOf*). *isCaseOf* is a more formal definition of *sourceOfDefinition*, which is just text.

> Note: in this case, most of the attributes are redundant because they are defined in the external standard. Attributes for information like *preferredName*, *unit* etc. are added to increase usability. Consistency w.r.t. the referenced submodel element definitions should be ensured by corresponding tooling.

If a concept description is not just a copy or refinement of an external standard, the provider of the Asset Administration Shell using this concept description shall be aware that an interoperability with other Asset Administration Shells cannot be ensured.

Data specification templates (*DataSpecification*) can be used to define a named set of additional attributes (besides those predefined by the metamodel) for an element. The data specification template following IEC 61360 is typically used for the concept description of properties, providing e.g. an attribute "preferredName". The <<*template*>> dependency is used to denote recommended data specification templates. See Clause 5.3.2.2 for details.

Data specification templates like the template for IEC 61360 property definitions (Part 3a) are explicitly predefined and recommended to be used by Plattform Industrie 4.0. See Clause 6 for details. If proprietary templates are used, interoperability with other Asset Administration Shells cannot be ensured.

Besides submodel elements including properties and concept descriptions, other identifiable elements may also use additional templates (*HasDataSpecification*). Data specification templates are selected at design time. Further details are provided in Clause 6.

Figure 12 gives a complete overview of all elements defined in the metamodel and specified in Clause 5.3. The UML packages reflect the structure of Clause 5.3. The elements of package "Core" are specified as first class citizens in Clause 5.3, except for their imported packages: the elements of package "SubmodelElements" are specified in Clause 5.3.6. Elements of package "Common" are specified in Clause 5.3.2. The elements of package "Reference" are specified in Clause 5.3.10. Elements from package "Types" are specified in Clause 5.3.11. The only package that is not listed is "Data Specifications (Templates)" because data specifications are handled differently. Data specification templates are explained in Clause 6.

Note: the abstract classes are numbered h0_, h1_, etc. (e.g. h1_Referable); their aliases however are defined without this prefix. The reason for this naming is that no order for inherited classes can be defined in the tooling used for UML modelling (Enterprise Architect), since they are ordered alphabetically. The order is important for some serializations (e.g. for XML).

Figure 11 shows the so-called environment. The environment's purpose is to list all Asset Administration Shells, all submodels, and all concept descriptions – in other word, all identifiables within an ecosystem.

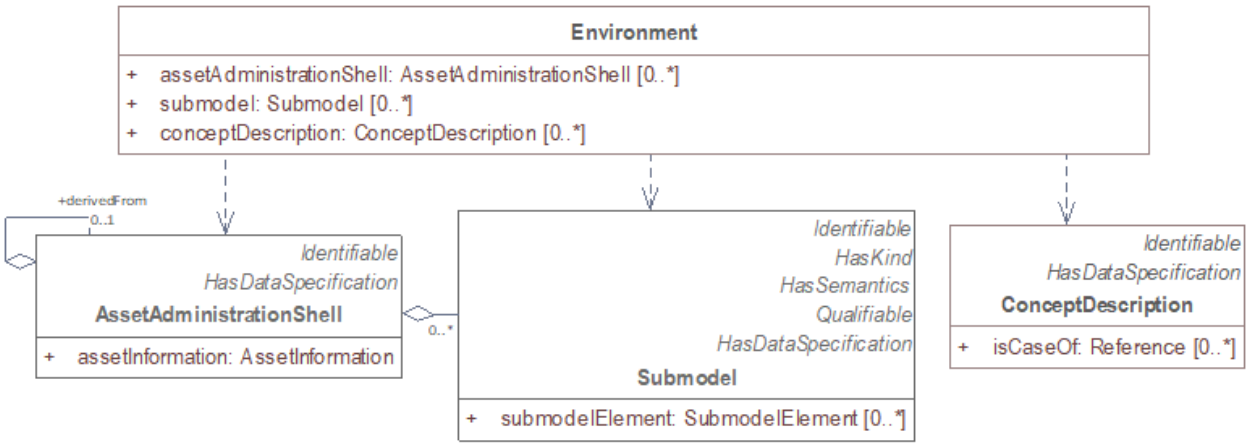


Figure 11. Metamodel of Environment

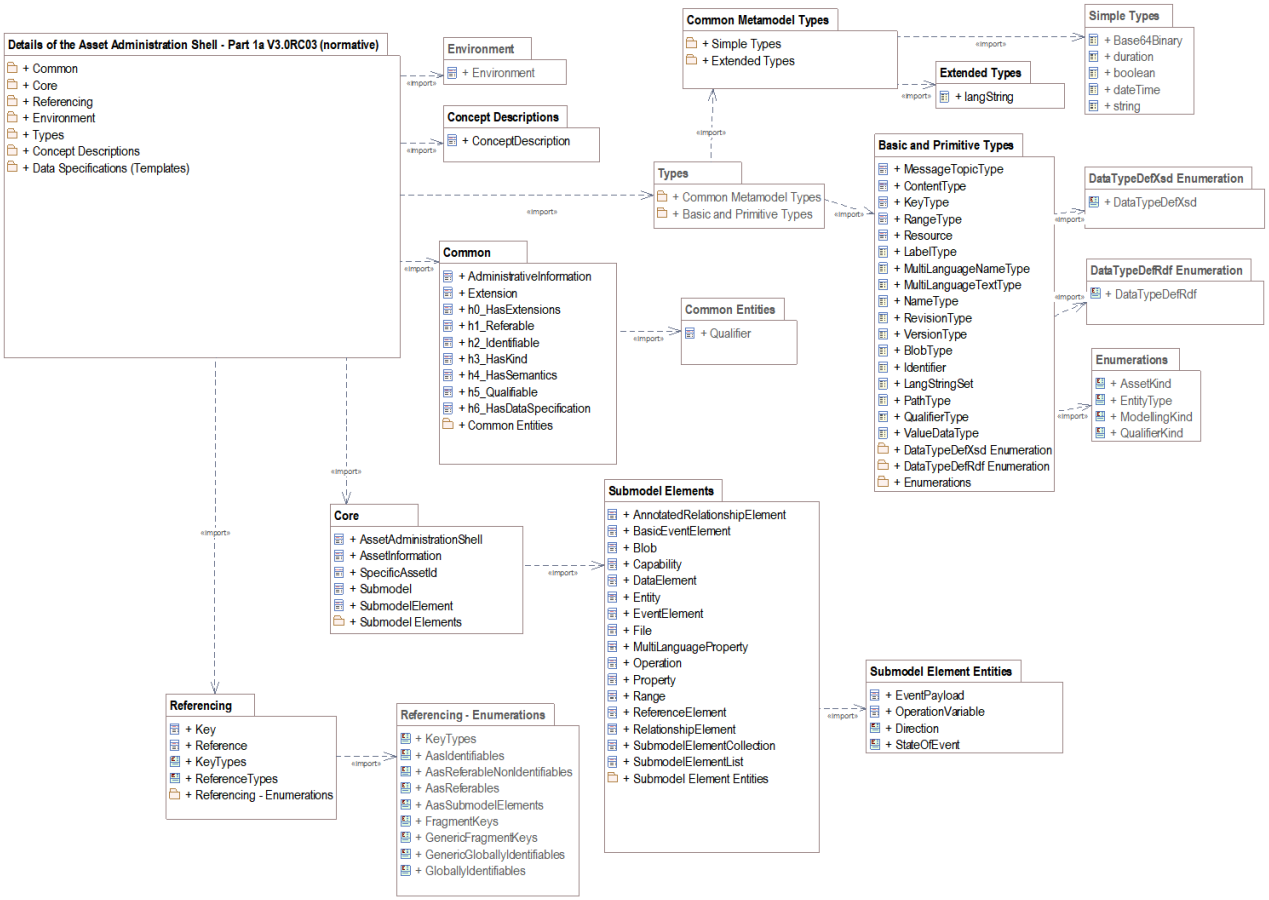


Figure 12. Metamodel Package Overview

5.3. Metamodel Specification Details: Designators

5.3.1. General

This clause specifies the classes of the metamodel in detail. An overview is provided in Clause 5.2. Annex E explains UML modelling together with the specifics used in this specification. Annex D depicts the templates used to describe the classes and relationships. Annex G shows some of the diagrams together with all their inherited attributes to give a complete overview.

To understand the specifications, it is crucial to understand the common attributes first (Clause 5.3.2). They are reused throughout the specifications of the other classes ("inherits from") and define important concepts like identifiable, qualifiable, etc. They are abstract, i.e. there is no object instance of such classes.

The concept of referencing and how a reference is represented in the UML diagrams and the tables is explained in Clause 5.3.9 and Annex E.

Constraints that are no invariants of classes are specified in Clause 5.3.11.3.

5.3.2. Common Attributes

General

This clause specifies the abstract classes that represent commonly used attributes and terminology, together with the classes and data types exclusively used in these classes. They are represented in alphabetical order.

Administrative Information Attributes

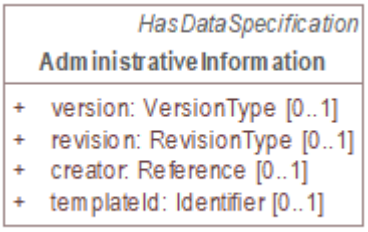


Figure 13. Metamodel of Administrative Information

Every identifiable may contain administrative information. Administrative information includes, for example,

- information about the version of the element,
- information about who created or who made the last change to the element,
- information about the languages available in case the element contains text; the master or default language may also be defined for translating purposes,
- information about the submodel template that guides the creation of the submodel

In principle, the version corresponds to the *version_identifier* according to IEC 62832. However, it is not used for concept identifiers only (IEC TS 62832-1), but for all identifiable elements. Together, version and revision correspond to the version number according to IEC 62832.

Other attributes of the administrative information like creator refer to ISO 15836-1:2017, the Dublin Core metadata element set.

For more information on the concept of subject, see Attribute Based Access Control (ABAC) [49]. The assumption is that every subject has a unique identifier.

AdministrativeInformation allows the usage of templates (*HasDataSpecification*). Data specifications are defined in separate documents.

> Note 1: two submodels with the same semanticId but different administrative information shall have different IDs (*Submodel/id*), since they denote that the submodel is not backward compatible or has some other major administrative changes. The *idShort* typically does not change. The same applies to

other identifiables (*Identifiable/id*). Otherwise, the ID of a submodel would not be sufficient to identify the data or service provided by the submodel.

Note 2: since submodels with different versions shall have different identifiers, it is possible that an Asset Administration Shell has two submodels with the same *semanticId* but different versions, i.e. different administrative metainformation.

Note 3: some of the administrative information like the version number might need to be part of the identification. This is similar to the handling of identifiers for concept descriptions using IRDIs. In ECLASS, the IRDI 0173-1#02-AO677#002 contains the version information #002.

Note 4: the process of versioning or adding other administrative information to elements is done by external version or configuration management software and not by the Asset Administration Shell itself.

Class:	AdministrativeInformation
Explanation:	Administrative metainformation for an element like version information Constraint AASd-005: If *AdministrativeInformation*/version is not specified, *AdministrativeInformation/revision* shall also be unspecified. This means that a revision requires a version. If there is no version, there is no revision. Revision is optional.
Inherits from:	HasDataSpecification

Attribute	Explanation	Type	Card.
version	Version of the element	VersionType	0..1
revision	Revision of the element	RevisionType	0..1
creator	The subject ID of the subject responsible for making the element	Reference	0..1

Attribute	Explanation	Type	Card.
templateId	Identifier of the template that guided the creation of the element Note 1: in case of a submodel, the template ID is the identifier of the submodel template that guided the creation of the submodel. Note 2: the submodel template ID is not relevant for validation. Here, the *Submodel/semanticId* shall be used. Note 3: usage of the template ID is not restricted to submodel instances. The creation of submodel templates can also be guided by another submodel template.	Identifier	0..1

Has Data Specification Attributes

Figure 14. Metamodel of HasDataSpecification

Class:	HasDataSpecification <<abstract>>
Explanation:	Element that can be extended by using data specification templates. A data specification template defines a named set of additional attributes an element may or shall have. The data specifications used are explicitly specified with their global ID.
Inherits from:	—

Attribute	Explanation	Type	Card.
dataSpecification	External reference to the data specification template used by the element Note: this is an external reference.	Reference	0..*

For more details on data specifications, please see Clause 6.

Extensions Attributes

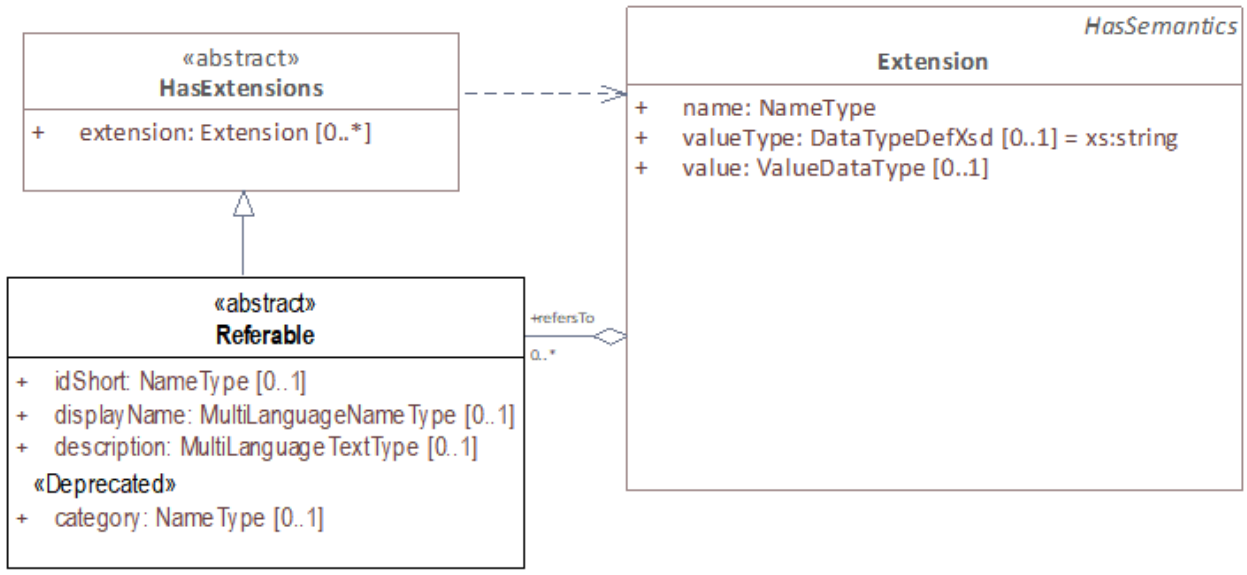


Figure 15. Metamodel of Has Extensions

Class:	HasExtensions <<abstract>>
Explanation:	Element that can be extended by proprietary extensions Note 1: see Clause 5.3.12.4 for constraints related to extensions. Note 2: extensions are proprietary, i.e. they do not support global interoperability.
Inherits from:	-

Attribute	Explanation	Type	Card.
extension	An extension of the element.	Extension	0..*

Class:	Extension
Explanation:	Single extension of an element
Inherits from:	HasSemantics

Attribute	Explanation	Type	Card.
name	Name of the extension	NameType	1
valueType	Data type of the value attribute of the extension Default: xs:string	DataTypeDefXsd	0..1
value	Value of the extension	ValueDataType	0..1
refersTo	Reference to an element the extension refers to	ModelReference<Referable>	0..*

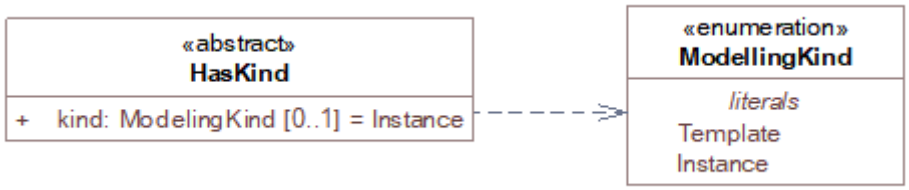


Figure 16. Metamodel of HasKind

Has Kind Attributes

Class:	HasKind
Explanation:	An element with a kind is an element that can either represent a template or an instance. Default for an element is that it is representing an instance.
Inherits from:	—

Attribute	Explanation	Type	Card.
kind	Kind of the element: either type or instance Default Value = *Instance*	ModellingKind	0..1

The kind enumeration is used to denote whether an element is of kind *Template* or *Instance*. It is used to distinguish between submodels and submodel templates.

Enumeration:	ModellingKind
Explanation:	Enumeration for denoting whether an element is a template or an instance
Set of:	—

Literal	Explanation
Template	specification of the common features of a structured element in sufficient detail that such a instance can be instantiated using it
Instance	concrete, clearly identifiable element instance. Its creation and validation may be guided by a corresponding element template.

Has Semantics Attributes

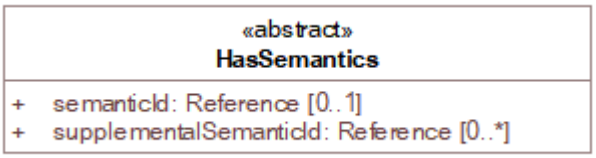


Figure 17. Metamodel of Semantic References (HasSemantics)

For matching algorithm, see Clause 4.4.1.

Class:	HasSemantics <<abstract>>
Explanation:	Element that can have a semantic definition plus some supplemental semantic definitions Constraint AASd-118: If a supplemental semantic ID (*HasSemantics/supplementalSemanticId*) is defined, there shall also be a main semantic ID (*HasSemantics/semanticId*).
Inherits from:	—

Attribute	Explanation	Type	Card.
semanticId	Identifier of the semantic definition of the element called semantic ID or also main semantic ID of the element Note: it is recommended to use an external reference.	Reference	0..1
supplementalSemanticId	Identifier of a supplemental semantic definition of the element called supplemental semantic ID of the element Note: it is recommended to use an external reference.	Reference	0..*

Identifiable Attributes

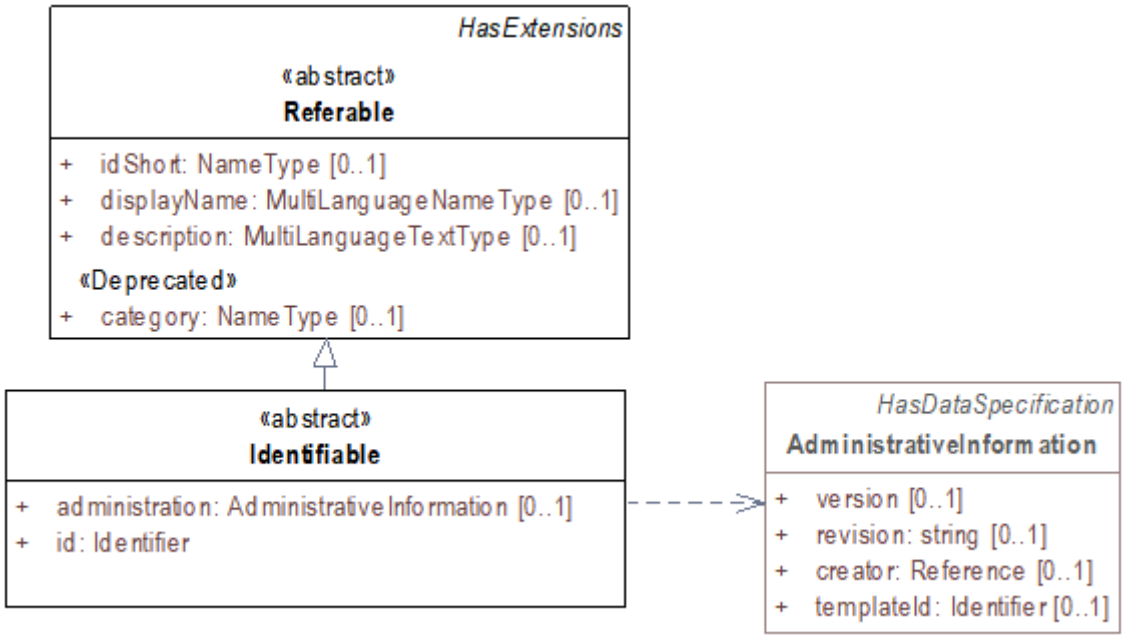


Figure 18. Metamodel of Identifiables

An identifiable element is a referable with a globally unique identifier (*Identifier*). Only the global ID (*Identifiable/id*) shall be used to reference an identifiable, because the *idShort* is not unique for an identifiable. Identifiables may have administrative information like version, etc.

Non-identifiable referable elements can be referenced. However, this requires the context of the element. A referable that is not identifiable has a short identifier (*idShort*) that is unique just in its context, its name

space.

Information about identification can be found in Clause 4.3. See Clause 4.3.4 for constraints and recommendations on when to use which type of identifier.

See Clause 4.3.4 for information about which identifier types are supported.

Class:	Identifiable <<abstract>>
Explanation:	An element that has a globally unique identifier Note: see Clause 5.3.12.2 for constraints related to identifiables.
Inherits from:	Referable

Attribute	Explanation	Type	Card.
administration	Administrative information of an identifiable element Note: some of the administrative information like the version number might need to be part of the identification.	AdministrativeInformation	0..1
id	The globally unique identification of the element	Identifier	1

Qualifiable Attributes

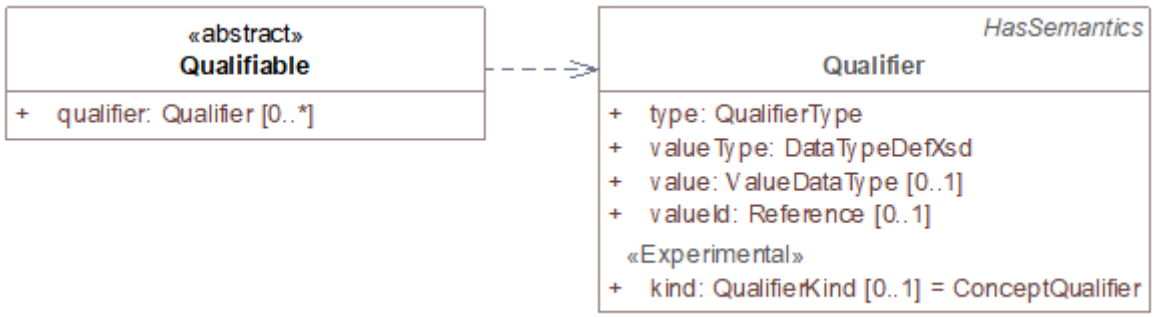


Figure 19. Metamodel of Qualifiables

Class:	Qualifiable <<abstract>>

Explanation:	A qualifiable element may be further qualified by one or more qualifiers. Note: see Clause 5.3.12.3 for constraints related to qualifiables.
Inherits from:	—

Attribute	Explanation	Type	Card.
qualifier	Additional qualification of a qualifiable element	Qualifier	0..*

Qualifier Attributes

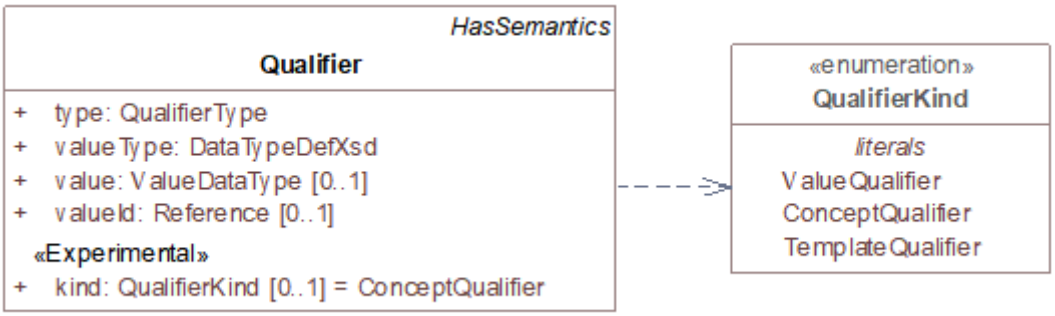


Figure 20. Metamodel of Qualifiers

Qualifiers may be defined for qualifiable elements.

There are standardized qualifiers defined in IEC CDD, IEC61360-4 – IEC/SC 3D. A level qualifier defining the level type minimal, maximal, typical, and nominal value is specified in IEC 62569-1. In DIN SPEC 92000, qualifier types like e.g. expression semantics and expression logic are defined.

Class:	Qualifier

Explanation:	A qualifier is essentially a type-value-pair. Depending on the kind of qualifier, it makes additional statements • w.r.t. the value of the qualified element, • w.r.t the concept, i.e. semantic definition of the qualified element, • w.r.t. existence and other meta information of the qualified element type. Constraint AASd-006: If both, the *value* and the *valueId* of a *Qualifier* are present, the value needs to be identical to the value of the referenced coded value in *Qualifier/valueId*. Constraint AASd-020: The value of *Qualifier/value* shall be consistent with the data type as defined in *Qualifier/valueType*.
Inherits from:	HasSemantics

Attribute	Explanation	Type	Card.
Kind <<Experimental>>	The qualifier kind describes the kind of qualifier that is applied to the element. Default: ConceptQualifier	QualifierKind	0..1
type	The qualifier type describes the type of qualifier that is applied to the element.	QualifierType	1
valueType	Data type of the qualifier *value*	DataTypeDefXsd	1
value	The qualifier value is the value of the qualifier.	ValueDataType	0..1
valueId	Reference to the global unique ID of a coded value Note: it is recommended to use an external reference.	Reference	0..1

It is recommended to add a *semanticId* for the concept of the *Qualifier*. *Qualifier/type* is the preferred name of the concept of the qualifier or its short name.

Enumeration:	QualifierKind
Explanation:	Enumeration for kinds of qualifiers

Set of:	—

Literal	Explanation
ValueQualifier	Qualifies the value of the element; the corresponding qualifier value can change over time. Value qualifiers are only applicable to elements with *kind="Instance".*
ConceptQualifier	Qualifies the semantic definition (*HasSemantics/semanticId*) the element is referring to; the corresponding qualifier value is static.
TemplateQualifier	Qualifies the elements within a specific submodel on concept level; the corresponding qualifier value is static. Note: template qualifiers are only applicable to elements with kind="Template". See constraint AASd-129.

Example of a *ValueQualifier*: property "temperature" and qualifier "value quality" with different qualifier values like "measured", "substitute value".

Example of a *ConceptQualifier:* an Asset Administration Shell with two submodels with different IDs but the same semanticId = "Bill of Material". The qualifier could denote the life cycle with qualifier values like "as planned", "as maintained" etc. (see
Figure 21).

Example of a *TemplateQualifier:* a submodel element with qualifier value "mandatory" or "optional". This qualification is needed to build a correct submodel instance. For more information see [48].

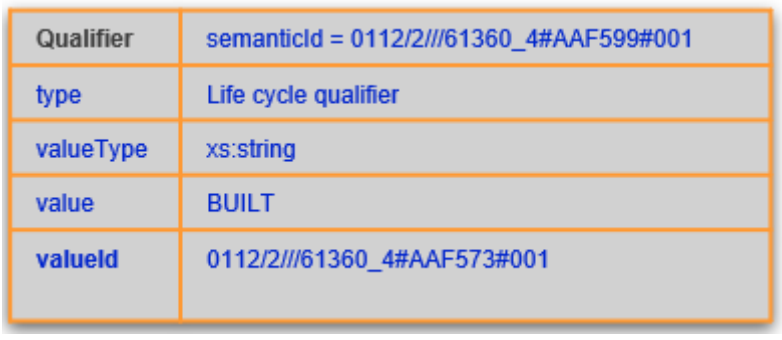


Figure 21. Example: Qualifier from IEC CDD

Referable Attributes

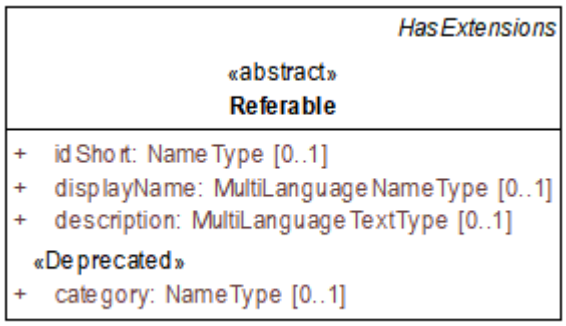


Figure 22. Metamodel of Referables

The metamodel differentiates between elements that are identifiable, referable, or none of both. The latter means they are neither inheriting from *Referable* nor from *Identifiable*, which applies e.g. to *Qualifier*s.

Referable elements can be referenced via the *idShort*. For details on referencing, see Clause 5.3.9.

Not every element of the metamodel is referable. There are elements that are just attributes of a referable.

The *idShort* shall be unique in its name space for non-identifiable referables (see Constraint AASd-022). A name space is defined as follows in this context: the parent element(s), which an element is part of and that is either referable or identifiable, is the name space of the element. Examples: a submodel is the name space for the properties directly contained in it; the name space of a submodel element contained in a submodel element collection is the submodel element collection.

Class:	Referable <<abstract>>
Explanation:	Note1 : an element that is referable by its idShort. This ID is not globally unique. This ID is unique within the name space of the element. Note 2: see Clause 5.3.12.2 for constraints related to referables. Constraint AASd-002: *idShort* of *Referable*s shall only feature letters, digits, underscore ("*"); starting mandatory with a letter, i.e. [a-zA-Z][a-zA-Z0-9*]*.
Inherits from:	HasExtensions

Attribute	Explanation	Type	Card.
category <<Deprecated>>	The category is a value that gives further meta information w.r.t. the class of the element. It affects the expected existence of attributes and the applicability of constraints. Note: The category is not identical to the semantic definition (*HasSemantics*) of an element. The category could e.g. denote that the element is a measurement value, whereas the semantic definition of the element would denote that it is the measured temperature.	NameType	0..1
idShort	In case of identifiables, this attribute is a short name of the element. In case of a referable, this ID is an identifying string of the element within its name space. Note: if the element is a property and the property has a semantic definition (*HasSemantics/semanticId*) conformant to IEC61360, the *idShort* is typically identical to the short name in English, if available.	NameType	0..1
displayName	Display name; can be provided in several languages	MultiLanguageNameType	0..1

Attribute	Explanation	Type	Card.
description	Description or comments on the element The description can be provided in several languages. If no description is defined, the definition of the concept description that defines the semantics of the element is used. Additional information can be provided, e.g. if the element is qualified and which qualifier types can be expected in which context or which additional data specification templates.	MultiLanguageTextType	0..1

Predefined categories are described in Table 6.

> Note: categories are deprecated and should no longer be used.

Table 6. Categories[2] for Elements with Value

Category:	Applicable to, Examples:	Explanation:
CONSTANT	Property ReferenceElement	An element with the category CONSTANT is an element with a value that does not change over time. In ECLASS, this kind of category has the category "Coded Value".
PARAMETER	Property MultiLanguageProperty Range SubmodelElementCollection	An element with the category PARAMETER is an element that is once set and then typically does not change over time. This applies e.g. to configuration parameters.
VARIABLE	Property SubmodelElementList	An element with the category VARIABLE is an element that is calculated during runtime, i.e. its value is a runtime value.

5.3.3. Asset Administration Shell Attributes

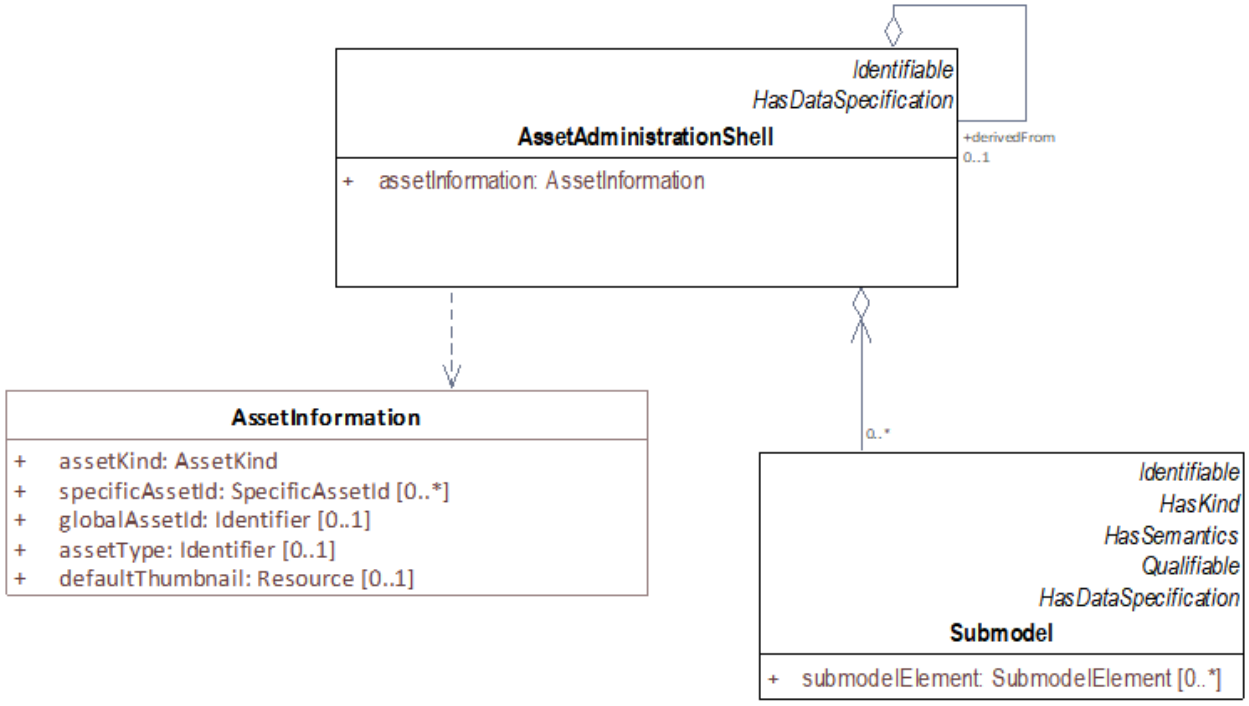


Figure 23. Metamodel of an AssetAdministrationShell

An Administration Shell is uniquely identifiable since it inherits from *Identifiable*.

The *derivedFrom* attribute is used to establish a relationship between two Asset Administration Shells that are derived from each other. For more detailed information on the *derivedFrom* concept, see Clause 4.2.

Class:	AssetAdministrationShell
Explanation:	An Asset Administration Shell
Inherits from:	Identifiable; HasDataSpecification]

Attribute	Explanation	Type	Card.
derivedFrom	The reference to the Asset Administration Shell, which the Asset Administration Shell was derived from	ModelReference<AssetAdministrationShell>	0..1
assetInformation	Meta information about the asset the Asset Administration Shell is representing	AssetInformation	1

Attribute	Explanation	Type	Card.
submodel	Reference to a submodel of the Asset Administration Shell A submodel is a description of an aspect of the asset the Asset Administration Shell is representing. The asset of an Asset Administration Shell is typically described by one or more submodels. Temporarily, no submodel might be assigned to the Asset Administration Shell.	ModelReference<Submodel>	0..*

5.3.4. Asset Information Attributes

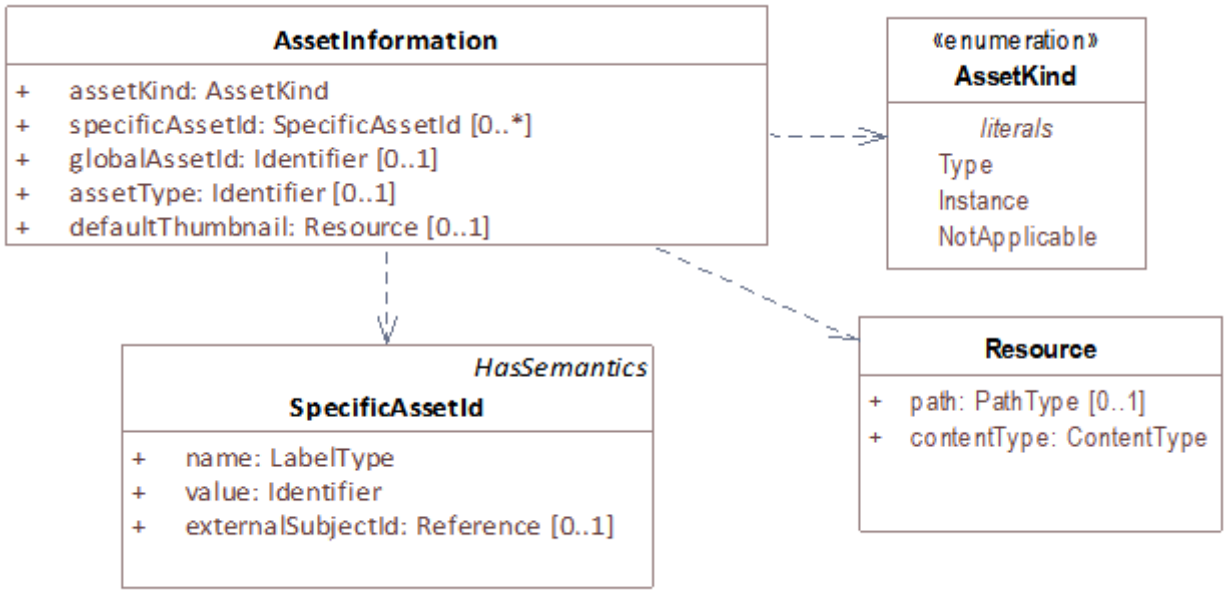


Figure 24. Metamodel of Asset Information

Class:	AssetInformation

Explanation:	In *AssetInformation,* identifying meta data of the asset that is represented by an Asset Administration Shell is defined. The asset may either represent a type asset or an instance asset. The asset has a globally unique identifier, plus – if needed – additional domain-specific (proprietary) identifiers. However, to support the corner case of very first phase of life cycle where a stabilized/constant global asset identifier does not already exist, the corresponding attribute "globalAssetId" is optional. Constraint AASd-131: The *globalAssetId* or at least one *specificAssetId* shall be defined for *AssetInformation*. Note: see Clause 5.3.12.5 for constraints related to asset information.
Inherits from:	—

Attribute	Explanation	Type	Card.
assetKind	Denotes whether the *Asset* is of kind "Type" or "Instance"	AssetKind	1
globalAssetId	Identifier of the asset the Asset Administration Shell is representing This attribute is required as soon as the Asset Administration Shell is exchanged via partners in the life cycle of the asset. In a first phase of the life cycle, the asset might not yet have a global asset ID but already an internal identifier. The internal identifier would be modelled via "specificAssetId".	Identifier	0..1
specificAssetId	Additional domain-specific, typically proprietary identifier for the asset like serial number, manufacturer part ID, customer part IDs, etc	SpecificAssetId	0..*

Attribute	Explanation	Type	Card.
assetType	In case *AssetInformation/assetKind* is applicable the *AssetInformation/assetType* is the asset ID of the type asset of the asset under consideration as identified by *AssetInformation/globalAssetId*. Note: in case *AssetInformation/assetKind* is "Instance" then the AssetInformation/assetType denotes which "Type" the asset is of. But it is also possible to have an *AssetInformation/assetType* of an asset of kind "Type".	Identifier	0..1
defaultThumbnail	Thumbnail of the asset represented by the Asset Administration Shell; used as default.	Resource	0..1

Note: besides this asset information, there still might be an identification submodel with further information. Specific asset IDs mainly serve the purpose of supporting discovery of Asset Administration Shells for an asset.

Class:	Resource
Explanation:	Resource represents an address to a file (a locator). The value is a URI that can represent an absolute or relative path.
Inherits from:	—

Attribute	Explanation	Type	Card.
path	Path and name of the resource (with file extension) The path can be absolute or relative.	PathType	1

Attribute	Explanation	Type	Card.
contentType	Content type of the content of the file The content type states which file extensions the file can have.	ContentType	0..1

Enumeration:	AssetKind
Explanation:	Enumeration for denoting whether an asset is a type asset or an instance asset or whether this kind of classification is not applicable
Set of:	—

Literal	Explanation
Type	Type asset
Instance	Instance asset
NotApplicable	Neither a type asset nor an instance asset

For more information on types and instances, see Clause 4.2.

Class:	SpecificAssetId
Explanation:	A specific asset ID describes a generic supplementary identifying attribute of the asset. The specific asset ID is not necessarily globally unique. Constraint AASd-133: *SpecificAssetId/externalSubjectId* shall be a global reference, i.e. *Reference/type = ExternalReference*.
Inherits from:	HasSemantics

Attribute	Explanation	Type	Card.
name	Name of the asset identifier	LabelType	1
value	The value of the specific asset identifier with the corresponding name	Identifier	1
externalSubjectId	The unique ID of the (external) subject the specific asset ID *value* belongs to or has meaning to Note: this is an external reference.	Reference	0..1

Note 1: names for specificAssetIds do not need to be unique.

Note 2: semanticIds for the single specificAssetIds do not need to be unique.

For more information on the concept of subject, see Attribute Based Access Control (ABAC) [49]. The assumption is that every subject has a unique identifier.

5.3.5. Submodel Attributes

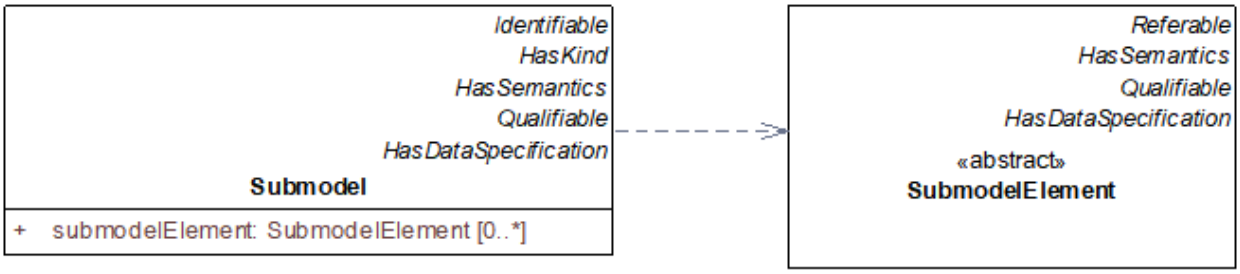


Figure 25. Metamodel of Submodel

Adding a *semanticId* for a submodel is recommended (see Table 2).

If the submodel is of *kind=Template* (modelling kind as inherited by *HasKind*), the submodel elements within the submodel are presenting submodel element templates. If the submodel is of *kind=Instance*, its submodel elements represent submodel element instances.

Note: validators shall handle a submodel like *SubmodelElementCollection/submodelElements* and not like a *SubmodelElementList/value*. The difference is that a submodel is identifiable and a predefined unit of information within the Asset Administration Shell.

Class:	Submodel
Explanation:	A submodel defines a specific aspect of the asset represented by the Asset Administration Shell. A submodel is used to structure the digital representation and technical functionality of an Administration Shell into distinguishable parts. Each submodel refers to a well-defined domain or subject matter. Submodels can become standardized and, in turn, submodel templates.
Inherits from:	Identifiable; HasKind; HasSemantics; Qualifiable; HasDataSpecification

Attribute	Explanation	Type	Card.
submodelElement	A submodel consists of zero or more submodel elements.	SubmodelElement	0..*

5.3.6. Submodel Element Attributes

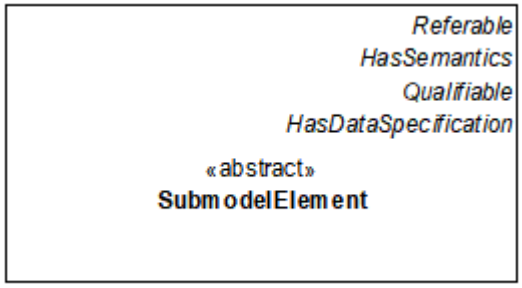


Figure 26. Metamodel of Submodel Element

Submodel elements are qualifiable elements, i.e. one or more qualifiers may be defined for each of them.

It is recommended to add a *semanticId* to a *SubmodelElement.*

Submodel elements may also have defined data specification templates. A template might be defined to mirror some of the attributes like *preferredName* and *unit* of a property concept definition if there is no corresponding concept description available. Otherwise, there is only the property definition referenced by *semanticId* available for the property; the attributes must be looked up online in a different way and are not available offline.

Class:	SubmodelElement <<abstract>>
Explanation:	A submodel element is an element suitable for the description and differentiation of assets.
Inherits from:	Referable; HasSemantics; Qualifiable; HasDataSpecification

Attribute	Explanation	Type	Card.

5.3.7. Overview of Submodel Element Types

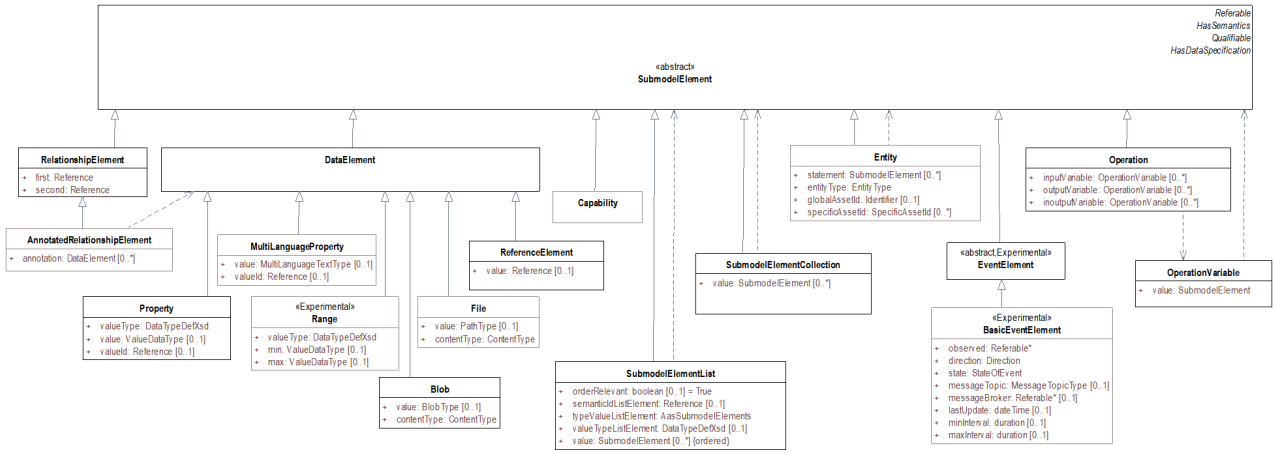


Figure 27. Metamodel Overview for Submodel Element Subtypes

Submodel elements include data properties as well as operations, events and other elements needed to describe a model for an asset (see Figure 27).

General

All submodel elements including abstract classes like data elements are specified in alphabetical order.

Annotated Relationship Element Attributes

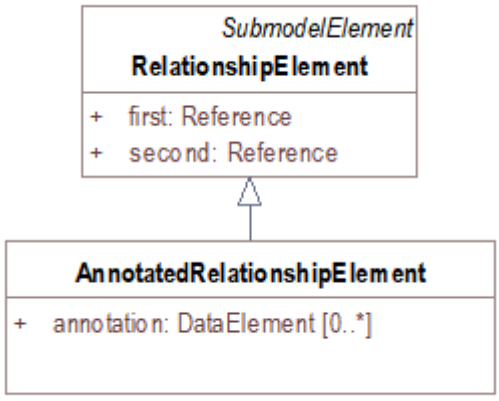


Figure 28. Metamodel of Annotated Relationship Elements

An annotated relationship is a relationship similar to a ternary association in UML. The semantics of the relationship is defined via the *semanticId* of the *RelationshipElement*. If this semantic definition requires additional information not contained in the *first* or *second* object referenced via the relationship, this information needs to be stored as annotation.

Class:	AnnotatedRelationshipElement
Explanation:	An annotated relationship element is a relationship element that can be annotated with additional data elements.

Inherits from:	RelationshipElement

Attribute	Explanation	Type	Card.
annotation	A data element that represents an annotation that holds for the relationship between the two elements	DataElement	0..*

Basic Event Element Attributes

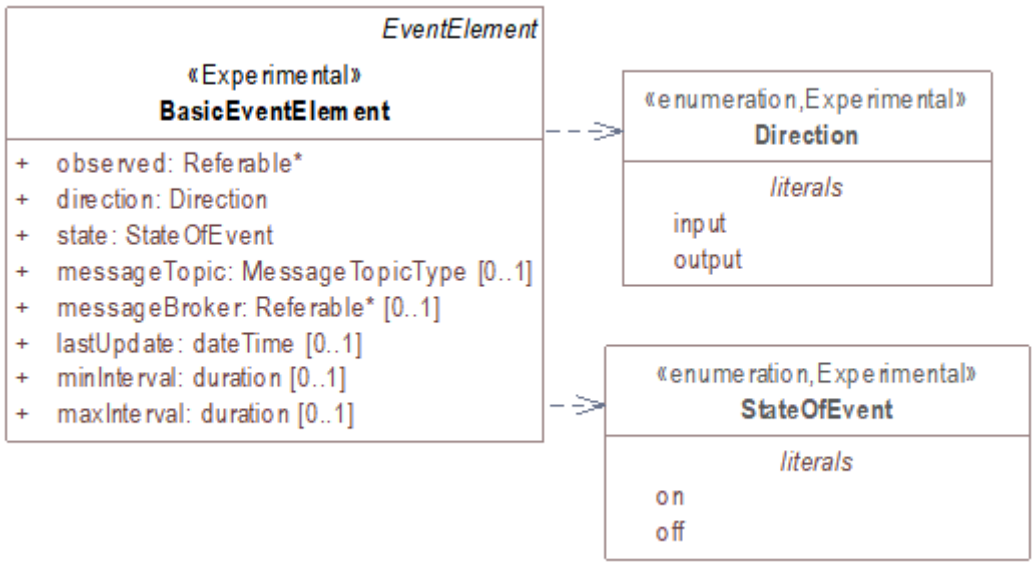


Figure 29. Metamodel of Basic Event Element

Class:	BasicEventElement <<Experimental>>
Explanation:	A basic event element
Inherits from:	EventElement

Attribute	Explanation	Type	Card.
observed	Reference to a referable, e.g. a data element or a submodel that is being observed	ModelReference<Referable>	1
Direction	Direction of event Can be \{ input, output }	Direction	1
State	State of event Can be \{ on, off }	StateOfEvent	1

Attribute	Explanation	Type	Card.
messageTopic	Information for the outer message infrastructure to schedule the event for the respective communication channel.	MessageTopicType	0..1
messageBroker	Information about which outer message infrastructure shall handle messages for the *EventElement*; refers to a *Submodel, SubmodelElementList*, *SubmodelElementCollection* or *Entity*, which contains *DataElement*s describing the proprietary specification for the message broker Note: this proprietary specification could be standardized by using respective submodels for different message infrastructure, e.g. OPC UA, MQTT or AMQP.	ModelReference<Referable>	0..1
lastUpdate	Timestamp in UTC, when the last event was received (input direction) or sent (output direction)	dateTime	0..1
minInterval	For input direction reports on the maximum frequency, the software entity behind the respective referable can handle input events. For output events, the maximum frequency of outputting this event to an outer infrastructure is specified. Might be not specified, i.e. if there is no minimum interval.	duration	0..1

Attribute	Explanation	Type	Card.
maxInterval	Not applicable for input direction For output direction: maximum interval in time, the respective referable shall send an update of the status of the event, even if no other trigger condition for the event was not met. Might not be specified, i.e. if there is no maximum interval.	duration	0..1

Enumeration:	Direction <<Experimental>>
Explanation:	Direction
Set of:	—

Literal	Explanation
input	Input direction
output	Output direction

Enumeration:	StateOfEvent <<Experimental>>
Explanation:	State of an event
Set of:	—

Literal	Explanation
on	Event is on
off	Event is off

Events sent or received by an Asset Administration Shell always comply with a general format. Exception: events exchanged in the course of an Industry 4.0 interaction pattern.

The payload of such an event is specified below.

> Note: the payload is not part of the information model as exchanged via the AASX package format but used in re-active Asset Administration Shells.

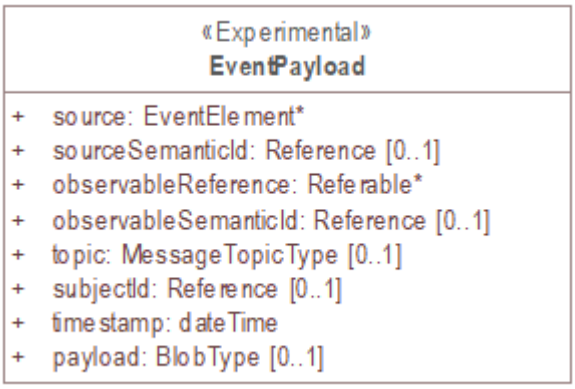


Figure 30. Metamodel of Event Payload

Class:	EventPayload <<Experimental>>
Explanation:	Defines the necessary information of an event instance sent out or received
Inherits from:	-

Attribute	Explanation	Type	Card.
source	Reference to the source event element	ModelReference<EventElement>	1
sourceSemanticId	semanticId of the source event element, if available Note: it is recommended to use an external reference.	Reference	0..1
observableReference	Reference to the referable, which defines the scope of the event.	ModelReference<Referable>	1
observableSemanticId	semanticId of the referable, which defines the scope of the event, if available. Note: it is recommended to use an external reference.	Reference	0..1
topic	Information for the outer message infrastructure to schedule the event for the respective communication channel	MessageTopicType	0..1

Attribute	Explanation	Type	Card.
subjectId	Subject, who/which initiated the creation Note: this is an external reference.	Reference	0..1
timestamp	Timestamp in UTC, when this event was triggered	dateTime	1
payload	Event-specific payload	BlobType	0..1

For more information on the concept of subject, see Attribute Based Access Control (ABAC) [49]. The assumption is that every subject has a unique identifier.

Blob Attributes

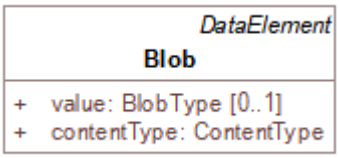


Figure 31. Metamodel of Blobs

For information on content type, see Clause 5.3.7.9 on submodel element "File".

Class:	Blob
Explanation:	A Blob is a data element representing a file that is contained in the value attribute with its source code.
Inherits from:	DataElement

Attribute	Explanation	Type	Card.
value	The value of the blob instance of a blob data element Note: in contrast to the file property, the file content is stored directly as value in the Blob data element.	BlobType	0..1

Attribute	Explanation	Type	Card.
contentType	Content type of the content of the blob. The content type (MIME type) states which file extensions the file can have. Valid values are content types like "application/json", "application/xls", "image/jpg". The allowed values are defined as in RFC2046.	ContentType	1

Capability Attributes

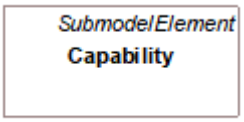


Figure 32. Metamodel of Capabilities

Note: the *semanticId* of a capability is typically an ontology, which enables reasoning on capabilities. For information and examples on how to apply the concept of capability and how to map it to one or more skills implementing the capability, please refer to [27]. The mapping is done via a relationship element with the corresponding semantics. A skill is typically a property or an operation. In more complex cases, the mapping can also be a collection or a complete submodel.

Class:	Capability
Explanation:	A capability is the implementation-independent description of the potential of an asset to achieve a certain effect in the physical or virtual world.
Inherits from:	SubmodelElement

Attribute	Explanation	Type	Card.

Data Element and Overview of Data Element Types

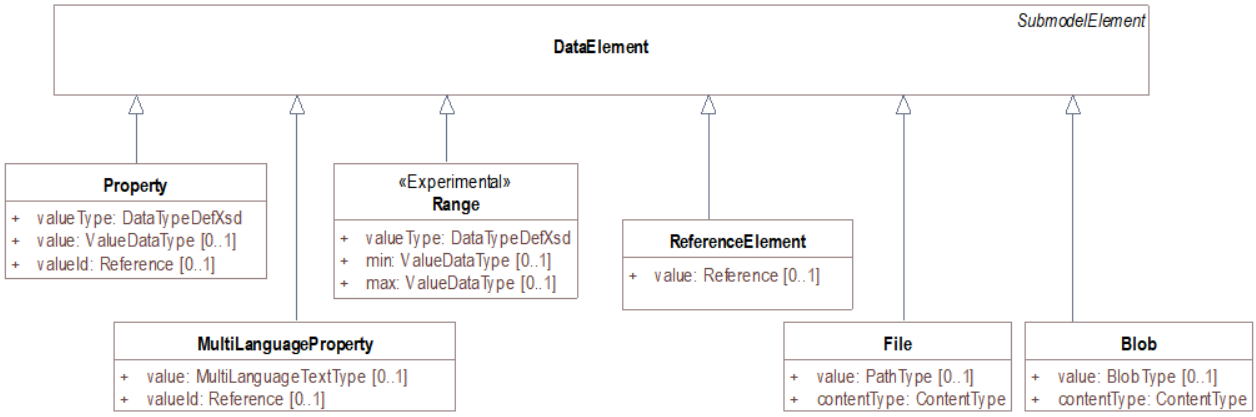


Figure 33. Metamodel of Data Elements

A data element is a submodel element that is not further composed of other submodel elements.

A data element is a submodel element that has a value or a predefined number of values like range data elements.

The type of value differs for different subtypes of data elements. Data elements include properties, file handling, and reference elements, see Figure 33.

Class:	DataElement <<abstract>>
Explanation:	A data element is a submodel element that is not further composed of other submodel elements. A data element is a submodel element that has a value. The type of value differs for different subtypes of data elements. Constraint AASd-090: for data elements, *category* (inherited by *Referable*) shall be one of the following values: CONSTANT, PARAMETER or VARIABLE. Default: VARIABLE Note: categories are deprecated and should no longer be used.
Inherits from:	SubmodelElement

Attribute	Explanation	Type	Card.

Entity Attributes

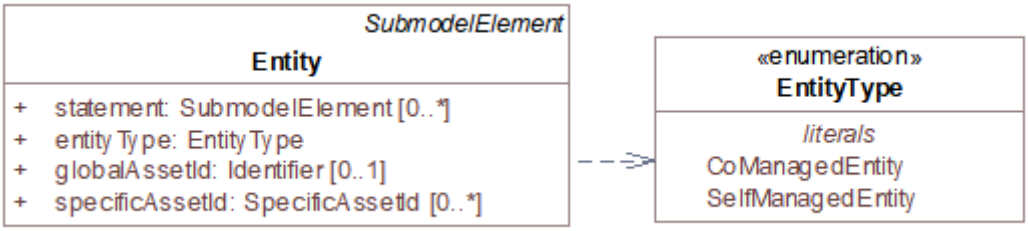


Figure 34. Metamodel of Entities

The entity submodel element is designed to be used in submodels defining the relationship between the parts of the composite asset it is composed of (e.g. bill of material). These parts are called entities. Not all entities have a global asset ID.

Class:	Entity
Explanation:	An entity is a submodel element that is used to model entities. Constraint AASd-014: Either the attribute *globalAssetId* or *specificAssetId* of an *Entity* must be set if *Entity/entityType* is set to "*SelfManagedEntity*". Otherwise, they do not exist.
Inherits from:	SubmodelElement

Attribute	Explanation	Type	Card.
statement	Describes statements applicable to the entity by a set of submodel elements, typically with a qualified value	SubmodelElement	0..*
entityType	Describes whether the entity is a co-managed entity or a self-managed entity	EntityType	1
globalAssetId	Global identifier of the asset the entity is representing	Identifier	0..1
specificAssetId	Reference to a specific asset ID representing a supplementary identifier of the asset represented by the Asset Administration Shell	SpecificAssetId	0..*

Enumeration:	EntityType
Explanation:	Enumeration for denoting whether an entity is a self-managed entity or a co-managed entity
Set of:	—

Literal	Explanation
CoManagedEntity	There is no separate Asset Administration Shell for co-managed entities. Co-managed entities need to be part of a self-managed entity.
SelfManagedEntity	Self-managed entities have their own Asset Administration Shell but can be part of another composite self-managed entity. The asset of an I4.0 Component is a self-managed entity per definition.

Event Attributes

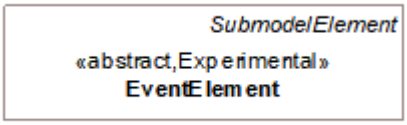


Figure 35. Metamodel of Events

Class:	EventElement <<abstract>> <<Experimental>>
Explanation:	An event element
Inherits from:	SubmodelElement

Attribute	Explanation	Type	Card.

File Attributes

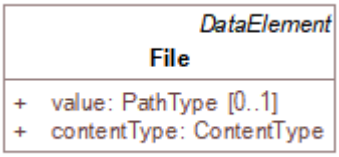


Figure 36. Metamodel of File Submodel Element

A media type (also MIME type and content type) is a two-part identifier for file formats and format contents transmitted via the Internet. The Internet Assigned Numbers Authority (IANA) is the official authority for the standardization and publication of these classifications.

Note: for information on handling supplementary external files in exchanging Asset Administration Shell specification in AASX package format see also Part 5 of the series "Details of the Asset Administration Shell". An absolute path is used in case the file exists independently of the Asset Administration Shell. A relative path, relative to the package root, should be used if the file is part of a serialized package of

the Asset Administration Shell.

Class:	File
Explanation:	A file is a data element that represents an address to a file (a locator). The value is a URI that can represent an absolute or relative path.
Inherits from:	DataElement

Attribute	Explanation	Type	Card.
value	Path and name of the file (with file extension) The path can be absolute or relative.	PathType	0..1
contentType	Content type of the content of the file	ContentType	1

Multi Language Property Attributes

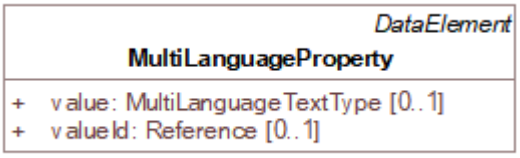


Figure 37. Metamodel of Multi Language Properties

Class:	MultiLanguageProperty
Explanation:	A property is a data element that has a multi-language value. Constraint AASd-012: if both the *MultiLanguageProperty/value* and the *MultiLanguageProperty/valueId* are present, the meaning must be the same for each string in a specific language, as specified in *MultiLanguageProperty/valueId*.
Inherits from:	DataElement

Attribute	Explanation	Type	Card.
value	The value of the property instance	MultiLanguageTextType	0..1

Attribute	Explanation	Type	Card.
valueId	Reference to the global unique ID of a coded value. Note: it is recommended to use an external reference.	Reference	0..1

Operation Attributes

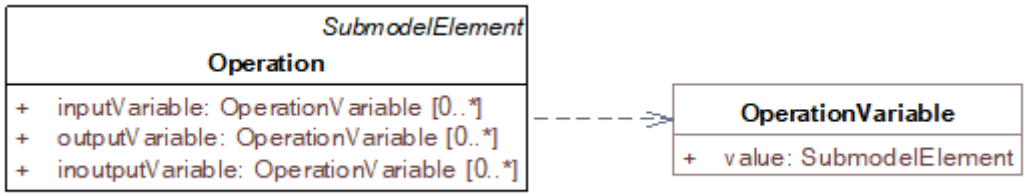


Figure 38. Metamodel of Operations

Class:	Operation
Explanation:	An operation is a submodel element with input and output variables. Constraint AASd-134: For an *Operation,* the *idShort* of all *inputVariable/value*, *outputVariable/value,* and *inoutputVariable/value* shall be unique.
Inherits from:	SubmodelElement

Attribute	Explanation	Type	Card.
inputVariable	Input parameter of the operation	OperationVariable	0..*
outputVariable	Output parameter of the operation	OperationVariable	0..*
inoutputVariable	Parameter that is input and output of the operation	OperationVariable	0..*

Class:	OperationVariable
Explanation:	The value of an operation variable is a submodel element that is used as input and/or output variable of an operation.
Inherits from:	—

Attribute	Explanation	Type	Card.
value	Describes an argument or result of an operation via a submodel element	SubmodelElement	1

Note 1: operations typically specify the behavior of a component in terms of procedures. Hence, operations enable the specification of services with procedure-based interactions [23].

Note 2: OperationVariable is introduced as separate class to enable future extensions, e.g. for adding a default value or cardinality (option/mandatory).

Note 3: even if the submodel element as the value of an input and an output variable have the same idShort, this does not mean that they are identical or mapped to the same variable since OperationVariables are no referables. The same applies to two input variables or an input variable and an inoutputVariable a.s.o.

Property Attributes

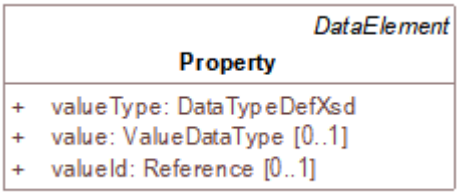


Figure 39. Metamodel of Properties

Class:	Property
Explanation:	A property is a data element that has a single value. Constraint AASd-007: If both the *Property/value* and the *Property/valueId* are present, the value of *Property/value* needs to be identical to the value of the referenced coded value in *Property/valueId*.
Inherits from:	DataElement

Attribute	Explanation	Type	Card.
valueType	Data type of the value attribute	DataTypeDefXsd	1
value	The value of the property instance	ValueDataType	0..1

Attribute	Explanation	Type	Card.
valueId	Reference to the global unique ID of a coded value Note: it is recommended to use an external reference.	Reference	0..1

Range Attributes

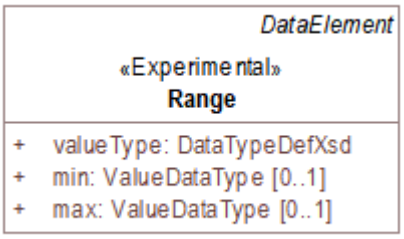


Figure 40. Metamodel of Ranges

Class:	Range <<Experimental>>
Explanation:	A range data element is a data element that defines a range with min and max.
Inherits from:	DataElement

Attribute	Explanation	Type	Card.
valueType	Data type of the min und max attributes	DataTypeDefXsd	1
min	The minimum value of the range If the min value is missing, the value is assumed to be negative infinite.	ValueDataType	0..1
max	The maximum value of the range If the max value is missing, the value is assumed to be positive infinite.	ValueDataType	0..1

Reference Element Attributes

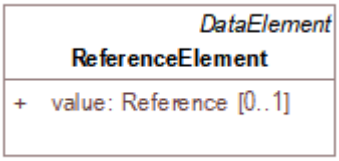


Figure 41. Metamodel of Reference Elements

Class:	ReferenceElement
Explanation:	A reference element is a data element that defines a logical reference to another element within the same or another Asset Administration Shell or a reference to an external object or entity.
Inherits from:	DataElement

Attribute	Explanation	Type	Card.
value	External reference to an external object or entity or a logical reference to another element within the same or another Asset Administration Shell (i.e. a model reference to a *Referable*)	Reference	0..1

For more information on references, see Clause 5.3.9.

Relationship Element Attributes

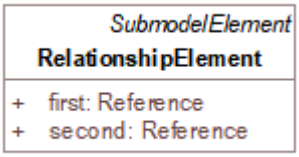


Figure 42. Metamodel of Relationship Elements

Class:	RelationshipElement
Explanation:	A relationship element is used to define a relationship between two elements being either referable (model reference) or external (external reference).
Inherits from:	SubmodelElement

Attribute	Explanation	Type	Card.
first	Reference to the first element in the relationship taking the role of the subject	Reference	1

Attribute	Explanation	Type	Card.
second	Reference to the second element in the relationship taking the role of the object	Reference	1

Submodel Element Collection Attributes

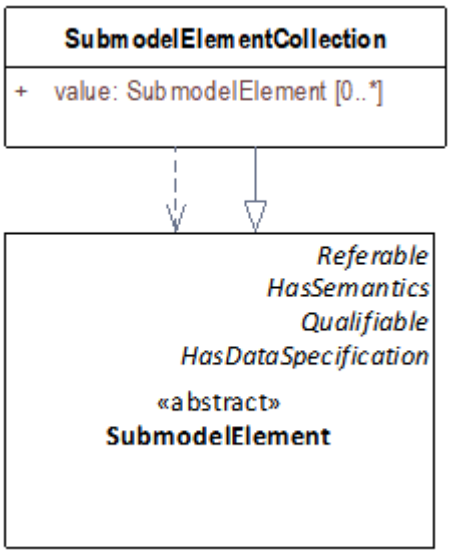


Figure 43. Metamodel of Submodel Element Collections

Submodel Element Collections are used for complex elements with a typically fixed set of properties with unique names. This set of properties is typically predefined by the semantic definition (referenced via *semanticId*) of the submodel element collection. Each property within the collection itself should have clearly defined semantics.

> Note: the different elements of a submodel element collection do not have to have different *semanticId*s. However, in these cases the usage of a *SubmodelElementList* should be considered.

Example: a single document has a predefined set of properties like title, version, author, etc. They logically belong to a document. So a single document is represented by a *SubmodelElementCollection*. An asset usually has many different documents available like operating instructions, safety instructions, etc. The set of all documents is represented by a *SubmodelElementList* (see Clause *5.3.7.17)*. In this case, we have a *SubmodelElementList* of *SubmodelElementCollection*s.

> Note: the elements within a submodel element collection are not ordered. Every element has a unique ID (its "idShort"). However, it is recommended to adhere to the order defined in the submodel template.

Class:	SubmodelElementCollection

Explanation:	A submodel element collection is a kind of struct, i.e. a logical encapsulation of multiple named values.
Inherits from:	SubmodelElement

Attribute	Explanation	Type	Card.
value	Submodel element contained in the collection	SubmodelElement	0..*

Submodel Element List Attributes

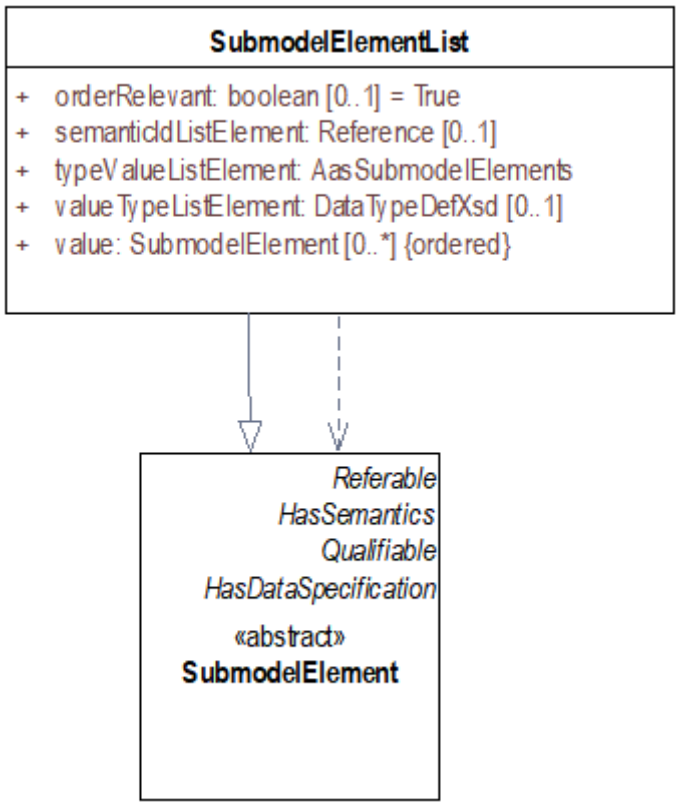


Figure 44. Metamodel of Submodel Element Lists

Submodel element lists are used for sets (i.e. unordered collections without duplicates), ordered lists (i.e. ordered collections that may contain duplicates), bags (i.e. unordered collections that may contain duplicates), and ordered sets (i.e. ordered collections without duplicates).

> Note: there is no *idShort* for submodel elements in lists (see Constraint AASd-120).

Submodel element lists are also used to create multi-dimensional arrays. A two-dimensional array list[3][5] with *Property* values would be realized like follows: the first submodel element list would contain three *SubmodelElementList* elements. Each of these three *SubmodelElementLists* would contain 5 single *Property*

elements. The *semanticId* of the contained properties would be the same for all lists in the first list, i.e. *semanticIdListElement* would be identical for all three lists contained in the first list. The *semanticId* of the three contained lists would differ depending on the dimension it represents. In case of complex values in the array, a *SubmodelElementCollection* would be used as values in the leaf lists.

Similarly, a table with three columns can be represented. In this case a *SubmodelElementCollection* with three *SubmodelElementLists* would be contained and the *semanticId* as well as the *semanticIdListElement* for the three columns would differ.

Matching strategies for semantic IDs are explained in Clause 4.4.1.

Class:	SubmodelElementList
Explanation:	A submodel element list is an ordered list of submodel elements. Note: the list is ordered although the ordering might not be relevant (see attribute "orderRelevant".) The numbering starts with Zero (0). Constraint AASd-107: If a first level child element in a *SubmodelElementList* has a semanticId, it shall be identical to *SubmodelElementList/semanticIdListElement*. Constraint AASd-114: If two first level child elements in a *SubmodelElementList* have a *semanticId*, they shall be identical. Constraint AASd-115: If a first level child element in a *SubmodelElementList* does not specify a *semanticId*, the value is assumed to be identical to *SubmodelElementList/semanticIdListElement*. Constraint AASd-108: All first level child elements in a *SubmodelElementList* shall have the same submodel element type as specified in *SubmodelElementList/typeValueListElement*. Constraint AASd-109: If *SubmodelElementList/typeValueListElement* is equal to *Property* or *Range, SubmodelElementList/valueTypeListElement* shall be set and all first level child elements in the *SubmodelElementList* shall have the value type as specified in *SubmodelElementList/valueTypeListElement*.
Inherits from:	SubmodelElement

Attribute	Explanation	Type	Card.
orderRelevant	Defines whether order in list is relevant. If *orderRelevant* = false, the list represents a set or a bag. Default: True	boolean	0..1
value	Submodel element contained in the list	SubmodelElement	0..*
semanticIdListElement	Semantic ID which the submodel elements contained in the list match Note: it is recommended to use an external reference.	Reference	0..1
typeValueListElement	The submodel element type of the submodel elements contained in the list	AasSubmodelElements	1
valueTypeListElement	The value type of the submodel element contained in the list	DataTypeDefXsd	0..1

5.3.8. Concept Description Attributes

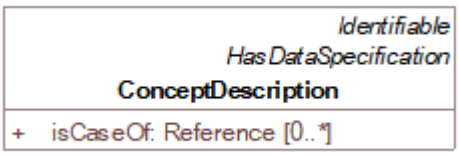


Figure 45. Metamodel of Concept Descriptions

Class:	ConceptDescription
Explanation:	The semantics of a property or other elements that may have a semantic description is defined by a concept description. The description of the concept should follow a standardized schema (realized as data specification template).
Inherits from:	Identifiable; HasDataSpecification

Attribute	Explanation	Type	Card.
isCaseOf	Reference to an external definition the concept is compatible to or was derived from Note: it is recommended to use an external reference, i.e. *Reference/type = ExternalReference*. Note: compare with is-case-of relationship in ISO 13584-32 & IEC EN 61360	Reference	0..*

Different types of submodel elements require different attributes to describe their semantics. This is why a concept description has at least one data specification template associated with it. This template defines the attributes needed to describe the semantics.

See Clause 5.3.11.3 for predefined data specification templates.

5.3.9. Environment Attributes

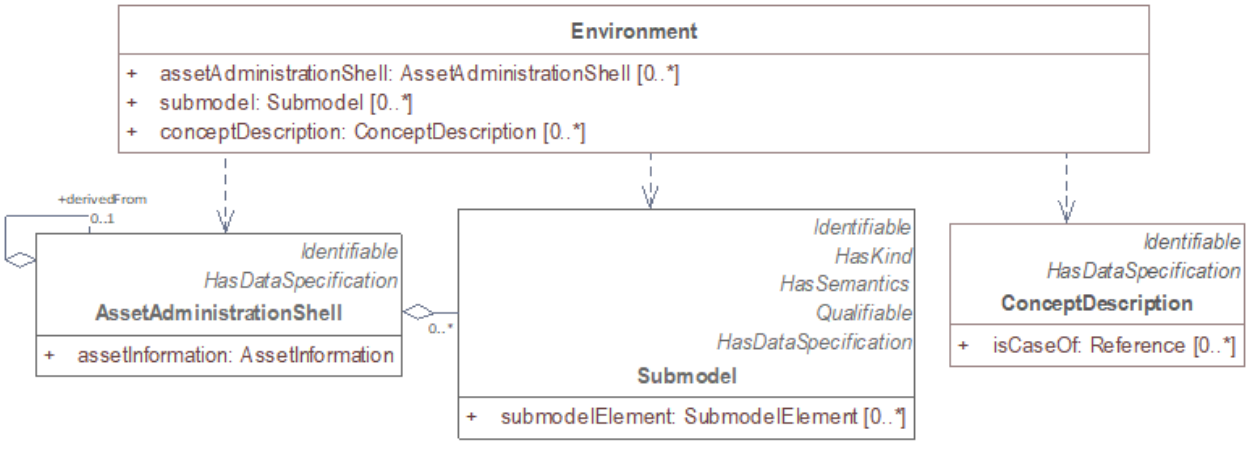


Figure 46. Metamodel for Environment

Note: *Environment* is not an identifiable or referable element. It is introduced to enable file transfer as well as serialization.

Class:	Environment
Explanation:	Container for the sets of different identifiables
Inherits from:	Reference

Attribute	Explanation	Type	Card.
assetAdministrationShell	Asset Administration Shell	AssetAdministrationShell	0..*
submodel	Submodel	Submodel	0..*
conceptDescription	Concept description	ConceptDescription	0..*

5.3.10. Referencing in Asset Administration Shells

Overview

To date, two kinds of references are distinguished: references to external objects or entities (external reference) and references to model elements of the same or another Asset Administration Shell (model reference). Model references are also used for metamodel inherent relationships like submodels of an Asset Administration Shell (notation see Annex A).

An external reference is a unique identifier. The identifier can be a concatenation of different identifiers, representing e.g. an IRDI path.

> Note: references should not be mixed up with locators. Even URLs can be used as identifiers and do not necessarily describe a resource that can be accessed.

Reference Attributes

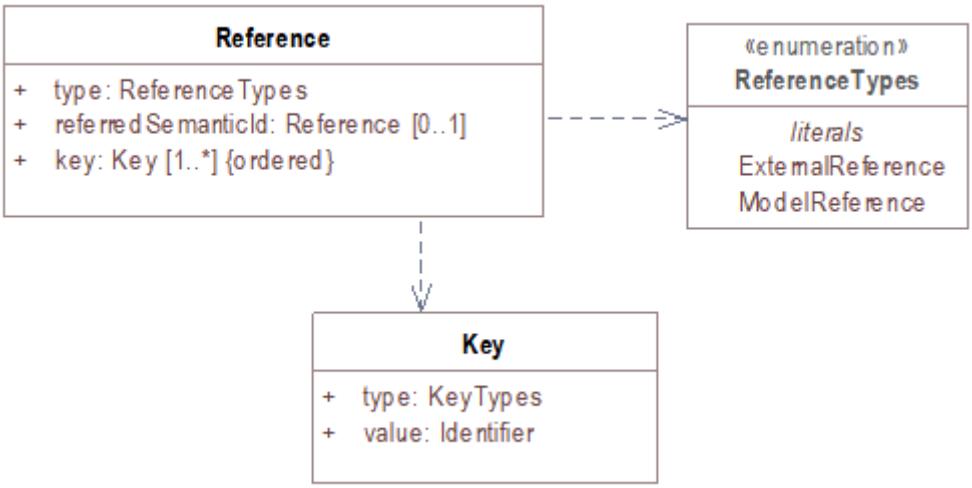


Figure 47. Metamodel of Reference

See Clause 4.4.2 for reference matching.

Class:	Reference
Explanation:	Reference to either a model element of the same or another Asset Administration Shell or to an external entity A model reference is an ordered list of keys, each key referencing an element. The complete list of keys may, for example, be concatenated to a path that gives unique access to an element. An external reference is a reference to an external entity.
Inherits from:	—

Attribute	Explanation	Type	Card.
type	Type of the reference Denotes whether reference is an external reference or a model reference	ReferenceTypes	1

Attribute	Explanation	Type	Card.
referredSemanticId	Expected semantic ID of the referenced model element (*Reference/type=ModelReference*); there typically is no semantic ID for for the referenced object of external references *(Reference/type=ExternalReference)*. Note 1: if Reference/referredSemanticId is defined, the semanticId of the model element referenced should have a matching semantic ID. If this is not the case, a validator should raise a warning. Note 2: it is recommended to use an external reference for the semantic ID expected from the referenced model element.	Reference	0..1
key <<ordered>>	Unique reference in its name space	Key	1..*

Enumeration:	ReferenceTypes
Explanation:	Enumeration for denoting whether an element is an external or model reference
Set of:	—

Literal	Explanation
ExternalReference	External reference
ModelReference	Model reference

Key Attributes

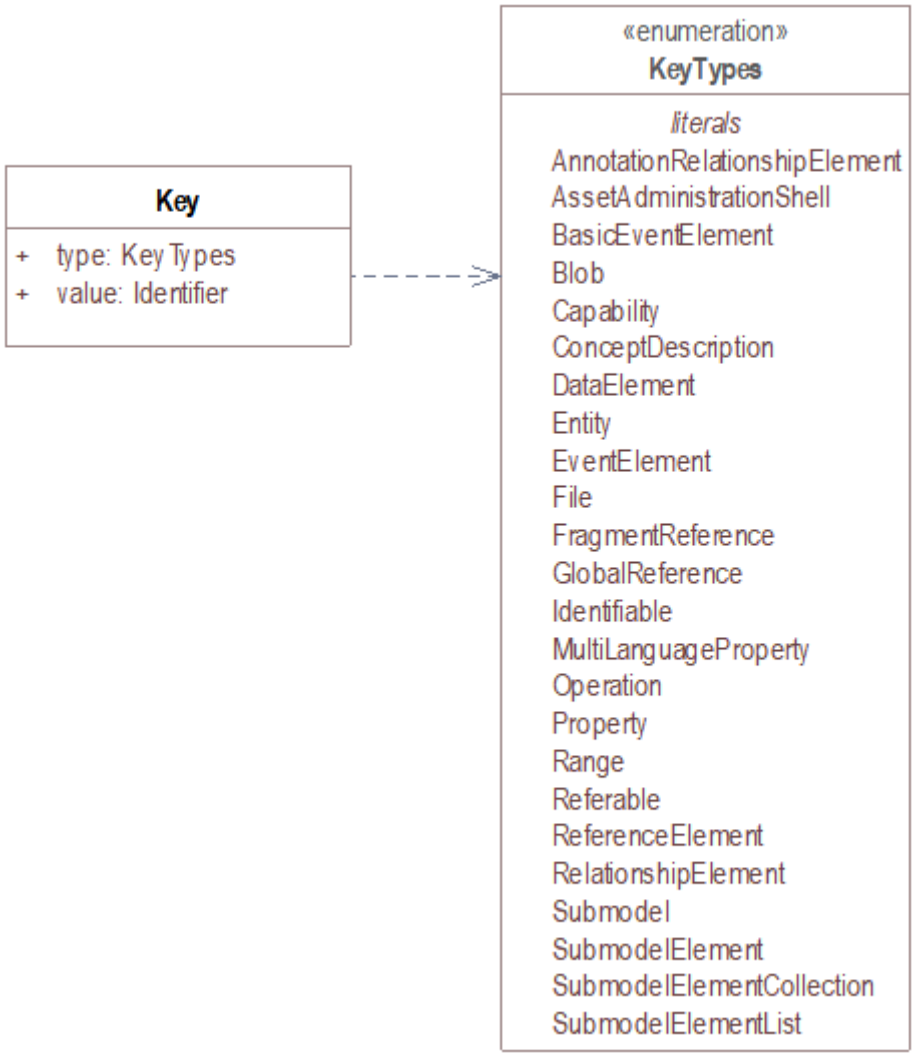


Figure 48. Metamodel of Keys

Keys are used to define references (*Reference*).

Figure 49 presents a logical model of key types. These logical enumerations are used to formulate constraints.

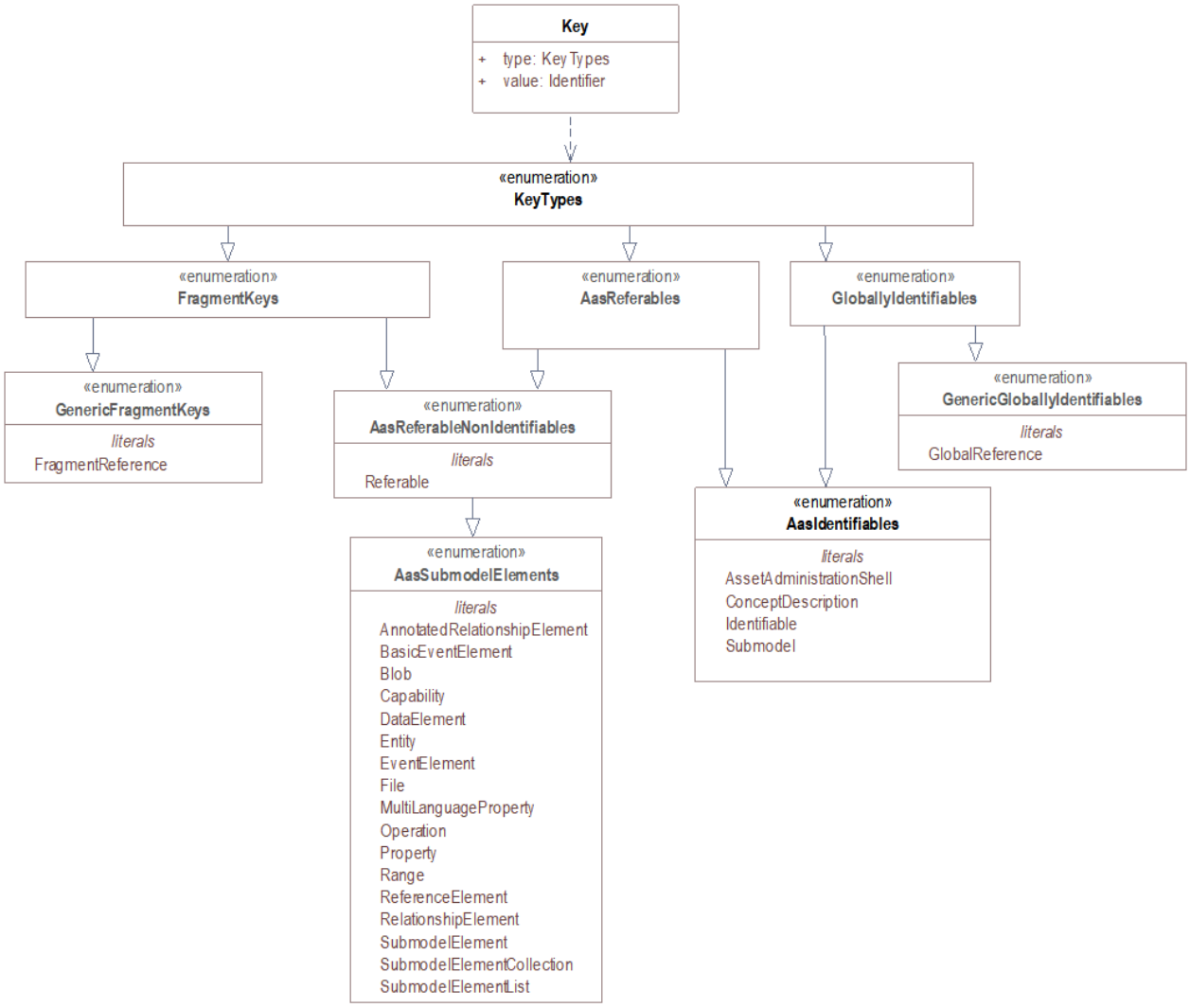


Figure 49. Logical Model for Keys of References (non-normative)

Figure 50. Metamodel of KeyTypes Enumeration

Class:	Key
Explanation:	A key is a reference to an element by its ID
Inherits from:	—

Attribute	Explanation	Type	Card.
type	Denotes which kind of entity is referenced If *Key/type = GlobalReference,* the key represents a reference to a source that can be globally identified. If *Key/type = FragmentReference,* the key represents a bookmark or a similar local identifier within its parent element as specified by the key that precedes this key. In all other cases, the key references a model element of the same or another Asset Administration Shell. The name of the model element is explicitly listed.	KeyTypes	1
value	The key value, for example an IRDI or an URI	Identifier	1

An example for using a *FragmentId* as type of a key is a reference to an element within a file that is part of an Asset Administration Shell aasx package.

Figure 51. Metamodel of AasSubmodelElements Enumeration

Enumeration:	KeyTypes
Explanation:	Enumeration of different key value types within a key
Set of:	FragementKeys; AasReferables, GloballyIdentifiables

Literal	**Explanation**
AnnotatedRelationshipElement	Annotated relationship element
AssetAdministrationShell	Asset Administration Shell
BasicEventElement	Basic event element
Blob	Blob
Capability	Capability
ConceptDescription	Concept Description
DataElement	Data Element Note: data elements are abstract, i.e. if a key uses "DataElement", the reference may be a property, file, etc.

Literal	Explanation
Entity	Entity
EventElement	Event Note: event element is abstract.
File	File
FragmentReference	Bookmark or a similar local identifier of a subordinate part of a primary resource
GlobalReference	Global reference
Identifiable	Identifiable Note: identifiable is abstract, i.e. if a key uses "Identifiable" the reference may be an Asset Administration Shell, a submodel or a concept description.
MultiLanguageProperty	Property with a value that can be provided in multiple languages
Operation	Operation
Property	Property
Range	Range with min and max
Referable	==== Note: referables are abstract, i.e. if a key uses "Referable", the reference may be an Asset Administration Shell, a property, etc. ====
ReferenceElement	Reference
RelationshipElement	Relationship
Submodel	Submodel
SubmodelElement	Submodel element Note: submodel elements are abstract, i.e. if a key uses "SubmodelElement", the reference may be a property, a submodel element list, an operation, etc.
SubmodelElementCollection	Struct of submodel elements
SubmodelElementList	List of submodel elements

Enumeration:	FragmentKeys
Explanation:	Enumeration of different fragment key value types within a key Note: not used as type but in constraints.
Set of:	AASReferableNonIdentifiables, GenericFragmentKeys

Literal	Explanation
AnnotatedRelationshipElement	Annotated relationship element
BasicEventElement	Basic event element
Blob	Blob
Capability	Capability
DataElement	Data element Note: data elements are abstract, i.e. if a key uses "DataElement", the reference may be a property, a file, etc.
Entity	Entity
EventElement	Event Note: event elements are abstract.
File	File
FragmentReference	Bookmark or a similar local identifier of a subordinate part of a primary resource
MultiLanguageProperty	Property with a value that can be provided in multiple languages
Operation	Operation
Property	Property
Range	Range with min and max
ReferenceElement	Reference
RelationshipElement	Relationship

Literal	Explanation
SubmodelElement	Submodel element Note: submodel elements are abstract, i.e. if a key uses "SubmodelElement", the reference may be a property, a submodel element list, an operation, etc.
SubmodelElementCollection	Struct of submodel elements
SubmodelElementList	List of submodel elements

Enumeration:	GloballyIdentifiables
Explanation:	Enumeration of different key value types within a key Note: not used as type but in constraints.
Set of:	AasIdentifiables, GenericGloballyIdentifiables

Literal	Explanation
AssetAdministrationShell	Asset Administration Shell
ConceptDescription	Concept description
GlobalReference	Global reference
Identifiable	Identifiable Note: identifiables are abstract, i.e. if a key uses "Identifiable", the reference may be an Asset Administration Shell, a submodel, or a concept description.
Submodel	Submodel

Enumeration:	AasReferableNonIdentifiables
Explanation:	Enumeration of different fragment key value types within a key Note: not used as type but in constraints.
Set of:	AasSubmodelElements

Literal	Explanation
AnnotatedRelationshipElement	Annotated relationship element
BasicEventElement	Basic event element
Blob	Blob
Capability	Capability
DataElement	Data element Note: data elements are abstract, i.e. if a key uses "DataElement", the reference may be a property, a file, etc.
Entity	Entity
EventElement	Event Note: event elements are abstract.
File	File
MultiLanguageProperty	Property with a value that can be provided in multiple languages
Operation	Operation
Property	Property
Range	Range with min and max
ReferenceElement	Reference
RelationshipElement	Relationship
SubmodelElement	Submodel element Note: submodel elements are abstract, i.e. if a key uses "SubmodelElement", the reference may be a property, a SubmodelElementList, an operation, etc.
SubmodelElementCollection	Struct of submodel elements
SubmodelElementList	List of submodel elements

Enumeration:	AasSubmodelElements

Explanation:	Enumeration of different fragment key value types within a key
Set of:	—

Literal	Explanation
AnnotatedRelationshipElement	Annotated relationship element
BasicEventElement	Basic event element
Blob	Blob
Capability	Capability
DataElement	Data element Note: data elements are abstract, i.e. if a key uses "DataElement", the reference may be a property, a file, etc.
Entity	Entity
EventElement	Event Note: event elements are abstract.
File	File
MultiLanguageProperty	Property with a value that can be provided in multiple languages
Operation	Operation
Property	Property
Range	Range with min and max
ReferenceElement	Reference
RelationshipElement	Relationship
SubmodelElement	Submodel element Note: Submodel elements are abstract, i.e. if a key uses "SubmodelElement", the reference may be a property, a SubmodelElementList, an operation, etc.
SubmodelElementCollection	Struct of submodel elements

Literal	Explanation
SubmodelElementList	List of submodel elements

Enumeration:	AasReferables
Explanation:	Enumeration of referables Note: not used as type but in constraints.
Set of:	AASReferableNonIdentifiables, AasIdentifiables

Literal	Explanation
AssetAdministrationShell	Asset Administration Shell
ConceptDescription	Concept description
Identifiable	Identifiable Note: Identifiables are abstract, i.e. if a key uses "Identifiable", the reference may be an Asset Administration Shell, a submodel, or a concept description.
Submodel	Submodel
AnnotatedRelationshipElement	Annotated relationship element
BasicEventElement	Basic event element
Blob	Blob
Capability	Capability
DataElement	Data element Note: data elements are abstract, i.e. if a key uses "DataElement", the reference may be a property, a file, etc.
Entity	Entity
EventElement	Event Note: event elements are abstract.

Literal	Explanation
File	File
MultiLanguageProperty	Property with a value that can be provided in multiple languages
Operation	Operation
Property	Property
Referable	Referable Note: referables are abstract, i.e. if a key uses "Referable", the reference may be an Asset Administration Shell, a property, etc.
Range	Range with min and max
ReferenceElement	Reference
RelationshipElement	Relationship
SubmodelElement	Submodel element Note: submodel elements are abstract, i.e. if a key uses "SubmodelElement", the reference may be a property, a submodel element list, an operation, etc.
SubmodelElementCollection	Struct of submodel elements
SubmodelElementList	List of submodel elements

Enumeration:	GenericFragmentKeys
Explanation:	Enumeration of different fragment key value types within a key Note: not used as type but in constraints.
Set of:	—

Literal	Explanation
FragmentReference	Bookmark or a similar local identifier of a subordinate part of a primary resource

Enumeration:	AasIdentifiables

Explanation:	Enumeration of different key value types within a key Note: not used as type but in constraints.
Set of:	—

Literal	Explanation
AssetAdministrationShell	Asset Administration Shell
ConceptDescription	Concept description
Identifiable	Identifiable Note: Identifiables are abstract, i.e. if a key uses "Identifiable", the reference may be an Asset Administration Shell, a submodel, or a concept description.
Submodel	Submodel

Enumeration:	GenericGloballyIdentifiables
Explanation:	Enumeration of different key value types within a key
Set of:	—

Literal	Explanation
GlobalReference	Global reference

Constraints

Constraint AASd-121: For *References*, the value of *Key/type* of the first *key* of *Reference/keys* shall be one of *GloballyIdentifiables*.

Constraint AASd-122: For external references, i.e. *References* with *Reference/type* = *ExternalReference*, the value of *Key/type* of the first key of *Reference/keys* shall be one of *GenericGloballyIdentifiables*.

Constraint AASd-123: For model references, i.e. *References* with *Reference/type* = *ModellReference*, the value of *Key/type* of the first *key* of *Reference/keys* shall be one of *AasIdentifiables.*

Constraint AASd-124: For external references, i.e. *References* with *Reference/type* = *ExternalReference*, the last *key* of *Reference/keys* shall be either one of *GenericGloballyIdentifiables* or one of *GenericFragmentKeys.*

Constraint AASd-125: For model references, i.e. *Reference*s with *Reference/type* = *ModelReference* with more than one key in *Reference/keys,* the value of *Key/type* of each of the keys following the first key of *Reference/keys* shall be one of *FragmentKeys*.

> Note: constraint AASd-125 ensures that the shortest path is used.

Constraint AASd-126: For model references, i.e. *Reference*s with *Reference/type* = *ModelReference* with more than one key in *Reference/keys,* the value of *Key/type* of the last *Key* in the reference key chain may be one of *GenericFragmentKeys* or no key at all shall have a value out of *GenericFragmentKeys.*

Constraint AASd-127: For model references, i.e. *Reference*s with *Reference/type* = *ModelReference* with more than one key in *Reference/keys,* a key with *Key/type FragmentReference* shall be preceded by a key with *Key/type File* or *Blob*. All other Asset Administration Shell fragments, i.e. *Key/type* values out of *AasSubmodelElements,* do not support fragments*.*

> Note: which kind of fragments are supported depends on the content type and the specification of allowed fragment

identifiers for the corresponding resource referenced.

Constraint AASd-128: For model references, i.e. *Reference*s with *Reference/type* = *ModelReference*, the *Key/value* of a *Key* preceded by a *Key* with *Key/type=SubmodelElementList* is an integer number denoting the position in the array of the submodel element list.

Examples for valid references:

(Submodel)https://example.com/aas/1/1/1234859590

(GlobalReference)https://example.com/specification.html

Examples for valid external references:

(GlobalReference)https://example.com/ressource

(GlobalReference)0173-1#02-EXA123#001

(GlobalReference)https://example.com/specification.html (FragmentReference)Hints

> Note: (GlobalReference)https://example.com/specification.html (FragmentReference)Hints represents the path with fragment identifier https://example.com/specification.html#Hints

Examples for valid model references:

(AssetAdministrationShell)https://example.com/aas/1/0/12348

(Submodel)https://example.com/aas/1/1/1234859590

(Submodel)https://example.com/aas/1/1/1234859590, (File)Specification

(ConceptDescription)0173-1#02-BAA120#008

(Submodel)https://example.com/aas/1/1/1234859590, (SubmodelElementList)Documents, (SubmodelElementCollection)0, (MultiLanguageProperty)Title

(Submodel)https://example.com/aas/1/1/1234859590, (SubmodelElementCollection)Manual, (MultiLanguageProperty)Title

Note: "(SubmodelElementCollection)0, (MultiLanguageProperty)Title" may be identical to "(SubmodelElementCollection)Manual, (MultiLanguageProperty)Title" semantically and content-wise. The difference is that more than one document is allowed in the first submodels and thus a submodel element list is defined: elements in a list are numbered. However, it is also possible to define a display name in this case. The display name of the SubmodelElementCollection should be the same in both bases, e.g. "Users Manual".

(Submodel)https://example.com/aas/1/1/1234859590, (File)Specification, (FragmentReference)Hints

Note: assuming the file has the value using the absolute path https://example.com/specification.html (and not a relative path), the first reference points to the same information as the global reference (GlobalReference)https://example.com/specification.html, (FragmentReference)Hints.

(Submodel)https://example.com/aas/1/1/1234859590, (Blob)Specification, (FragmentReference)Hints

Examples for invalid model references:

(GlobalReference)https://example.com/aas/1/1/1234859590

(Property)0173-1#02-BAA120#008

(Submodel)https://example.com/aas/1/1/1234859590, (EventElement)Event, (FragmentReference)Comment

(AssetAdministrationShell)https://example.com/aas/1/0/12348**,**
(Submodel)https://example.com/aas/1/1/1234859590, (Property)Temperature

is not a valid model reference because *AssetAdministrationShell* and *Submodel* are both global identifiables.

5.3.11. Primitive and Simple Data Types

Predefined Simple Data Types

The metamodel of the Asset Administration Shell uses basic data types as defined in the XML Schema Definition (XSD)[3]. See Table 7 for an overview of the used types. Their definition is outside the scope of this document.

The meaning and format of xsd types is specified in https://www.w3.org/XML/Schema. The simple type "langString" is specified in the Resource Description Framework (RDF)[4].

See Clause 5.3.12.6 for constraints on types.

Table 7. Simple Data Types Used in Metamodel

Source	Basic Data Type	Value Range	Sample Values
xsd	string	Character string (but not all Unicode character strings)	"Hello world", "¬¬¬¬¬¬¬¬ ¬¬¬¬¬¬", "¬¬¬¬¬¬¬¬""
xsd	base64Binary	base64-encoded binary data	"a3Vtb3dhc2hlcmU="
xsd	boolean	true, false	true, false
xsd	dateType	Date and time with or without time zone	"2000-01-01T14:23:00", "2000-01-01T14:23:00.66372+14:00"[5]
xsd	duration	Duration of time	"-P1Y2M3DT1H", "PT1H5M0S"
rdf	langString	Strings with language tags	"Hello"@en, "Hallo"@de Note: this is written in RDF/Turtle syntax, only "Hello" and "Hallo" are the actual values.

Primitive Data Types

Table 8 lists the Primitives used in the metamodel. Primitive data types start with a capital letter.

Note: see Clause 5.3.12.6 for constraints on types.

Table 8. Primitive Data Types Used in Metamodel

Primitive	Definition	Value Examples
BlobType	*base64binary* to represent file content (binaries and non-binaries)	<?xml version="1.0" encoding="UTF-8"?> <schema elementFormDefault="qualified" targetNamespace="http://www.admin-shell.io/aas/2/0" xmlns="http://www.w3.org/2001/XMLSchema" xmlns:aas="http://www.admin-shell.io/aas/2/0" /> MZ¬_____ÿÿ__¸_____@_____ _____€_____º__´ Í!¸_LÍ!This program cannot be run in DOS mode.$_____PE__L___Rö\^_____à_
ContentType	*string* with max 100 and min 1 characters Note: any content type as in RFC2046. A media type (also MIME type and content type) […] is a two-part identifier for file formats and format contents transmitted on the Internet. The Internet Assigned Numbers Authority (IANA) is the official authority for the standardization and publication of these classifications. Media types were originally defined in Request for Comments 2045 in November 1996 as a part of MIME specification, for denoting type of email message content and attachments.[6]	application/pdf image/jpeg
Identifier	*string* with max 2,000 and min 1 characters	https://cust/123456 0173-1#02-BAA120#008

Primitive	Definition	Value Examples
LabelType	*string* with max 64 and min 1 characters	"ABC1234"
LangStringSet	*Array of elements of type langString* Note 1: langString is a RDF data type. Note 2: a langString is a string value tagged with a language code. Realization depends on the serialization rules for a technology.	In xml: <aas:langString lang="EN">This is a multi-language value in English</aas:langString> <aas:langString lang="DE"> Das ist ein Multi-Language-Wert in Deutsch </aas:langString> In rdf: "This is a multi-language value in English"@en ; "Das ist ein Multi-Language-Wert in Deutsch"@de In JSON: "description": [\{ "language":"en", "text": "This is a multi-language value in English." }, \{ "language":"de", "text": "Das ist ein Multi-Language-Wert in Deutsch." }]
MessageTopicType	*string* with max 255 and min 1 characters	

Primitive	Definition	Value Examples
MultiLanguageNameType	*LangStringSet* Each langString within the array of strings has a max of 1023 and a min of 1 characters (as for NameType).	*See LangStringSet*
MultiLanguageTextType	*LangStringSet* Each string within langString has a max of 1,023 and min of 1 characters.	*See LangStringSet*
NameType	*string* with max 128 and min 1 characters	"ManufacturerPartId"
PathType	*Identifier* Note: for any string conformant to RFC8089, the "file" URI scheme (for relative and absolute file paths) applies.	*/Specification.pdf* file:c:/local/Specification.pdf file://host.example.com/path/to/file
RevisionType	*string* with max 4 and min 1 characters following the following regular expression: ^([0-9]|[1-9][0-9]*)$	"0" "7" "567"
QualifierType	*NameType*	"ExpressionSemantic" (as specified in DIN SPEC 92000:2019-09, see [16]) "life cycle qual" (as specified in IEC 61360-7 - IEC/SC 3D - Common Data Dictionary (CDD - V2.0015.0004)

Primitive	Definition	Value Examples
VersionType	*string* with max 4 and min 1 characters following the following regular expression: ^([0-9]|[1-9][0-9]*)$	"1" "9999"
ValueDataType	*any xsd atomic type as specified via DataTypeDefXsd*	"This is a string value" 10 1.5 2020-04-01 True

Enumeration for Submodel Element Value Types

Enumerations are primitive data types. Most of the enumerations are defined in the context of their class. This clause defines enumerations for submodel element value types[7].

The predefined types used to define the type of values of properties and other values use the names and the semantics of XML Schema Definition (XSD)[8]. Additionally, the type "langString" with the semantics as defined in the Resource Description Framework (RDF)[9] is used. "langString" is a string value tagged with a language code.

> Note 1: RDF[10] recommends to not use the following xsd data types. That is why they are excluded from the allowed data types.

- XSD BuildIn List types are not supported (ENTITIES, IDREFS and NMTOKENS).
- XSD string BuildIn types are not supported (normalizedString, token, language, NCName, ENTITY, ID, IDREF).
- The following XSD primitive types are not supported: NOTATION, QName.

> Note 2: the following RDF types are not supported: HTML and XMLLiteral.

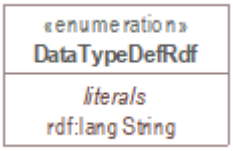


Figure 52. DefTypeDefRdf Enumeration

The enumeration is derived from Figure 54.

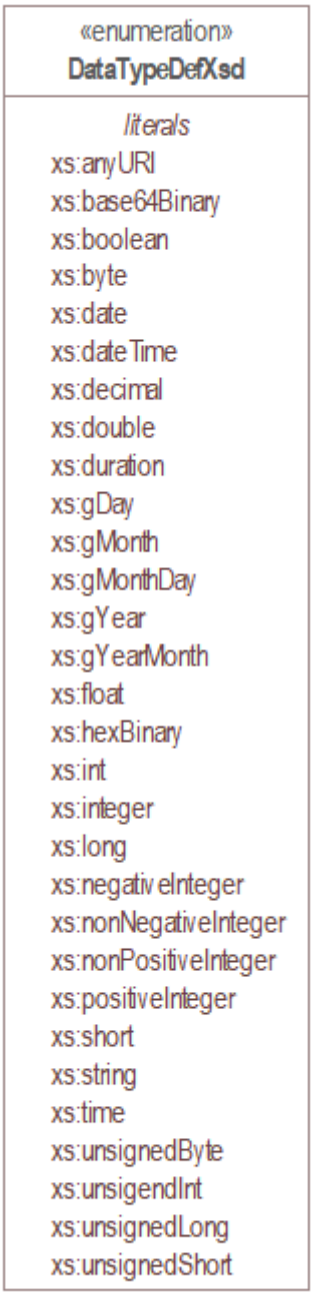


Figure 53. Data TypeDefXsd Enumeration

Table 9 depicts example values and the value range of the different data type"

shows the data types which can be used for submodel element values. The data types are defined according to the W3C XML Schema (https://www.w3.org/TR/xmlschema-2/#built-in-datatypes and

https://www.w3.org/TR/xmlschema-2/#built-in-derived). "Value Range" further explains the possible range of data values for this data type. The right column shows related examples for values of the corresponding data type.

Table 9. Data Types with Examples[11]

	Data Type	Value Range	Sample Values
Core types	xs:string	Character string (but not all Unicode character strings)	"Hello world" "¬¬¬¬¬¬¬¬ ¬¬¬¬¬" "¬¬¬¬¬¬¬"
	xs:boolean	true, false	true, false
	xs:decimal	Arbitrary-precision decimal numbers	-1.23 126789672374892739424.543233 +100000.00, 210
	xs:integer	Arbitrary-size integer numbers	-1 0 126789675432332938792837429837429837429 +100000
IEEE floating-point numbers	xs:double	64-bit floating point numbers incl. ±Inf, ±0, NaN	-1.0 +0.0 -0.0 234.567e8 -INF NaN

	Data Type	Value Range	Sample Values
	xs:float	32-bit floating point numbers incl. ±Inf, ±0, NaN	-1.0 +0.0 -0.0 234.567e8 -INF NaN
Time and dates	xs:date	Dates (yyyy-mm-dd) with or without time zone	"2000-01-01" "2000-01-01Z" "2000-01-01+12:05"
	xs:time	Times (hh:mm:ss.sss…) with or without time zone	"14:23:00" "14:23:00.527634Z" "14:23:00+03:00"
	xs:dateTime	Date and time with or without time zone	"2000-01-01T14:23:00" "2000-01-01T14:23:00.66372+14:00"[12]
Recurring and partial dates	xs:gYear	Gregorian calendar year	"2000" "2000+03:00"
	xs:gMonth	Gregorian calendar month	"--04" "--04+03:00"
	xs:gDay	Gregorian calendar day of the month	"---04" "---04+03:00"
	xs:gYearMonth	Gregorian calendar year and month	"2000-01" "2000-01+03:00"
	xs:gMonthDay	Gregorian calendar month and day	"--01-01" "--01-01+03:00"

	Data Type	Value Range	Sample Values
	xs:duration	Duration of time	"P30D" "-P1Y2M3DT1H", "PT1H5M0S"
Limited-range integer numbers	xs:byte	-128…+127 (8 bit)	-1, 0 127
	xs:short	-32768…+32767 (16 bit)	-1, 0 32767
	xs:int	2147483648…+2147483647 (32 bit)	-1, 0 2147483647
	xs:long	-9223372036854775808…+9223372036854775807 (64 bit)	-1 0, 9223372036854775807
	xs:unsignedByte	0…255 (8 bit)	0 1 255
	xs:unsignedShort	0…65535 (16 bit)	0 1 65535
	xs:unsignedInt	0…4294967295 (32 bit)	0 1 4294967295
	xs:unsignedLong	0…18446744073709551615 (64 bit)	0 1 18446744073709551615
	xs:positiveInteger	Integer numbers >0	1 734568374657836485736847563874 5

	Data Type	Value Range	Sample Values
	xs:nonNegativeInteger	Integer numbers ¬0	0 1 7345683746578364857368475 63
	xs:negativeInteger	Integer numbers <0	-1 • 2348726384762837648273648726 3
	xs:nonPositiveInteger	Integer numbers ¬0	-1 0 -9384583749857398749879898 73
Encoded binary data	xs:hexBinary	Hex-encoded binary data	"6b756d6f77617368657265"
	xs:base64Binary	Base64-encoded binary data	"a3Vtb3dhc2hlcmU="
Miscellaneous types	xs:anyURI	Absolute or relative URIs and IRIs	https://customer.com/demo/aas/1/1/1234859590 "urn:example:company:1.0.0"
	rdf:langString	Strings with language tags	"Hello"@en "Hallo"@de Note: this is written in RDF/Turtle syntax, @en and de are the language tags.

Enumeration:	DataTypeDefXsd
Explanation:	Enumeration listing all xsd anySimpleTypes For more details see https://www.w3.org/TR/rdf11-concepts/#xsd-datatypes
Set of:	—

Literal	Explanation
xs:anyURI	see: https://www.w3.org/TR/xmlschema11-2/#anyURI
xs:base64Binary	see: https://www.w3.org/TR/xmlschema11-2/#base64Binary
xs:boolean	see https://www.w3.org/TR/xmlschema11-2/#boolean
xs:byte	see https://www.w3.org/TR/xmlschema11-2/#byte
xs:date	see https://www.w3.org/TR/xmlschema11-2/#date
xs:dateTime	see https://www.w3.org/TR/xmlschema11-2/#dateTime
xs:decimal	see https://www.w3.org/TR/xmlschema11-2/#decimal
xs:double	see https://www.w3.org/TR/xmlschema11-2/#double
xs:duration	see https://www.w3.org/TR/xmlschema11-2/#duration
xs:float	see https://www.w3.org/TR/xmlschema11-2/#float
xs:gDay	see https://www.w3.org/TR/xmlschema11-2/#gDay
xs:gMonth	see https://www.w3.org/TR/xmlschema11-2/#gMonth
xs:gMonthDay	see https://www.w3.org/TR/xmlschema11-2/#gMonthDay
xs:gYear	see https://www.w3.org/TR/xmlschema11-2/#gYear
xs:gYearMonth	see https://www.w3.org/TR/xmlschema11-2/#gYearMonth
xs:hexBinary	see https://www.w3.org/TR/xmlschema11-2/#hexBinary
xs:int	see https://www.w3.org/TR/xmlschema11-2/#int
xs:integer	see https://www.w3.org/TR/xmlschema11-2/#integer
xs:long	see https://www.w3.org/TR/xmlschema11-2/#long
xs:negativeInteger	see https://www.w3.org/TR/xmlschema11-2/#negativeInteger
xs:nonNegativeInteger	see: https://www.w3.org/TR/xmlschema11-2/#nonNegativeInteger
xs:nonPositiveInteger	see: https://www.w3.org/TR/xmlschema11-2/#nonPositiveInteger
xs:positiveInteger	see: https://www.w3.org/TR/xmlschema11-2/#positiveInteger
xs:short	see: https://www.w3.org/TR/xmlschema11-2/#short
xs:string	see: https://www.w3.org/TR/xmlschema-2/#string
xs:time	see: https://www.w3.org/TR/xmlschema-2/#time
xs:unsignedByte	see: https://www.w3.org/TR/xmlschema11-2/#unsignedShort

Literal	Explanation
xs:unsignedInt	see: https://www.w3.org/TR/xmlschema11-2/#unsignedInt
xs:unsignedLong	see: https://www.w3.org/TR/xmlschema11-2/#unsignedLong
xs:unsignedShort	see: https://www.w3.org/TR/xmlschema11-2/#unsignedShort
xs:yearMonthDuration	see: https://www.w3.org/TR/xmlschema11-2/#yearMonthDuration

Enumeration:	DataTypeDefRdf
Explanation:	Enumeration listing all RDF types
Set of:	—

Literal	Explanation
rdf:langString	String with a language tag

RDF requires IETF BCP 47[13] language tags. Simple two-letter language tags for locales like "de" conformant to ISO 639-1 are allowed, as well as language tags plus extension like "de-DE" for country code, dialect, etc. like in "en-US" for English (United States) or "en-GB" for English (United Kingdom). IETF language tags are referencing ISO 639, ISO 3166 and ISO 15924.

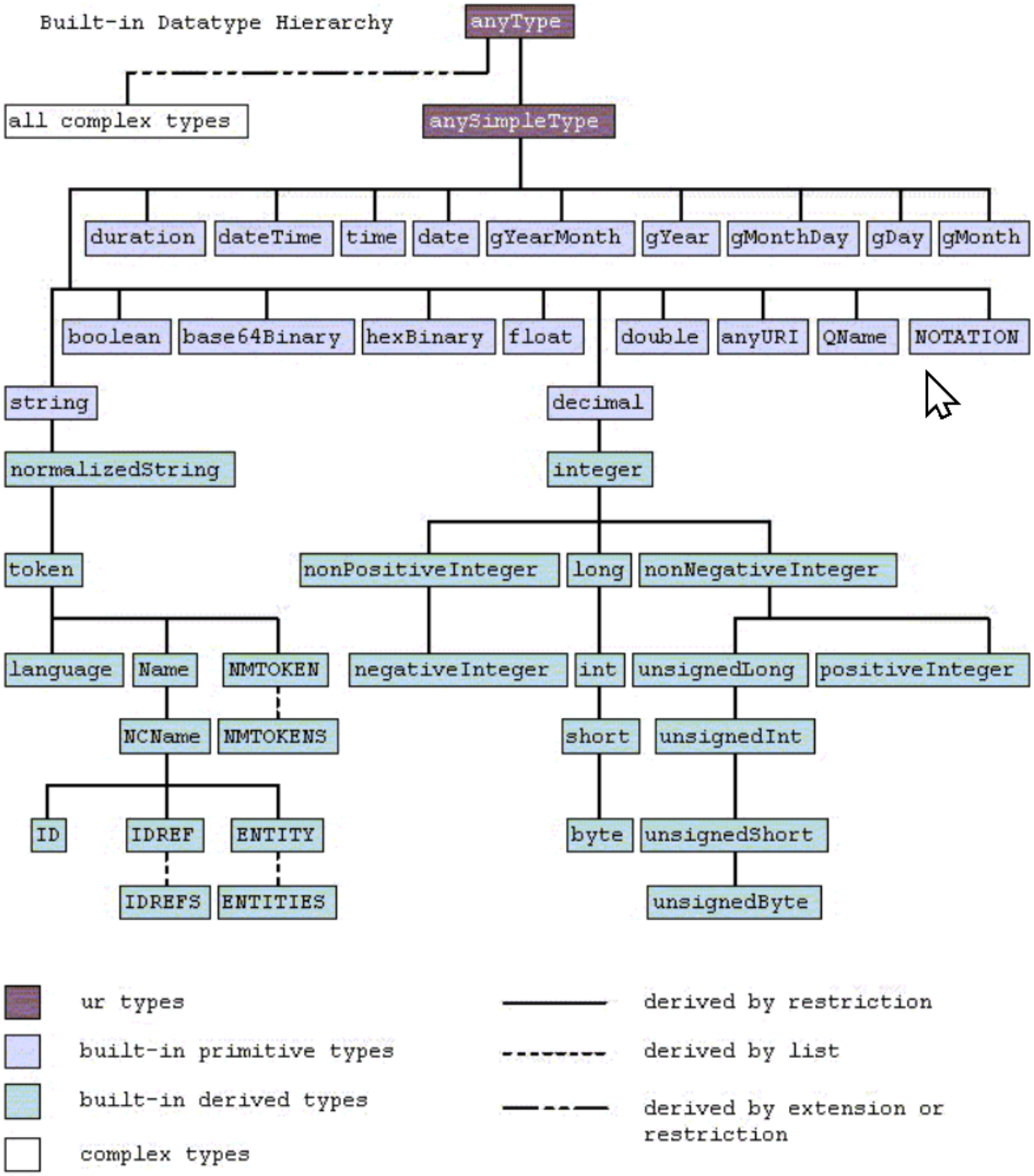


Figure 54. Built-In Types of XML Schema Definition 1.0 (XSD)[14]

5.3.12. Constraints: Global Invariants

Introduction

This clause documents constraints that represent global invariants, i.e. constraints that cannot be assigned to a single class.

In contrast, a class invariant is a constraint that must be true for all instances of a class at any time. They are documented as part of the class specification.

Constraints for Referables and Identifiables

Constraint AASd-117: *idShort* of non-identifiable *Referable*s not being a direct child of a *SubmodelElementList* shall be specified.

> Note: in other words (AASd-117), *idShort* is mandatory for all *Referable*s except for referables being direct childs of *SubmodelElementList*s and for all *Identifiable*s.

Constraint AASd-120: *idShort* of submodel elements being a direct child of a *SubmodelElementList* shall not be specified.

Constraint AASd-022: *idShort* of non-identifiable referables within the same name space shall be unique (case-sensitive).

> Note: AASd-022 also means that *idShort*s of referables shall be matched sensitive to the case.

Constraints for Qualifiers

Constraint AASd-021: Every qualifiable can only have one qualifier with the same *Qualifier/type.*

Constraint AASd-119: If any *Qualifier/kind* value of a *Qualifiable/qualifier* is equal to *TemplateQualifier* and the qualified element inherits from "*hasKind*", the qualified element shall be of kind *Template* (*HasKind/kind* = *"Template"*).

Constraint AASd-129: If any *Qualifier/kind* value of a *SubmodelElement/qualifie*r (attribute *qualifier* inherited via *Qualifiable*) is equal to *TemplateQualifier*, the submodel element shall be part of a submodel template, i.e. a *Submodel* with *Submodel/kind* (attribute *kind* inherited via *HasKind*) value equal to *Template.*

Constraints for Extensions

Constraint AASd-077: The name of an extension (*Extension/name*) within *HasExtensions* needs to be unique.

Constraints for Asset-Related Information

Constraint AASd-116: "*globalAssetId*" (case-insensitive) is a reserved key. If used as value for *SpecificAssetId/name, SpecificAssetId/value* shall be identical to *AssetInformation/globalAssetId*.

> Note: AASd-116 is important to enable a generic search across global and specific asset IDs.

Constraints for Types

Constraint AASd-130: an attribute with data type "string" shall consist of these characters only: ^[\x09\x0A\x0D\x20-\uD7FF\uE000-\uFFFD\u00010000-\u0010FFFF]*$.

Constraint AASd-130 ensures that encoding and interoperability between different serializations is possible. It corresponds to the restrictions as defined for the XML Schema 1.0[15].

[2] Note: categories of referables are deprecated.

[3] https://www.w3.org/XML/Core/, former https://www.w3.org/XML/Schema

[4] see: https://www.w3.org/TR/rdf11-concepts/

[5] Corresponds to xs:dateTimeStamp in XML Schema 1.1

[7] E.g. Property/valueType

[8] see https://www.w3.org/XML/Schema, https://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes

[9] see: https://www.w3.org/TR/rdf11-concepts/

[10] See https://www.w3.org/TR/rdf11-concepts/#xsd-datatypes

[11] See list of RDF-compatible XSD types with short description https://www.w3.org/TR/rdf11-concepts/#xsd-datatypes. Examples from https://openmanufacturingplatform.github.io/sds-bamm-aspect-meta-model/bamm-specification/v1.0.0/datatypes.html

[13] see https://tools.ietf.org/rfc/bcp/bcp47.txt

[14] Source: https://www.w3.org/TR/xmlschema-2/#built-in-primitive-datatypes

[15] https://www.w3.org/TR/xml/#charsets

Chapter 6. Data Specification Templates (normative)

6.1. Introduction

A data specification template specifies which additional attributes, which are not part of the metamodel, shall be added to an element instance. Typically, data specification templates have a specific scope. For example, templates for concept descriptions differ from templates for operations, etc. More than one data specification template can be defined and used for an element instance. *HasDataSpecification* defines, which templates are used for an element instance.

Figure 55 shows the concept of data specification for a predefined data specification conformant to IEC61360[4] that, for example, can be used for concept descriptions for single properties.

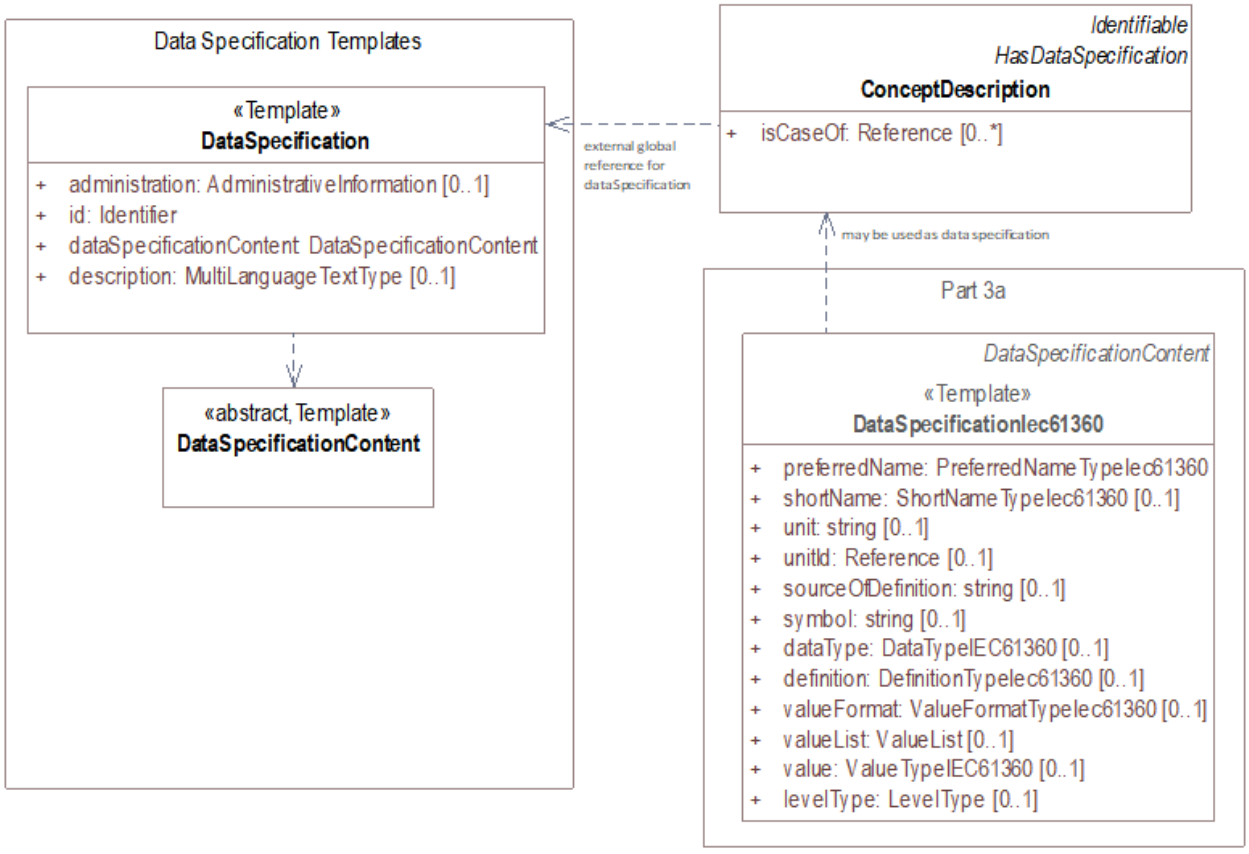


Figure 55. Core Elements of Using Data Specifications (non-normative)

The template introduced to describe the concept of a property, a value list, or a value is based on IEC 61360. Figure 55 also shows how concept descriptions and the predefined data specification templates are related to each other.

6.1.1. Data Specification Template Attributes

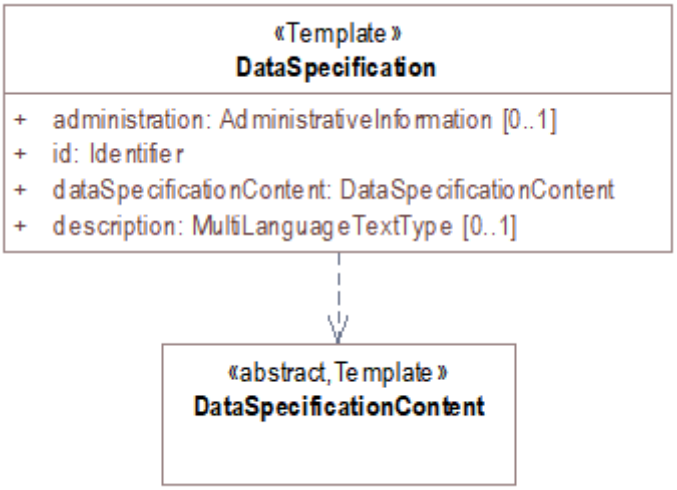


Figure 56. Data Specification Templates

Note: the data specification templates do not belong to the metamodel of the Asset Administration Shell. In serializations that choose specific templates, the corresponding data specification content may be directly incorporated.

It is required that a data specification template has a global unique ID so that it can be referenced via *HasDataSpecification/dataSpecification*.

A template consists of the *DataSpecificationContent* containing the additional attributes to be added to the element instance that references the data specification template, as well as meta information about the template itself. These are two separate classes in UML.

Class:	DataSpecification <<Template>>
Explanation:	Data specification template
Inherits from:	—

Attribute	Explanation	Type	Card.
administration	Administrative information of an identifiable element Note: some of the administrative information like the version number might need to be part of the identification.	AdministrativeInformation	0..1
id	The globally unique identification of the element	Identifier	1
dataSpecificationContent	The content of the template without meta data	DataSpecificationContent	1
description	Description of how and in which context the data specification template is applicable; can be provided in several languages.	MultiLanguageTextType	0..1

Class:	DataSpecificationContent <<Template>><<abstract>>
Explanation:	Data specification content is part of a data specification template and defines, which additional attributes shall be added to the element instance that references the data specification template and meta information about the template itself.
Inherits from:	—

Attribute	Explanation	Type	Card.

[4] Since the data specification templates are specified and maintained in separate documents, these templates are considered as examples only, although there is a similarity to existing data specifications.

Chapter 7. Mappings to Data Formats to Share I4.0-Compliant Information (normative)

7.1. General

It is crucial for Industry 4.0 applications to share information between different systems throughout the areas covered by the entire RAMI4.0 model [1] [2]. OPC UA is a frequently used format for information models in the domain of production operations, but there is a need for other formats for further areas and the relationships between them.

This document specifies the Asset Administration Shell in a technology-neutral format, UML. Different data formats are used or recommended to be used in the different life cycle phases of a product[4]. Serializations and mappings of the Asset Administration Shell are provided to cover the complete life cycle of each of these formats. Figure 57.

Table 10 explains the main purpose of each of the formats: OPC UA information models[5], AutomationML, XML, JSON, and RDF. The different purposes are visualized in Figure 57.

Table 10. Distinction of Different Data Formats for the Asset Administration Shell

Data format	Purpose / motivation
XML, JSON	Serialization of information for the technical communication between phases
RDF	Mapping of information to enable full use of the advantages of semantic technologies
AutomationML	Sharing of type and instance information about assets, particularly during engineering; transfer of this information into the operational phase (cf. OPC UA and the corresponding mapping)
OPC UA Information models	Access to all information of the administration data and sharing of live data within production operations; access for higher-level factory systems to this information

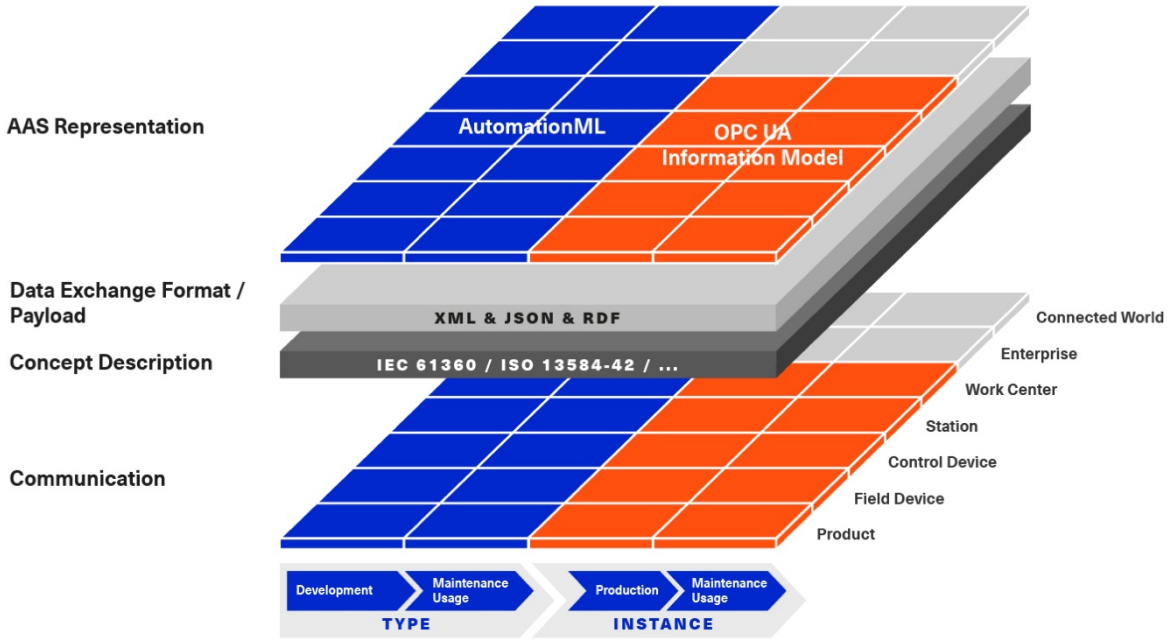


Figure 57. Graphic View on Exchange Data Formats for the Asset Administration Shell[6]

Note: the mapping specifications and schemata themselves are not part of the specification any longer but maintained open source. This eases usage of the specification and the different formats in open-source code projects.

7.2. General Rules

7.2.1. Introduction

There are some general rules that apply to all serializations or can be used in different serializations.

7.2.2. Encoding

Blobs require the following encoding: base64 string.

7.2.3. Serialization of Values of Type "Reference"

Some mappings or serializations convert the type "Reference" into a single string. In this case, the following serialization is required:

Grammar:

> **<Reference> ::= ["[" <ReferenceType> ["- " <referredSemanticId> " -"] "]"] <Key> \{(", " <Key> }***
>
> **<ReferenceType> ::= "ExternalRef" | "ModelRef" value of AAS:Reference/type**
>
> **<SemanticId> ::= ["[" <ReferenceType> "]"] <Key> \{(", " <Key> }***
> **value of AAS:Reference/referredSemanticId**
>
> **<Key> ::= "(" <KeyType> ")" <KeyValue>**
>
> **<KeyType> ::= value of AAS:Key/type**
>
> **<KeyValue> ::= value of AAS:Key/value**

Note 1: an IRI may also contain special symbols like "(", "," and "[". A blank is added before the new key or value to distinguish beginning and end of a new key.

Note 2: *ReferenceType* is optional. It is clear from the first key in the key chain whether the reference is a global or a model reference. The examples in this document therefore do not use this prefix.

Valid Examples:

References:

> **(GlobalReference)0173-1#02-BAA120#008**
>
> **[ExternalRef](GlobalReference)0173-1#02-BAA120#008**
>
> **(Submodel)https://example.com/aas/1/1/1234859590, (SubmodelElementList)Documents, (SubmodelElementCollection)0, (MultiLanguageProperty)Title**

Model References:

> **(ConceptDescription)0173-1#02-BAA120#008**

> **[ModelRef](ConceptDescription)0173-1#02-BAA120#008**
>
> **(Submodel)https://example.com/aas/1/1/1234859590,**
> **(Property)Temperature**
>
> **[ModelRef- (ConceptDescription)0173-1#02-BAA120#008**
>
> **-](Submodel)https://example.com/aas/1/1/1234859590**

7.2.4. Semantic Identifiers for Metamodel and Data Specifications

Rules for creating identifiers are defined to enable the unique identification of concepts as used and defined in the metamodel of the Asset Administration Shell.

The following grammar is used to create valid identifiers:

> **<Namespace> ::= <AAS Namespace> | <Data Specification Namespace>**
>
> **<Namespace Qualifier> ::= <AAS Namespace Qualifier> | <Data Specification Qualifier>**
>
> **<AAS Namespace> ::= <Shell-Namespace> "/aas/" <Version>**
>
> **<Data Specification Namespace> ::= <Shell-Namespace> "/DataSpecifications/" <idShort of Data Specification> <Version>**
>
> **<Shell-Namespace> ::= "https://admin-shell.io/"**
>
> **<Version> ::= \{<Digit>}+ "/" \{<Digit>}+ ["/" \{<Character>}+]**
>
> **<Digit> ::= "0"| "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"**
>
> **<Character> ::= characters conformant to regular expression [a..zA..Z-]**

Up to this point, two data specification templates are defined. Now we need to define which data specification namespace is to be used for every data specification.

> **<AAS Namespace Qualifier> ::= "AAS:"**
>
> **<Data Specification Qualifier> ::= defined per Data Specification**

A concrete unique identifier is defined as follows:

> **<AAS Unique Concept Identifier> ::= (<Namespace> | <Namespace Qualifier>) "/" <AAS Concept Identifier>**
>
> **<AAS Concept Identifier> ::= <AAS Class Name> [(<AAS Attribute> | <AAS Enumeration >)]**
>
> **<AAS Attribute> ::= "/" <AAS Attribute Name> [\{ "/" <AAS Attribute Name>}**]*
>
> **<AAS Enumeration> ::= [\{ "/" <AAS Attribute Name>}]** "/" <AAS Enumeration Value>*

Examples for valid unique Asset Administration Shell concept identifiers:

> **https://admin-shell.io/aas/2/0/AssetAdministrationShell/administration/version**
>
> **AAS:AssetAdministrationShell/administration/version**
>
> **AAS:AssetInformation/assetKind/Instance**

The application of the pattern is explained as follows:

The concept identifier of a Class follows the pattern

> <*AAS Class name>*

This also applies to abstract classes and types including enumerations.

Valid examples:

> **AAS:Submodel**
>
> **AAS:Qualifier**
>
> **AAS:Reference**
>
> **AAS:ContentType**

> **AAS:KeyTypes**

Attributes of classes are separated by "/". Inherited attributes can also be referenced in this way, if the concrete referable is important in the context.

Basic Pattern:

> **<AAS Class name>"/"<AAS Attribute Name>**

Examples[7]:

> **AAS:Referable/idShort**
>
> **AAS:Property/idShort**
>
> **AAS:Qualifier/semanticId**

This also applies to attributes of attributes, if the cardinality of the attributes involved is not greater than 1:

> **<AAS Class Name> "/" <AAS Attribute Name> [\{ "/" <AAS Attribute Name>}**]*

Valid examples:

> **AAS:Identifiable/administration/version**

This also applies to values of enumerations:

> **<AAS Class Name>[\{ "/" <AAS Attribute Name>}]**["/" <AAS Enumeration Value>]*

Valid examples:

> **AAS:Key/type/Submodel**
>
> **AAS:AasSubmodelElements/Submodel**

In case of an attribute with a cardinality greater than 1, no further attributes or enumeration values can be added.

Note: although the attribute name in UML is always singular, the attribute name is annotated by the plural "s" if the cardinality is > 1.

Valid examples:

> **AAS:Operation/InputVariables**
>
> **AAS:AssetAdministrationShell/submodels**
>
> **AAS:Submodel/submodelElements**

Invalid examples:

> **AAS:AssetAdministrationShell/submodels/administration/version**
>
> **AAS:Submodel/Property/idShort**

These semantic identifiers are used as values for the *RefSemantic* attribute in AutomationML Mapping of the Asset Administration Shell. They are also used in OPC UA to describe the semantics of the metamodel via the OPC UA *HasDictionaryEntry* reference type.

Additional identifiers might be needed for specific serializations and mappings, e.g. for a set of Asset Administration Shells or a set of available concept descriptions. Here, the Asset Administration Shell metamodel and specification does not give any recommendations.

Data specification handling is special. Data specification templates do not belong to Part 1 of the Asset Administration Shell. However, serializations only support the predefined data specification templates as stipulated in this specification series, Part 3. Their corresponding name space qualifiers are defined individually.

Examples:

In xml and JSON, data specifications are embedded into the schema itself using the attribute "embeddedDataSpecification". Here, no concept identifier shall be used. For example,

> **AAS:ConceptDescription/embeddedDataSpecifications**

is not a valid concept identifier. *AAS:DataSpecificationContent* is a valid concept identifier.

7.2.5. Embedded Data Specifications

This specification predefines data specifications that can be used within an Asset Administration Shell to ensure interoperability.

Consequently, some serializations or mappings support exactly the data descriptions defined in this specification, although the metamodel as such is more flexible and would also support proprietary data specifications.

In the case of restricted use of data specifications, we speak of "embedded data specifications". Figure 58 explains the realization: instead of a set of external global references to externally defined data specifications, a set of pairs consisting of an external global reference to a data specification and the data specification content itself are directly "embedded". Here, the data specification content belongs to the schema, while the data specification including its content are not part of the schema in the general concept. This is similar to the concept of *semanticIds*: either it is an external global reference to an external concept dictionary, or it is a reference to a concept description within the schema. However, there is only one reference allowed for *semanticId*, whereas a set of data specification references is permitted for data specifications.

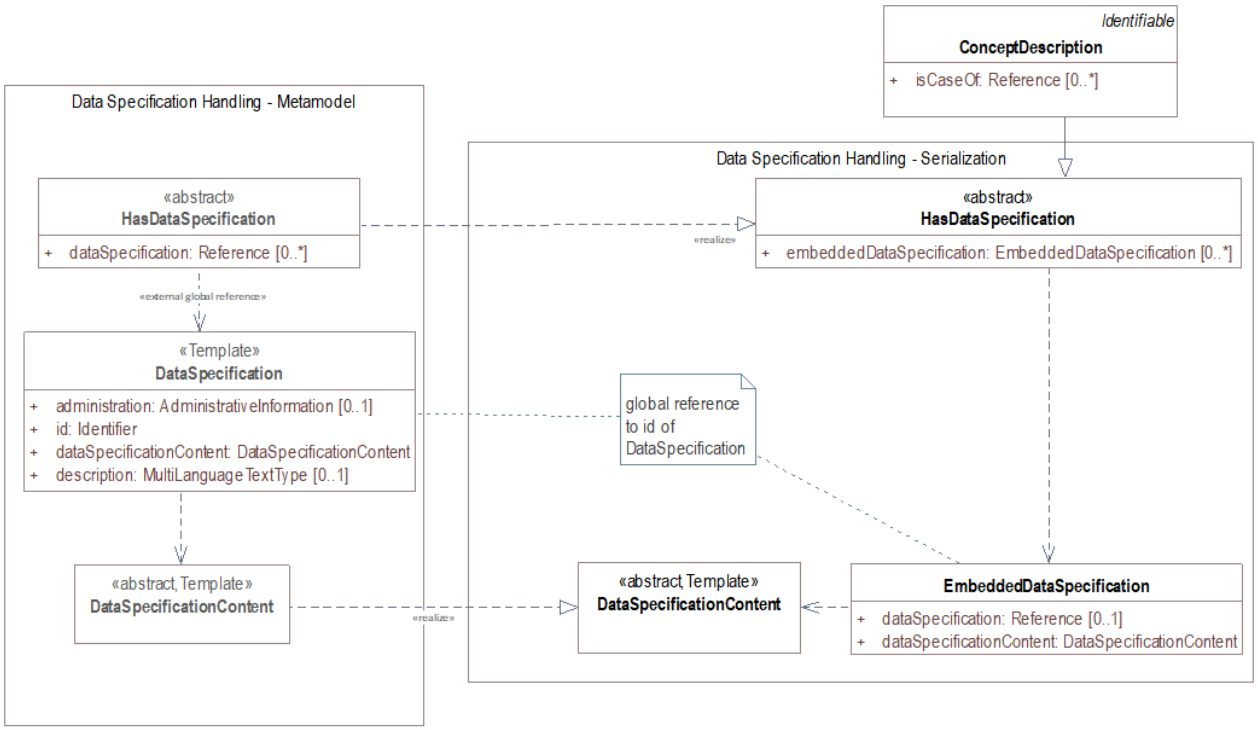


Figure 58. Realization of Embedded Data Specifications

7.3. XML

The metamodel of an Asset Administration Shell needs to be serialized for import and export scenarios. XML is a possible serialization format.

eXtensible Markup Language (XML[8]) is very well suited to derive information from an IT system, e.g. to process it manually and then feed it into another IT system. It meets the needs of the information sharing scenario defined in the leading picture in Clause 4.1. XML provides the possibilities of scheme definitions, which can be used to syntactically validate the represented information in each step.

Note 1: the xml schema (.xsd files) is maintained in the repository "aas-specs" of the github project admin-shell-io [51]: aas-specs-3.0\schemas\xml

Note 2: the mapping rules of how to derive the xml schema from the technology-neutral metamodel as defined in this specification can be found here: aas-specs-3.0\schemas\xml\Readme.md#xml-mappingrules

Note 3: example files can be found here: aas-specs-3.0\schemas\xml\examples

7.4. JSON

JSON[9] (JavaScript Object Notation) is a further serialization format that serializes the metamodel of an Asset Administration Shell for import and export scenarios.

Additionally, JSON format is used to describe the payload in the http/REST API for active Asset Administration Shells [37].

Note 1: the JSON schema (.json files) is maintained in the repository "aas-specs" of the github project admin-shell-io [51]: aas-specs-3.0\schemas\json

Note 2: the mapping rules of how to derive the JSON schema from the technology-neutral metamodel as defined in this specification can be found here: aas-specs-3.0\schemas\json\Readme.md#json-mappingrules

Note 3: example files can be found here: aas-specs-3.0\schemas\json\examples

7.5. RDF

The Resource Description Framework (RDF) [32] is the recommended standard of the W3C to unambiguously model and present semantic data. RDF documents are structured in the form of triples,

consisting of subjects, relations, and objects. The resulting model is often interpreted as a graph, with the subject and object elements as the nodes and the relations as the graph edges.

RDF is closely related to web standards, illustrated by the fact that all elements are encoded using (HTTP-)IRIs. As a common practice, the provision of additional information at the referenced location of an RDF entity directly allows the interlinking of entities[10] based on the web. This process – following links in order to discover related information – is called dereferencing a resource and is supported by any browser or web client. Connecting distributed data sources through the web in the described manner is referred to by the term “Linked Data”. Connecting the available resources and capabilities of linked data with the expressiveness of the Asset Administration Shell is one motivation for the RDF serialization.

In addition, RDF is the basis of a wide range of logical inference and reasoning techniques. Vocabularies like RDF Schema (RDFS) and the Web Ontology Language (OWL) combine the graph-based syntax of RDF with formal definitions and axioms. This allows automated reasoners to understand the relation between entities to some extent and draw conclusions.

Combining both features, the RDF mapping of the Asset Administration Shell can provide the basis for complex queries and requests. SPARQL, the standard query language for the Semantic Web, can combine reasoning features with the integration of external data sources. In order to benefit of these abilities, the Asset Administration Shell requires a clear scheme of its RDF representation.

> Note 1: the RDF scheme/OWL files (.ttl files) are maintained in the repository "aas-specs" of the github project admin-shell-io [51]: aas-specs-3.0\schemas\rdf

> Note 2: the mapping rules of how to derive the RDF schema from the technology-neutral metamodel as defined in this specification can be found here: aas-specs-3.0\schemas\rdf\Readme.md#rdf-mappingrules

> Note 3: example files can be found here: aas-specs-3.0\schemas\rdf\examples

7.6. AutomationML

AutomationML (IEC 62714) is especially suited for serializing the import and export scenarios of the metamodel of an Asset Administration Shell in the engineering phase.

In general, the serialization approach is to map each object of the Asset Administration Shell metamodel to an AutomationML Role Class or to an AutomationML Role Class accompanied by an AutomationML Interface Class. This role class and (if applied) interface class also define the required attributes in AutomationML.

Asset Administration Shells themselves shall be modelled as AutomationML System Unit Classes or as Internal Elements within an Instance Hierarchy depending of the kind information of type and instance.

For the role classes and interface classes that are required for serialization, an AutomationML Role Class Library resp. an Interface Class Library are defined and provided to the public.

One of the goals is to ensure that the AutomationML model of the Asset Administration Shell can be used as a stand-alone AutomationML model, as well as in combination with existing AutomationML models such as the upcoming AutomationML Component Description. Therefore, the definition of the serialization approach defined in this clause is incorporated with the AutomationML definitions and applies the AutomationML technology definitions available on https://www.automationml.org/o.red.c/dateien.html.

[28] is the AutomationML application recommendation for the Asset Administration Shell (AR AAS).

> Note 1: the mapping of the Asset Administration Shell to AutomationML is carried out in a joint working group between AutomationML e.V. and Plattform Industrie 4.0.

> Note 2: the resulting application recommendation (AR 004E) "Asset Administration Shell (AAS) Representation" [28] can be found here: https://www.automationml.org/download-archive/, together with .aml files

7.7. OPC UA

OPC UA is suited for the operating phase of Asset Administration Shells[11] and especially applicable in case of machine-to-machine communication. The information model [42] is the basis for the definition of so-called OPC UA Information Models, or OPC UA Companion Specifications [43].

Note1 : the mapping to the OPC UA are carried out in a joint working group[12] "I4AAS" between OPC Foundation, ZVEI and VDMA (https://opcfoundation.org/markets-collaboration/I4AAS/ [30].

> Note 2: the different versions of the OPC UA Companion Specification for I4AAS can be found here: https://reference.opcfoundation.org/<version>/I4AAS/<version>/docs/, e.g. https://reference.opcfoundation.org/v104/I4AAS/v100/docs/ for release 1.00 [41].

[4] The word “data formats” is used as shortcut and includes the use of conceptual advantages such as information models, schemes, transmission protocols, etc.

[5] OPC UA Information Models should not be mixed up with the Client/Server or Pub/Sub Protocol of OPC UA. This would belong to the communication but not to the Asset Administration Shell representation layer.

[6] Only data formats considered in this document so far are mentioned in the figure.

[7] For simplicity reasons, most examples use the namespace qualifier and not the full path of the namespace.

[8] see: https://www.w3.org/TR/2008/REC-xml-20081126/

[9] see: https://tools.ietf.org/html/rfc8259 or https://www.ecma-international.org/publications/standards/Ecma-404.htm

[10] Note: entity as a generic term and entity as a specific submodel element subtype need tobe distinugished.

[11] OPC UA is used for several purposes in the context of Industry 4.0. In this document, however, the focus is on Asset Administration Shell representation only.

[12] see: https://opcfoundation.org/collaboration/i4aas/

Chapter 8. Summary and Outlook

This document has defined the metamodel for the structural viewpoint of the Asset Administration Shell using the technology-neutral modelling language UML.

Several serializations and mappings are offered for the Asset Administration Shell based on this specification:

- XML and JSON for Exchange between partners via the exchange format *.aasx,*
- RDF for reasoning,
- AutomationML for the engineering phase,
- OPC UA for the operation phase.

Additional parts of the document series cover (see [37]):

- interfaces and APIs for accessing the information of Asset Administration Shells (access, modify, query, and execute information and active functionality), Part 2; the payload of these APIs is based on the definitions of the information model in this document, Part 1,
- predefined data specification templates (Part 3 series), for example for concept descriptions of properties conformant to IEC61360 (Part 3a),
- the infrastructure, which hosts and interconnects multiple Asset Administration Shells, implementing registry, discovery services, endpoint handling, and more,
- security aspects including access control, Part 4.

Appendix A: Concepts of the Administration Shell

A.1. General

Annex A provides general information about sources of information and relevant concepts for the Asset Administration Shell. Some of these concepts are explained in a general manner. Some concepts are update in order to reflect actual design decisions. No new concepts are introduced. Thus, this clause can be seen as a fully informative annex to the specifications of the Administration Shell.

A.2. Relevant Sources and Documents

The following documents were used to identify requirements and concepts for the Administration Shell:

- implementation strategy of Plattform Industrie 4.0 [1][2],
- aspects of the research roadmap in application scenarios [7],
- continuation of the application scenarios [8],
- structure of the Administration Shell [4] [19],
- examples for the Administration Shell of the Industrie 4.0 Components [6],
- technical overview "Secure identities" [9],
- security of the Administration Shell [15],
- relationships between I4.0 components – composite components and smart production [13].

Note 1: the global Plattform Industrie 4.0 glossary can be found at: https://www.plattform-i40.de/PI40/Navigation/EN/Industrie40/Glossary/glossary.html

Note 2: the online library of the Plattform Industrie 4.0 can be found at: https://www.plattform-i40.de/PI40/Navigation/EN/Downloads-News/downloads-news.html

Note 3: the online library of the Industrial Digital Twin Association can be found at: https://industrialdigitaltwin.org/en/content-hub/downloads

A.3. Basic Concepts for Industry 4.0

Industry 4.0 describes concepts and definitions for the domain of smart manufacturing. For Industry 4.0, the term asset, being any "object which has a value for an organization", is of central importance [2] [21]. Industry 4.0 assets can take almost any form, e.g. a production system, a product, a software installation, intellectual properties, or even human resources.

According to [21], the "reference architecture model Industry 4.0 (RAMI4.0) provides a structured view of the main elements of an asset using a level model consisting of three axes […]. Complex interrelationships can thus be broken down into smaller, more manageable sections by combining all three axes at each point in the asset's life to represent each relevant aspect."

Assets shall have a logical representation in the "information world", e.g. managed by IT systems. Consequently, an asset needs a precise identification as an entity, shall have a "specific state within its life (at least a type or instance)", shall have communication capabilities, shall be represented by means of information and shall be able to provide technical functionality [21]. This logical representation of an asset is called Administration Shell [4]. The combination of asset and Administration Shell forms the so-called I4.0 component. In international papers [19], the term smart manufacturing replaces the term Industry 4.0.

As far as the large variety of assets in Industry 4.0 are concerned, the Asset Administration Shell allows these assets to be handled in the same manner within the information world. This reduces complexity and allows for scalability. Additional motivation can be found in [2] [4] [7] [8].

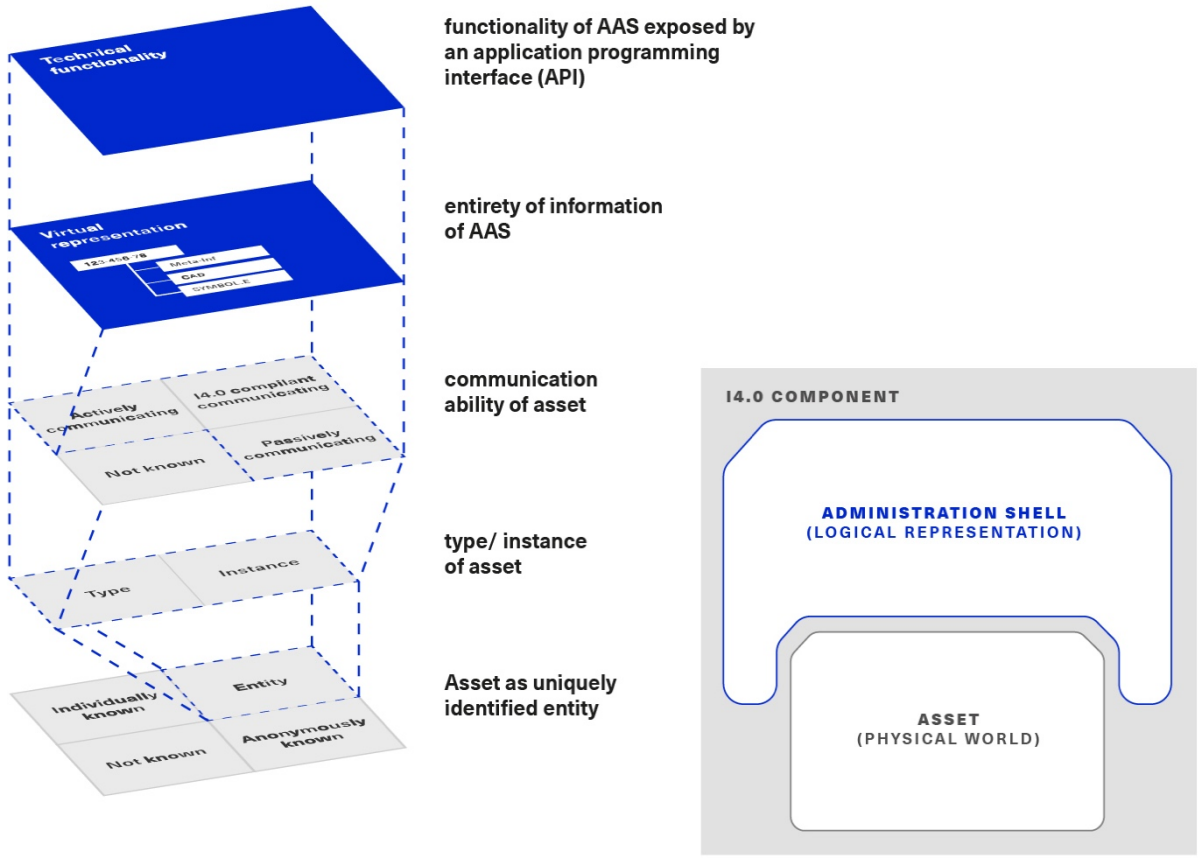


Figure 59. Important Concepts of Industry 4.0 attached to the Asset [2] [21]

A.4. The Concept of Properties

According to [20], the "IEC 61360 series provides a framework and an information model for product dictionaries. The concept of product type is represented by 'classes' and the product characteristics are represented by 'properties'".

Standardized data elements are an example for such properties. The definitions can be found in a range of repositories, such as IEC CDD (common data dictionary) or ECLASS. The definition of a property (aka standardized data element type, property type) associates a worldwide unique identifier with a definition, which is a set of well-defined attributes. Relevant attributes for the Administration Shell are, amongst others, the preferred name, the symbol, the unit of measure, and a human-readable textual definition of the property.

Code:	0112/2///62683#ACE424
Version:	001
Revision:	01
IRDI:	0112/2///62683#ACE424#001
Preferred name:	rated current
Synonymous name:	
Symbol:	In
Synonymous symbol:	
Short name:	
Definition:	maximum uninterrupted current equal to the conventional free-air thermal current (Ith)
Note:	
Remark:	
Primary unit:	A
Alternative units:	
Level:	
Data type:	LEVEL(MAX) OF REAL_MEASURE_TYPE

Figure 60. Exemplary Definition of a Property the IEC CDD

The instantiation of such a definition (just 'property', property instance) typically associates a value to the property. This mechanism makes it possible to convey semantically well-defined information to the Administration Shell.

> Note: Industry 4.0 and smart manufacturing in general will require many properties, which are beyond the current scope of IEC CDD, ECLASS, or other repositories. It is expected that these sets of properties will be introduced, as more and more domains are modelled and standardized (see next clause).

A.5. The Concept of Submodels

"The Administration Shell is the standardized digital representation of the asset, corner stone of the interoperability between the applications managing the manufacturing systems" [19]. Hence, it should provide a minimal but sufficient description according to the different application scenarios in Industry 4.0 [7] [8]. Many different (international) standards, consortia and manufacturer specifications can already contribute to this description [19].

As the figure shows, information from many different domains can be associated with a respective asset, and many different properties are required to be represented in Administration Shells of future I4.0 components. The architectural principle "separation of concerns" is supported by submodels.

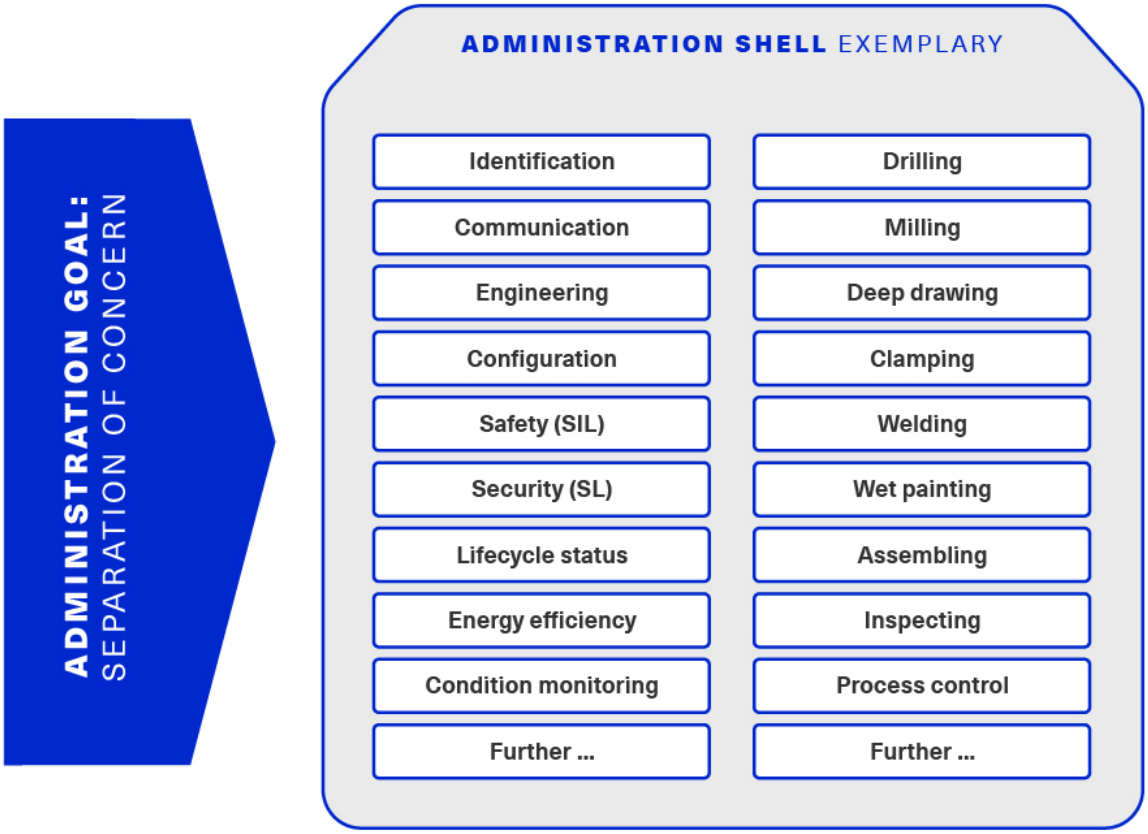


Figure 61. Examples of Different Domains Providing Properties for Submodels of the Administration Shell

The Administration Shell is made up of a series of submodels [4]. These represent different aspects of the asset concerned. For example, they may contain a description relating to safety or security [15] but they could also outline various process capabilities such as drilling or installation [6].

From an interoperability perspective, the goal is to standardize only a single submodel for each aspect/ technical domain. For example, it will be possible to find a drilling machine by searching for an Administration Shell containing a submodel "Drilling" with appropriate properties. Certain properties can then be assumed to exist for communication between different I4.0 components. In our example, a second submodel "energy efficiency" could make sure the drilling machine is able to cut its electricity consumption when out of operation.

> Note: a side benefit of the Administration Shell will be to simplify the update of properties from product design (and in particular system design) tools, update properties from real data collected in the instances of assets, improve traceability of assets along the life cycle, and help certify assets from data.

A.6. Basic Structure of the Asset Administration Shell

The document on the structure of the Asset Administration Shell [4] [19] presents a rough, logical view of the Asset Administration Shell's structure. The Asset Administration Shell – shown in blue in the following figure

– comprises different sets of information. Both the asset and the Administration Shell are identified by a globally unique identifier. It comprises a number of submodels, which characterize the Asset Administration Shell.

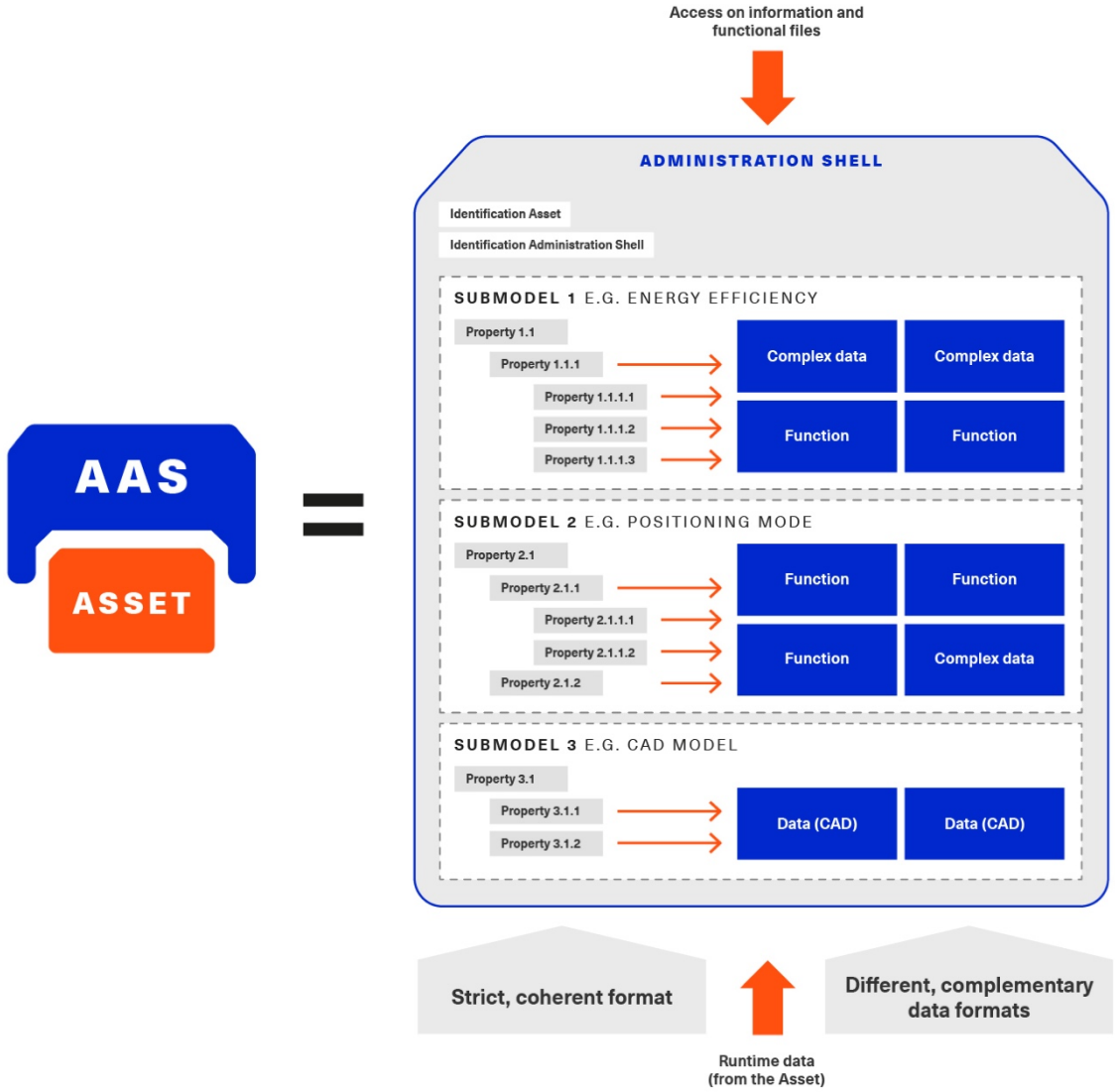


Figure 62. Basic Structure of the Asset Administration Shell

Properties, data, and functions will contain information that not every partner within a value-added network or even within an organizational unit should be able to access; its integrity and availability should be guaranteed. Therefore, the structure of the Administration Shell shall be able to handle aspects such as access protection, visibility, identity, and rights management, confidentiality, and integrity. Information security needs to be respected and has to be aligned with an overall security concept. Security implementation must go together with the implementation of other components of an overall system.

Each submodel contains a structured quantity of properties that can refer to data and functions. A standardized format based on IEC 61360-1/ ISO 13584-42 is envisaged for the properties. Property value definition shall follow the same principles as ISO 29002-10 and IEC 62832-2. Data and functions may be available in various complementary formats.

The properties of all the submodels therefore result in a constantly readable key information directory of the Administration Shell and hence of the I4.0 component. To enable binding semantics, Administration Shells, assets, submodels, and properties must all be clearly identified. For identification of these element the following types of global identifiers are allowed: IRDIs (used for example in ISO TS 29002-5, ECLASS and IEC CDD) and IRIs (Internationalized Resource Identifier, used for example in ontologies).

It should be possible to filter elements of the Administration Shell or submodels according to different given views (see example C.4 in [19]). This facilitates different perspectives or use cases to access the Administration Shell's information.

A.7. How Are New Identifiers Created?

Following the different identification types from Clause 4.3.4, it can be stated that:

a. IRDIs are assumed to already exist due to an external specification and standardization process in the creation of a certain Administration Shell. To bring such IRDI identifiers to life, please refer to Clause 5 of this document [4].

b. URIs and URLs can easily be created by developers when forming a certain Administration Shell. All they need is a valid authenticated URL, for example of the company. They also need to make sure that the domain (e.g. admin-shell.io) appended to the host's name is reserved in a semantically unique way for these identifiers. This way, each developer can create an arbitrary URI or URL by combining the host name and some chosen path, which only needs to be unique in the developer's organization.

c. Custom identifiers can also be easily formed by developers. They only need to make sure that internal custom identifiers can be clearly distinguished from (a) or (b).

d. Local identifiers can also be created on the fly. They have to be unique within their namespace.

A.8. Best Practice for Creating URI Identifiers

The approach for semantics and interaction for I4.0 components [18] suggests the use of the following structure (see Table 11) for URIs[4], which is slightly modified here. The idea is to always structure URIs following a scheme of different elements. However, this is just a recommendation and by no means mandatory.

Table 11. Proposed Structure for URIs

Element	Description	Syntax component
Organization	Legal body, administrative unit, or company issuing the ID	A

Element	Description	Syntax component
Organizational subunit/document ID/document subunit	Sub entity in organization above, released specification, or publication of organization above	P
Submodel/domain ID	Submodel of functional or knowledge-wise domain of asset or Administration Shell, which the identifier belongs to	P
Version	Version number in line with release of specification or publication of identifier	P
Revision	Revision number in line with release of specification or publication of identifier	P
Property/element ID	Property or further structural element ID of the Administration Shell	P
Instance number	Individual numbering of the instances within release of specification or publication	P

In the table, syntax component "A" refers to authority of RFC 3986 (URI) and namespace identifier of RFC 2141 (URN); "P" refers to path of RFC 3986 (URI) and namespace specific string of RFC 2141 (URN).

Grammar:

<AAS URI> ::= <scheme> ":" <authority> [<path>]

<scheme> ::= a valid URI scheme

<authority> ::= Organization

<path> ::= <subunit> <domain> <release> <element>

<subunit> ::= \{ ("/" | ":") <Organizational Subunit/Document ID/Document subunit> }*

<domain> ::= [("/" | ":") <Submodel / Domain-ID>

<release> ::= [("/" | ":") <Version> [("/" | ":") <Revision>]]*

<element> ::= [("/" | ":" | "#") \{(<Property/Element-ID> | <Instance number>)}]*

Using this scheme, valid URNs and URLs can be both created as URIs. The latter are preferred for Administration Shells, as functionality (such as REST services) can be bound to the identifiers. Examples of such identifiers are given in Table 12.

Table 12. Example URN and URL-based Identifiers of the Asset Administration Shell

Identifier	Examples
Administration Shell ID	urn:zvei:SG2:aas:1:1:demo11232322 https://www.zvei.de/SG2/aas/1/1/demo11232322
Submodel ID (Template)	urn:GMA:7.20:contractnegotiation:1:1 http://www.vdi.de/gma720/contractnegotiation/1/1
Submodel ID (Instance)	urn:GMA:7.20:contractnegotiation:1:1#001 http://www.vdi.de/gma720/ contractnegotiation/1/1#001
ID of type or Concept Description of a Property etc.	urn:PROFIBUS:PROFIBUS-PA:V3-02:Parameter:1:1:MaxTemp https://www.zvei.de/SG2/aas/1/1/demo11232322/maxtemp
Property, etc. * (not used by metamodel)*	urn:PROFIBUS:PROFIBUS-PA:V3-02:Parameter:1:1:MaxTemp#0002 https://www.zvei.de/SG2/aas/1/1/demo11232322/maxtemp#0002

Note: the last row of Table 12 is only used for completion; the metamodel does not foresee own unique identifiers for property/parameter/status instances.

[4] URLs are also URIs

Appendix B: Requirements

This annex collects the requirements from various documents that have impact on the specific structure of the Administration Shell. They serve as input for the specific description of the structures of the Administration Shell.

The following requirements are taken from the document "Implementation strategy of Plattform Industrie 4.0" [2][4]. They are marked "STRAT". The "Tracking" column validates the requirements by linking to features of the UML metamodel or this document in general.

ID	Requirement	Tracking
STRAT#1	A network of Industry 4.0 components must be structured to ensure that connections between any end points (Industry 4.0 components) are possible. The Industry 4.0 components and their contents are to follow a common semantic model.	Network possible but not scope of this part of the document series Common semantic model realized by domain-specific submodels (*HasSemantics*/ *ConceptDescription* and by *RelationshipElements*)
STRAT#2	It must be possible to define the concept of an Industry 4.0 component so that it can meet requirements with different focal areas, i.e. "office floor" or "shop floor".	Content-wise, many different submodels possible
STRAT#3	Industry 4.0-compliant communication must be performed in a way that enables the data of a virtual representation of an Industry 4.0 component to be kept either in the object itself or in a (higher level) IT system.	Metamodel and information representation independent of any deployment scenario
STRAT#4	In the case of a virtual representation of an I4.0 component in a higher-level system, the integrity must be ensured between the asset and its representation.	Integrity part of security approach
STRAT#5	A suitable reference model must be established to describe how a higher-level IT system can make the Administration Shell available in an Industry 4.0-compliant manner (SOA approach, delegation principle).	Scope of upcoming part of the document series; not scope of this part
STRAT#6	A description is required of how the Administration Shell can be "transported" from the originator (e.g. component manufacturer or electrical designer) to the higher-level IT system (e.g. as an attachment to an email).	Hierarchical representation by XML/JSON and package file format allows for different transport scenarios

ID	Requirement	Tracking
STRAT#7	Depending on the nature of the higher-level systems, it may be necessary for the administration objects to allow for deployment in more than one higher-level IT system.	Metamodel and information representation independent of any deployment scenario
STRAT#8	The Industry 4.0 component, the Administration Shell, its inherent functionality, and the protocols concerned are to be "encapsulation-capable" or "separable" from any field busses in use.	Metamodel and information representation independent of any communication scenario
STRAT#9	The aim of the Industry 4.0 component is to detect non-Industry 4.0-compliant communication relationships leading to or from the object's Administration Shell and to make them accessible to end-to-end engineering.	Non-Industry 4.0-compliant communication relationships could be modelled and made available by submodels
STRAT#10	It should be possible to logically assign other Industry 4.0 components to one Industry 4.0 component (e.g. an entire machine) to ensure (temporary) nesting.	*References, Entity, RelationshipElements* (see *Composite components* [12])
STRAT#11	Higher-level systems should be able to access all Industry 4.0 components in a purpose-driven and restricted manner, even when these are (temporarily) logically assigned.	Scope of upcoming part of the document series; not scope of this part
STRAT#12	Characteristics (1) identifiability	Given by *Identifiable*
STRAT#13	Characteristics (2) I4.0-compliant communication	Not scope of part 1
STRAT#14	Characteristics (3) I4.0-compliant services and multiple status	Standardization of submodels
STRAT#15	Characteristics (4) virtual description	Available by digital representation (*Submodel* and *SubmodelElement*)
STRAT#16	Characteristics (5) I4.0-compliant semantics	*HasSemantics*
STRAT#17	Characteristics (6) security and safety	Security through attribute-based and role-based access control Not scope of this part, security upcoming as Part 4
STRAT#18	Characteristics (7) quality of services	Metamodel and information representation independent of any communication scenario

ID	Requirement	Tracking
STRAT#19	Characteristics (8) status	Standardization of *submodels*
STRAT#20	Characteristics (9) nestability	Supported by *Submodels* representing a *Bill of Material* of an asset, *Entities* and *RelationshipElements*
STRAT#21	The minimum infrastructure must satisfy the principles of Security by Design (SbD).	Security through attribute-based and role-based access control Not scope of this part, security upcoming as Part 4

The following requirements are taken from the document "The Structure of the Administration Shell:

Trilateral perspectives from France, Italy and Germany" [19]. They are marked "tAAS".

> Note: the term "property" was used in a very broad sense in previous publications of Plattform Industrie 4.0. The metamodel in this document distinguishes between properties in a more classical sense such as data elements like "maximum temperature" and other submodel elements like operations, events, etc.

Source	Requirement	Tracking
tAAS-#1	The Administration Shell shall accept properties from different technical domains in mutually distinct submodels that can be version-controlled and maintained independently of each other.	*Identifiable* *AdministrativeInformation* *Submodel* Requirement tAAS-#1 implicitly contains the requirements of versioning, which is supported for all elements inheriting from *Identifiable*. Requirement tAAS-#1 is fulfilled because several submodels per Asset Administration Shell are possible. Every submodel is identifiable and an *Identifiable* may contain administrative information (*AdministrativeInformation*) for versioning. Submodels should be identifiable because they may be maintained independently of other submodels (requirement tAAS-#1) and can be reused within different Asset Administration Shells. However, since submodel elements may refer to elements from other Asset Administration Shells, dependencies have to be considered in parallel development and before reuse.
tAAS-#2	The Administration Shell should be capable of including properties from a wide range of technical domains and identify of which domain they were derived from.	*HasSemantics* Definitions from different dictionaries and different domains can be used within submodels via semantic reference properties. The only requirement is that the domain a property is derived from has a unique ID (*semanticId*).

Source	Requirement	Tracking
tAAS-#3	Different procedural models should be allowed to find definitions within each relevant technical domain. The models should meet the requirements of standards, consortium specifications, and manufacturer specifications sets.	*HasSemantics/semanticId* (see tAAS-#2) *ConceptDescription* Proprietary, manufacturer-specific property, or (more general) concept descriptions or copies from external dictionaries are supported by defining *ConceptDescriptions.* They are referenced in *semanticId* via their global *ID*. Predefined data specifications, e.g. for defining concept descriptions conformant to IEC 61360 are available in separate documents. Usage of proprietary concept descriptions is not recommended because interoperability cannot be ensured.
tAAS-#4	Different Administration Shells in respect of an asset must be capable of referencing each other. In particular, elements of an Administration Shell should be able to play the role of a "copy" of the corresponding components from another Administration Shell.	*AssetAdministrationShell/derivedFrom* The *derivedFrom* relationship is especially designed to support the relationship between an Asset Administration Shell representing a type asset and the Asset Administration Shells representing the instance assets of this type asset. See also tAAS-#16
tAAS-#5	Individual Administration Shells should, while retaining their structure, be combined into an overall Administration Shell.	*AssetAdministrationShell/assetInformation* *RelationshipElement* Relations between entities can be defined via the submodel element "*RelationshipElement*".

Source	Requirement	Tracking
tAAS-#6	Identification of assets, Administration Shells, properties, and relationships shall be achieved using a limited set of identifiers (IRDI, URI, GUID), providing they offer global uniqueness.	*Identifiable* Requirement tAAS-#6 is fulfilled for all elements inheriting from *Identifiable*. For example, this is the case for *Asset*, *AssetAdministrationShell,* and concept descriptions. Properties (like any other submodel element) are only referable. However, unique referencing is possible via the unique submodel ID and the *Reference* via *Key*s concept. The supported ID types include IRDI, URI (since URI are a subset of *IRI*), IRI and GUID (via *Custom*) as requested.
tAAS-#7	The Administration Shell should allow retrieval of alternative identifiers such as a GS1 and GTIN identifier in return to asset ID (dereferencing).	*AssetInformation/specificAssetIds* *AssetInformation/globalAssetId* Every asset has a globally unique identifier (*globalAssetId*). Besides this global identifier, additional external identifiers can be specified (*specificAssetIds*).
tAAS-#8	The Administration Shell consists of header and body.	*AssetAdministrationShell* *AssetAdministrationShell/id* *AssetAdministrationShell/administration* *AssetAdministrationShell/assetInformation* The Asset Administration Shell does not explicitly distinguish between header and body. However, the Asset Administration Shell has defined attributes like the global unique ID (*identification*), version information (*administration*), a mandatory reference to the asset identifier information (*assetInformation*), etc.

Source	Requirement	Tracking
tAAS-#9	The header contains information about the identification.	*AssetAdministrationShell/assetInformation* The Asset Administrative Shell represents an asset with a unique ID. See also tAAS-#7 See also tAAS-#13
tAAS-#10	The body contains information about the respective asset(s).	*AssetAdministrationShell/submodels* All submodels provide information related to the asset presented by the Asset Administration Shell. Note: an Asset Administration Shell represents exactly one asset. In case of a composite Asset Administration Shell, it implicitly represents several assets (see also tAAS-#5).
tAAS-#11	The information and functionality in the Administration Shell is accessible by means of a standardized application programming interface (API).	Covered in Part 2 of this document series [37]
tAAS-#12	The Administration Shell has a unique ID.	*AssetAdministrationShell/id* Since AssetAdministrationShell inherits from *Identifiable,* requirement tAAS-#12 is fulfilled.
tAAS-#13	The asset has a unique ID.	*AssetInformation/globalAssetId* The unique ID of the asset is the value of the globalAssetId. See also Requirement tAAS-#7.

Source	Requirement	Tracking
tAAS-#14	An industrial facility is also an asset, it has an Administration Shell and is accessible by means of ID.	*AssetInformation/globalAssetId* The only assumption is that the industrial facility also has a globally unique ID that can be used as value of the *globalAssetId*. Note: see also composite Asset Administration Shell (tAAS-#5) that allows the modelling of complex assets consisting of other assets that are represented by an Asset Administration Shell each.
tAAS-#15	Types and instances must be identified as such.	*AssetInformation/assetKind (*values: *Type or Instance)* *AssetAdministrationShell/derivedFrom* With attribute kind of Asset, Requirement tAAS-#15 is fulfilled, and type assets can be distinguished from instance assets. Additionally, a *derivedFrom* relationship can be established between the Asset Administration Shell for an instance asset and the Asset Administration Shell for the type asset.

Source	Requirement	Tracking
tAAS-#16	The Administration Shell can include references to other Administration Shells or Smart Manufacturing information.	*ReferenceElement* *File* *Blob* *AssetAdministrationShell/derivedFrom* The *derivedFrom* relationship between two Asset Administration Shells is special and is e.g. used to establish a relationship between instance assets and the type asset. For composite Asset Administration Shells (see tAAS-#5), there is also the relationship to the Asset Administration Shell, which the composite Asset Administration Shell is composed of. The *ReferenceElement* is very generic and can reference another Asset Administration Shell as well as information within another Asset Administration Shell or even some information that is completely outside any Asset Administration Shell (as long as it has a global unique ID). Files and blob can be used as submodel elements to include very generic manufacturing information that is not or cannot be modelled via properties or the other submodel elements defined for the Asset Administration Shell.

Source	Requirement	Tracking
tAAS-#17	Additional properties, e. g. manufacturer-specific, must be possible.	*HasDataSpecification* *ConceptDescription* *HasExtensions* Additional attributes for assets, properties, and other submodel elements, submodels, and even the Asset Administration Shell itself can be defined via data specification templates and checked by tools. New proprietary concept descriptions (*ConceptDescription*) can be added and used for semantic definition of properties or other submodel elements. Proprietary information can be added to any referable via extensions. The latter are not subject to standardization and can be ignored for interoperability use cases. Other submodel elements and submodels can be added via API (see tAAS-#11), assuming the corresponding access permissions are given.

Source	Requirement	Tracking
tAAS-#18	A reliable minimum number of properties must be defined for each Administration Shell.	*HasKind* for *Submodel* A reliable minimum number of properties is defined by the metamodel itself. They are called (class) attributes. *HasKind (with kind=Template) for Submodel* enables the definition of submodel (element) templates. These templates define the structure and the semantics of a submodel instance (via *semanticId*). Note: the term property within the metamodel has special semantics and shall not be confused with the implicitly available attributes of the different classes. Although these attributes might also be based on existing standards, they are no properties in the sense that a semantic reference can be added to define semantics externally. Semantics are defined for the metamodel itself in the class tables within this document.
tAAS-#19	The properties and other elements of information in the Administration Shell must be suitable for types and instances.	*HasKind* (with *kind=Template* or *kind=Instance*) for *Submodel* All elements inheriting from *HasKind* can distinguish between types (*kind=Template*) and instances (*kind=Instance*). This is especially true for *SubmodelElement* and *Submodel*. Note: submodels of *kind=Template* do not describe an asset of *kind=Type*.

Source	Requirement	Tracking
tAAS-#20	There must be a capability of hierarchical and countable structuring of the properties.	*SubmodelElementList* *SubmodelElementCollection* Requirement tAAS-#20 is fulfilled by lists and structs of data elements. Lists and structs are built recursively and contain other submodel elements of the same Asset Administration Shell. For referencing properties or other submodel elements of other Asset Administration Shells, a reference (*ReferenceElement*) or relationship element (*RelationshipElement*) needs to be included in the list or as element of the collection.
tAAS-#21	Properties shall be able to reference other properties, even in other Administration Shells.	*SubmodelElementList* *SubmodelElementCollection* *ReferenceElement* *RelationshipElement* *OperationVariable* in *Operation* A reference element can either reference any other element that is referable (i.e. inheriting from *Referable*) within the same or another Asset Administration Shell, or it can reference entities completely outside any Asset Administration Shell via its global ID. > Note: it is not always necessary to use a reference property for referencing elements within the same Asset Administration Shell. Depending on the context, submodel element collections, relations, etc. might be more suitable. Other elements are also referenced or used as input or output argument via *OperationVariable* within *operation*s.

Source	Requirement	Tracking
tAAS-#22	Properties must be able to reference information and functions of the Administration Shell.	*Operation* See also tAAS-#21 Functions in the sense of executable entities are represented as *operation*s.

[4] Only editorial changes

Appendix C: Backus-Naur-Form

The Backus-Naur form (BNF) – a meta-syntax notation for context-free grammars – is used to define grammars. For more information see Wikipedia[4].

A BNF specification is a set of derivation rules, written as

<symbol> ::= *expression*

where:

- <symbol> is a nonterminal (variable) and the *expression* consists of one or more sequences of either terminal or nonterminal symbols,
- ::= means that the symbol on the left must be replaced with the expression on the right,
- more sequences of symbols are separated by the vertical bar "|", indicating a choice, the whole being a possible substitution for the symbol on the left,
- symbols that never appear on a left side are terminals, while symbols that appear on a left side are non-terminals and are always enclosed between the pair of angle brackets <>,
- terminals are enclosed with quotation marks: "text". "" is an empty string,
- optional items are enclosed in square brackets: [<item-x>],
- items existing 0 or more times are enclosed in curly brackets are suffixed with an asterisk (**) such as <word> ::= <letter> \{<letter>}**,
- Items existing 1 or more times are suffixed with an addition (plus) symbol, , such as <word> ::= \{<letter>},
- round brackets are used to explicitly to define the order of expansion to indicate precedence, example: (<symbol1> | <symbol2>) <symbol3>,
- text without quotation marks is an informal explanation of what is expected; this text is cursive if grammar is non-recursive and vice versa.

Example:

> **<contact-address> ::= <name> "e-mail addresses:" <e-mail-Addresses>**
>
> **<e-mail-Addresses> ::= \{<e-mail-Address>}***
>
> **<e-mail-Addresse> ::= <local-part> "@" <domain>**
>
> **<name> ::= characters**

> **<local-part> ::= characters conformant to local-part in RFC 5322**
>
> **<domain> ::= characters conformant to domain in RFC 5322**

Valid contact addresses:

> **Hugo Me e-mail addresses: Hugo@example.com**
>
> **Hugo e-mail addresses: Hugo.Me@text.de**

Invalid contact addresses:

> **Hugo**
>
> **Hugo Hugo@ example.com**
>
> **Hugo@example.com**

[4] https://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form

Appendix D: Templates for UML Tables

D.1. General

The templates used for element specification are explained in this annex. For details for the semantics see Legend for UML Modelling.

For capitalization of titles, rules according to https://capitalizemytitle.com/ are used.

D.2. Template for Classes

Template for Classes:

Class:	<Class Name> [<<abstract>>] ["<<Experimental>>"] ["<<Deprecated>>"] ["<<Template>>"]
Explanation:	<Explanatory text>
Inherits from:	\{<Class Name> ";" }+ | "-"

Attribute	Explanation	Type	Card.
<attribute or association name> ["<<ordered>>"] ["<<Experimental>>"] ["<<Deprecated>>"]	<Explanatory text>	<Type>	<Card>

The following stereotypes can be used:

- <<abstract>>: Class cannot be instantiated but serves as superclass for inheriting classes
- <<Experimental>>: Class is experimental, i.e. usage is only recommended for experimental purposes because non backward compatible changes may occur in future versions
- <<Deprecated>>: Class is deprecated, i.e. it is recommended to not use the element any longer; it will be removed in a next major version of the model
- <<Template>>: Class is a template only, i.e. class is not instantiated but used for additional specification purposes (for details see parts 3 of document series)
- The following kinds of *Types* are distinguished:
- *Primitive:* Type is no object type (class) but a data type; it is just a value
- *Class:* Type is an object type (class); it is realized as composite aggregation (composition), and does not exist independent of its parent

- *Type:*
  - *<Class>:* Type is a class
  - *ModelReference<{Referable}>:* Type is a Reference with *Reference/type=ModelReference* and is called model reference; the {Referable} is to be substituted by any referable element (including *Referable* itself for the most generic case) – the element that is referred to is denoted in the *Key/type*=<{Referable}> for the last *Key* in the model reference; for the graphical representation see Legend for UML Modelling, Figure 82; for more information on referencing see Clause 5.3.9.
  - *<Primitive>: Type* is a primitive data type, see Clause 5.3.11
  - *<Enumeration>:* Type is an enumeration
- *Card.* is the cardinality (or multiplicity) defining the lower and upper bound of the number of instances of the member element. "**" denotes an arbitrary infinite number of elements of the corresponding Type. "0..1" means optional. "0..**" or "0..3" etc. means that the list may be either not available (null object) or empty.

Note 1: attributes having a default value are always considered to be optional; there is always a value for the attribute because the default value is used for initialization in this case.

Note 2: attributes or attribute elements with data type “string” or “langString” are considered to consist of at least one character.

Note 3: optional lists, i.e. attributes with cardinality > 1 and minimum 0, are considered to consist of at least one element.

Examples for valid model references

If class type equal to "ModelReference<Submodel>", the following reference would be a valid reference (using the text serialization as defined in Clause 7.2.3):

(Submodel)https://example.com/aas/1/1/1234859590

If class type equal to "ModelReference<Referable>", the following references would be valid references (using the text serialization as defined in Clause 7.2.3):

(Submodel)https://example.com/aas/1/1/1234859590

(Submodel)https://example.com/aas/1/1/1234859590, (Property)temperature

(Submodel)https://example.com/aas/1/1/1234859590, (File)myDocument

This would be an invalid reference for "ModelReference<Referable>", instead type "Reference" shall be used:

(Submodel)https://example.com/aas/1/1/1234859590, (File)myDocument (FragmentReference)Hints

This would be an invalid reference for "ModelReference<Submodel>"

(Submodel)https://example.com/aas/1/1/1234859590, (Property)temperature

D.3. Template for Enumerations

Template for Enumerations:

Enumeration:	<Enumeration Name> ["<<Experimental>>"] ["<<Deprecated>>"]
Explanation:	
Set of:	\{<Enumeration> ";" }+ | "-"

Literal	Explanation
<enumValue1>["<<Experimental>>"] ["<<Deprecated>>"]	<Explanatory text> Value of enumeration
<enumValue2> ["<<Experimental>>"] ["<<Deprecated>>"]	<Explanatory text> Value of enumeration, also included in one of the enumerations listed in "Set of:"

"Set Of" lists enumerations that are contained in the enumeration. It is only relevant for validation, making sure that all elements relevant for the enumeration are considered.

Enumeration values use Camel Case notation and start with a small letter. However, there might be exceptions in case of very well-known enumeration values.

D.4. Template for Primitives

Template for Primitive:

Primitive	Explanation	Value Examples
<Name of Primitive>	<Explanatory text>	Value examples

D.5. Handling of Constraints

Constraints are prefixed with **AASd-** followed by a three-digit number. The "d" in "AASd-" was motivated by "in Detail". The numbering of constraints is unique within namespace AASd; a number of a constraint that was removed will not be used again.

Note: in the Annex listing the metamodel changes, constraints with prefix AASs- or AASc- are also listed. These are security or data specification constraints, and are now part of the split document parts.

Appendix E: Legend for UML Modelling

E.1. OMG UML General

This annex explains the UML elements used in this specification. For more information, please refer to the comprehensive literature available for UML. The formal specification can be found in [35].

Figure 63 shows a class with name "Class1" and an attribute with name "attr" of type *Class2*. Attributes are owned by the class. Some of these attributes may represents the end of binary associations, see also Figure 70. In this case, the instance of *Class2* is navigable via the instance of the owning class *Class1*.[4]

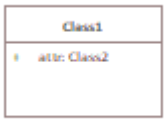


Figure 63. Class

Figure 64 shows that *Class4* inherits all member elements from *Class3*. Or in other word, *Class3* is a generalization of *Class4, Class4* is a specialization of *Class3*. This means that each instance of *Class4* is also an instance of *Class3*. An instance of *Class4* has the attributes *attr1* and *attr2,* whereas instances of *Class3* only have the attribute *attr1*.

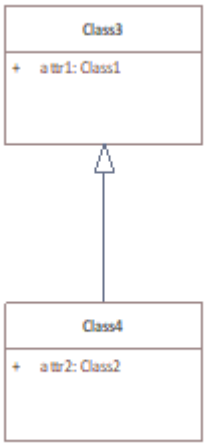


Figure 64. Inheritance/Generalization

Figure 65 defines the required and allowed multiplicity/cardinality within an association between instances of *Class1* and *Class2*. In this example, an instance of *Class2* is always related to exactly one instance of *Class1*. An instance of *Class1* is either related to none, one, or more (unlimited, i.e. no constraint on the upper bound) instances of *Class2*. The relationship can change over time.

Multiplicity constraints can also be added to attributes and aggregations.

The notation of multiplicity is as follows:

<lower-bound>.. <upper-bound>

where <lower-bound> is a value specification of type Integer - i.e. 0, 1, 2, … - and <upper-bound> is a value specification of type UnlimitedNatural. The star character (*) is used to denote an unlimited upper bound.

The default is 1 for lower-bound and upper-bound.

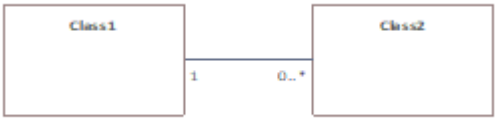


Figure 65. Multiplicity

A multiplicity element represents a collection of values. The default is a set, i.e. it is not ordered and the elements within the collection are unique and contain no duplicates. Figure 66 shows an ordered collection: the instances of *Class2* related to an instance of *Class1*. The stereotype <<ordered>> is used to denote that the relationship is ordered.

Figure 66. Ordered Multiplicity

Figure 67 shows that the member ends of an association can be named as well, i.e. an instance of *Class1* can be in relationship "relation" to an instance of *Class2*. Vice versa, the instance of *Class2* is in relationship "reverseRelation" to the instance of *Class1*.

Figure 67. Association

Figure 68 shows a composition, also called a composite aggregation. A composition is a binary association, grouping a set of instances. The individuals in the set are typed as specified by *Class2*. The multiplicity of instances of *Class2* to *Class1* is always 1 (i.e. upper-bound and lower-bound have value "1"). One instance of *Class2* belongs to exactly one instance of *Class1*. There is no instance of *Class2* without a relationship to an instance of *Class1*. Figure 69 shows the composition using an association relationship with a filled diamond as composition adornment.

Figure 68. Composition (Composite Aggregation)

Figure 69 shows an aggregation. An aggregation is a binary association. In contrast to a composition, an instance of *Class2* can be shared by several instances of *Class1*. Figure 69 shows the shared aggregation using an association relationship with a hallow diamond as aggregation adornment.

Figure 69. Aggregation

Figure 70 illustrates that the attribute notation can be used for an association end owned by a class. In this example, the attribute name is "attr" and the elements of this attribute are typed with *Class2.* The multiplicity, here "0..*", is added in square brackets. If the aggregation is ordered, it is added in curly brackets like in this example.

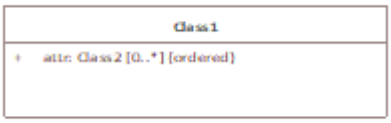


Figure 70. Navigable Attribute Notation for Associations

Figure 71 shows a class with three attributes with primitive types and default values. When a property with a default value is instantiated in the absence of a specific setting for the property, the default value is evaluated to provide the initial values of the property.

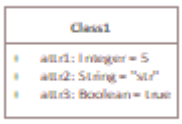


Figure 71. Default Value

Figure 72 shows that there is a dependency relationship between *Class1* and *Class2*. In this case, the dependency means that *Class1* depends on *Class2* because the type of attribute *attr* depends on the specification of class *Class2.* A dependency is depicted as dashed arrow between two model elements.

Figure 72. Dependency

Figure 73 shows an abstract class. It uses the stereotype <<abstract>>. There are no instances of abstract classes. They are typically used for specific member elements that are inherited by non-abstract classes.

Figure 73. Abstract Class

Figure 74 shows a package named "Package2". A package is a namespace for its members. In this example, the member belonging to *Package2* is class *Class2*.

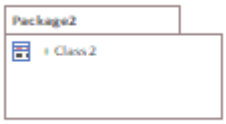


Figure 74. Package

Figure 75 shows that all elements in *Package2* are imported into the namespace defined by *Package1*. This is a special dependency relationship between the two packages with stereotype <<import>>.

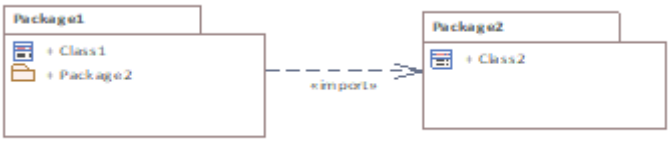


Figure 75. Imported Package

Figure 76 shows an enumeration with the name "Enumeration1". An enumeration is a data type with its values enumerated as literals. It contains two literal values, "a" and "b". It is a class with stereotype <<enumeration>>. The literals owned by the enumeration are ordered.

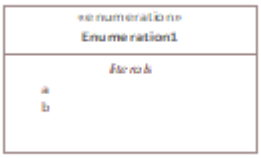


Figure 76. Enumeration[5]

Figure 77 shows the definition of the data type with the name "DataType1". A data type has instances that

are identified only by their value. It is a class with stereotype <<dataType>>.

Figure 77. Data Type

Figure 78 shows a primitive data type with the name "int". Primitive data types are predefined data types, without any substructure. The primitive data types are defined outside UML.

Figure 78. Primitive Data Type

Figure 79 shows how a note can be attached to an element, in this example to class "Class1".

Figure 79. Note

Figure 80 shows how a constraint is attached to an element, in this example to class "Class1".

Figure 80. Constraint

E.2. UML Naming Rules

The following rules are used for naming of classes, attributes etc.:

- all names use CamelCase; for exceptions see rules for Enumeration values,
- class names always start with a capital letter,
- attribute names always start with a small letter,
- primitive types start with a capital letter; exception: predefined types of XSD like string,
- enumerations start with a capital letter,

- names of member ends of an association start with a capital letter,
- all stereotypes specific to the Asset Administration Shell specification start with a capital letter, e.g. "<<Deprecated>>"; predefined stereotypes in UML start with a small letter, e.g. "<<abstract>>" or "<<enumeration>>".

In UML, the convention is to name associations and aggregations in singular form. The multiplicity is to be taken into account to decide on whether there are none, a single, or several elements in the corresponding association or aggregation.

Note: a plural form of the name of attributes with cardinality >=1 may be needed in some serializations (e.g. in JSON). In this case, it is recommended to add an "s". In case of resulting incorrect English (e.g. isCaseOf isCaseOfs), it must be decided whether or not to support such exceptions.

E.3. Templates, Inheritance, Qualifiers, and Categories

At first glance, there seems to be some overlapping within the concepts of data specification templates, extensions, inheritance, qualifiers, and categories introduced in the metamodel. This clause explains the commonalities and differences and gives hints for good practices.

In general, an extension of the metamodel by inheritance is foreseen. Templates might also be used as alternatives.

- Extensions can be used to add proprietary and/or temporary information to an element. Extensions do not support interoperability. They can be used as work-around for missing properties in the standard. In this case, the same extensions are attached to all elements of a specific class (e.g. to properties). However, in general, extensions can be attached in a quite arbitrary way. Properties are defined in a predefined way as key values pairs (in this case keys named "name").
- In contrast to extensions, templates aim at enabling interoperability between the partners that agree on the template. A template defines a set of attributes, each of them with clear semantics. This set of attributes corresponds to a (sub-)schema. Templates should only be used if different instances of the class follow different schemas and the templates for the schemas are not known at design time of the metamodel. Templates might also be used if the overall metamodel is not yet stable enough or a tool supports templates but not (yet) the complete metamodel. Typically, all instances of a specific class with the same category provide the same attribute values conformant to the template. In contrast to extensions, the attributes in the template have speaking names.

Note: categories are deprecated and should no longer be used.

- However, when using non-standardized proprietary data specification templates, interoperability cannot be ensured and thus should be avoided whenever possible.
- In case all instances of a class follow the same schema, inheritance and/or categories should be used.
- Categories can be used if all instances of a class follow the same schema but have different constraints depending on their category. Such a constraint might specify that an optional attribute is mandatory for this category (like the unit that is mandatory for properties representing physical values). Realizing the same via inheritance would lead to multiple inheritance – a state that is to be avoided[6].

Note: categories are deprecated and should no longer be used.

- Qualifiers are used if the structure and the semantics of the element is the same independent of its qualifiers. Only the quality or the meaning of the value for the element differs.
- Value qualifiers are used if only the quantity but not the semantics of the value changes. Depending on the application, either both value and qualifier define the "real" semantics together, or the qualifier is not really relevant and is ignored by the application. Example: the actual temperature might be good enough for non-critical visualization of trends, independent of whether the temperature is measured or just estimated (qualifier would denote: measured or estimated).
- Concept qualifiers are used to avoid multiplying existing semantically clearly defined concepts with the corresponding qualifier information, e.g. life cycle.
- Template qualifiers are used to guide the creation and validation of element instances.

E.4. Notes to Graphical UML Representation

Specific graphical modelling rules, which are used in this specification but not included in this form, are explained below [35].

Figure 81 shows different graphical representations of a composition (composite aggregation). In Variant A, a relationship with a filled aggregation diamond is used. In Variant B, an attribute with the same semantics is defined. And in Variant C, the implicitly assumed default name of the attribute in Variant A is explicitly stated. This document uses notation B.

It is assumed that only the end member of the association is navigable per default, i.e. it is possible to navigate from an instance of *Class1* to the owned instance of *Class2* but not vice versa. If there is no name for the end member of the association given, it is assumed that the name is identical to the class name but starting with a small letter – compared to Variant C.

Class2 instance only exists if the parent object of type *Class1* exists.

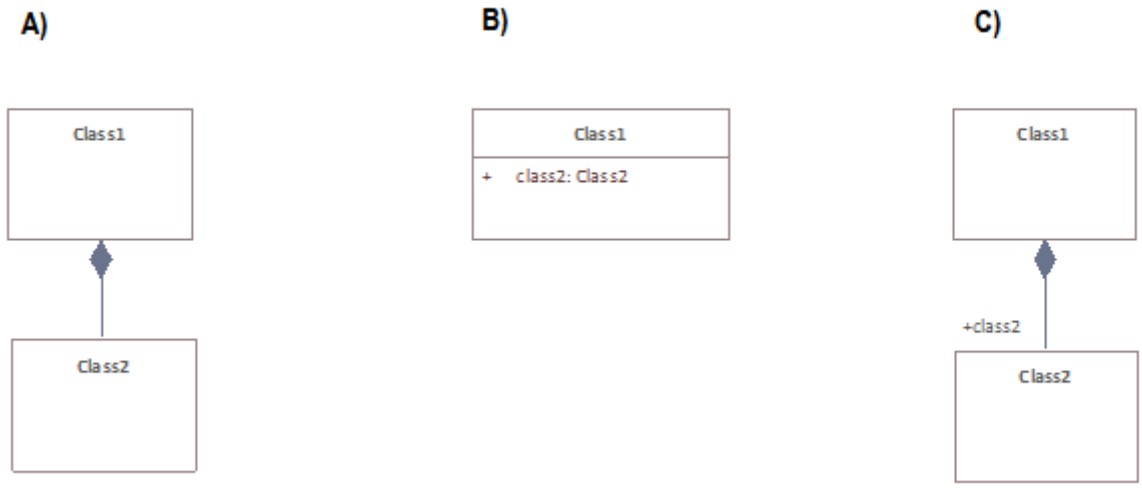


Figure 81. Graphical Representations of Composite Aggregation/Composition

Figure 82 shows different representations of a shared aggregation: a *Class2* instance can exist independently of a *Class1* instance; it only references the instances of *Class2*. Now an attribute with the same semantics is defined In Variant B. The reference is denoted by a star added after the type of the attribute.

It is assumed that only the end member of the aggregation association is navigable per default, i.e. it is possible to navigate from an instance of *Class1* to the owned instance of *Class2* but not vice versa. Otherwise, variant B would not be identical to Variant A.

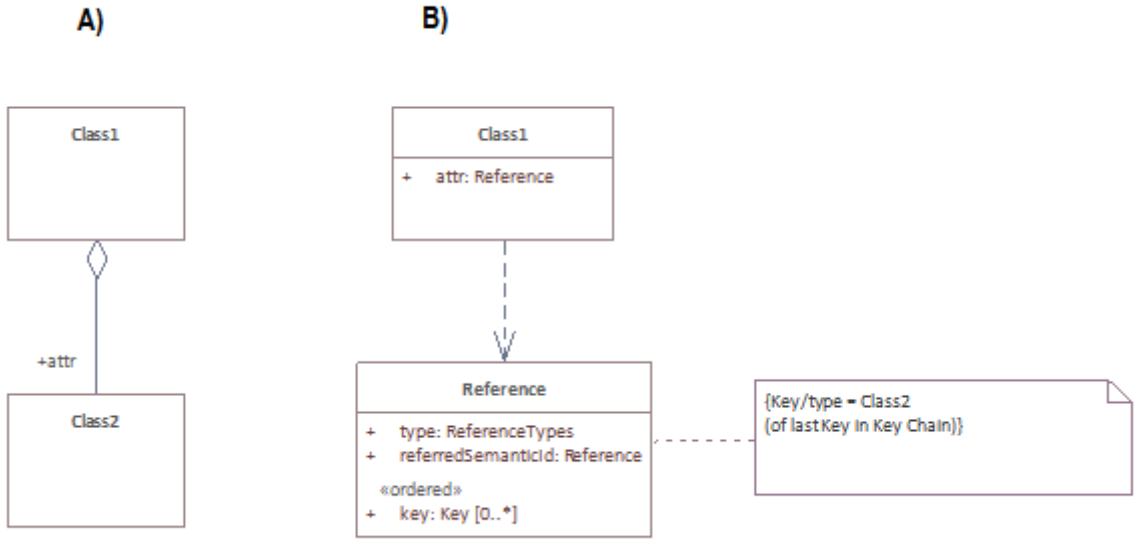


Figure 82. Graphical Representation of Shared Aggregation

A specialty in Figure 82 is that the aggregated instances are referables in the sense of the Asset Administration Shell metamodel (i.e. they inherit from the predefined abstract class "Referable"). This is why Variant B is identical to Variant A. This would not be the case for non-referable elements in the metamodel. The structure of a reference to a model element of the Asset Administration Shell is explicitly defined. A

model reference consists of an ordered list of keys. The last key in the key chain shall reference an instance of type *Class2* (i.e. Reference.

Figure 83 show different graphical representations of generalization. Variant A is the classical graphical representation as defined in [35]. Variant B is a short form, if *Class1* is not on the same diagram. The name of the class that *Class3* is inheriting from is depicted in the upper right corner.

Variant C not only shows which class Class3 instances are inheriting from, but also what they are inheriting. This is depicted by the class name it is inheriting from, followed by "::" and then the list of all inherited elements – here attribute *class2*. Typically, the inherited elements are not shown.

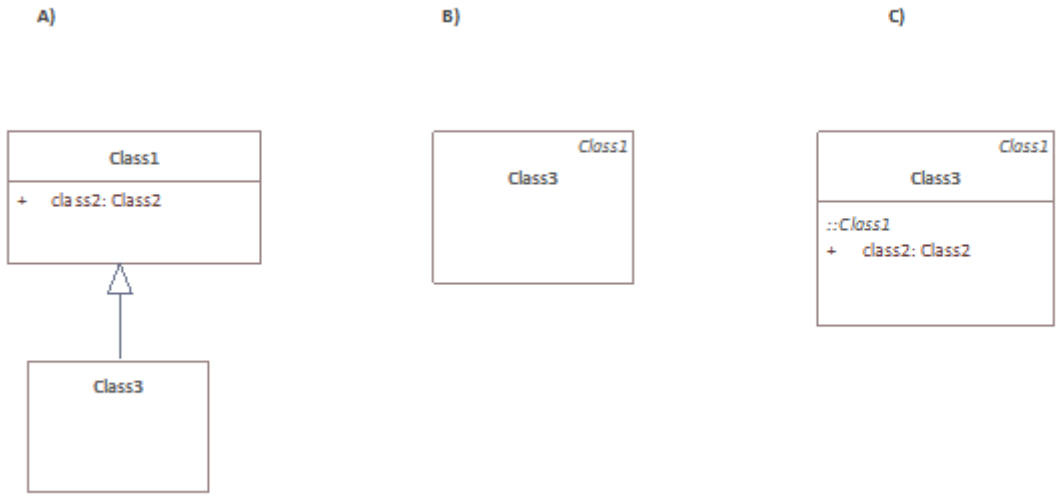


Figure 83. Graphical Representation of Generalization/Inheritance

Figure 84 depicts different graphical notations for enumerations in combination with inheritance. On the left side "Enumeration1" additionally contains the literals as defined by "Enumeration2".

> Note 1: the direction of inheritance is opposite to the one for class inheritance. This can be seen at the right side of Figure 84 that defines the same enumeration but without inheritance.

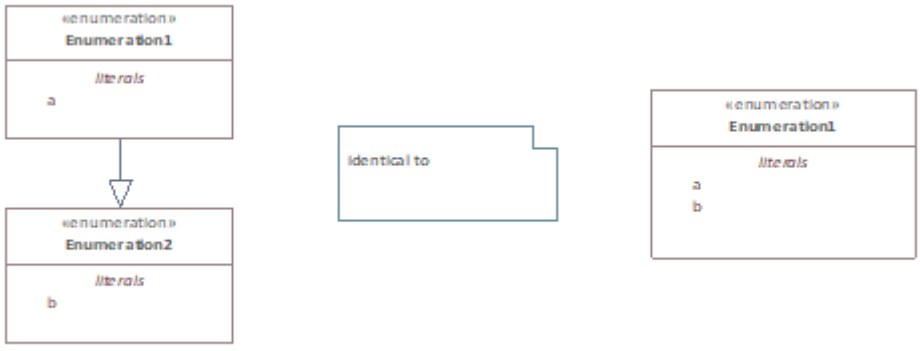


Figure 84. Graphical Representation for Enumeration with Inheritance

Note 2: in this specification all elements of an enumeration are ordered alphabetically.

Figure 85 shows an experimental class, marked by the stereotype "Experimental”.

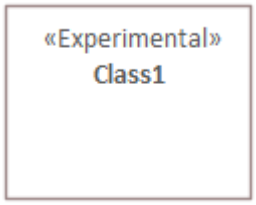


Figure 85. Graphical Representation for Experimental Classes

Figure 86 depicts a deprecated class, which is marked by the stereotype "Deprecated".

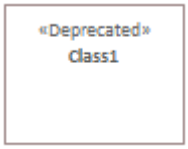


Figure 86. Graphical Representation for Deprecated Elements

Figure 87 shows a class representing a template. It is marked by the stereotype "Template".

Figure 87. Graphical Representation of a Template Class

[4] „Navigability notation was often used in the past according to an informal convention, whereby non-navigable ends were assumed to be owned by the Association whereas navigable ends were assumed to be owned by the Classifier at the opposite end. This convention is now deprecated. Aggregation type, navigability, and end ownership are separate concepts, each with their own explicit notation. Association ends owned by classes are always navigable, while those owned by associations may be navigable or not. [35]”

[5] In Enterprise Architect, the single enumeration values also have a stereotype <<enum>> each.

[6] Exception: multiple inheritance is used in this specification, but only in case of inheriting from abstract classes.

Appendix F: How to Use the Metamodel

F.1. Composite I4.0 Components

As described in Clause 4.2.1, there is a class of relationships between assets of different hierarchy levels. In this class of relationships, automation equipment is explained as a complex, interrelated graph of automation devices and products, performing intelligent production and self-learning/optimization tasks.

Details and examples for composite I4.0 Components can be found in [13].

The following modelling elements in the Asset Administration Shell metamodel can be used to realize such composite I4.0 Components:

- *RelationshipElement,* used to describe relationships between assets and other elements,
- *Submodel,* where a complex asset is composed out of other entities and assets, which are specified in a bill of material together with their relationship to each other.

> Note: the submodel template defining the structure of such a bill of material is not predefined by the Asset Administration Shell metamodel. However, *Entity* elements were designed for the purpose of building bills of material.

- Not every entity (*Entity*) that is part of the bill of material of an asset necessarily has its own Asset Administration Shell. As described in [13], self-managed entities are distinguished from co-managed entities (*Entity/entityType*).
    1. Self-Managed Entities have their own Asset Administration Shell. This is why a reference to this asset is specified via *Entity/globalAssetId* or an *Entity/specificAssetId*. Additionally, further property statements (*Entity/statement*) [16] can be added to the asset that are not specified in the Asset Administration Shell of the asset itself, since they are specified in relation to the composite I4.0 Component only.
    2. There is no separate Asset Administration Shell for co-managed entities. The relationships and property statements of such entities are managed within the Asset Administration Shell of the composite I4.0 Component.

Figure 88 shows an extract of the metamodel containing the most important elements to describe composite I4.0 Components.

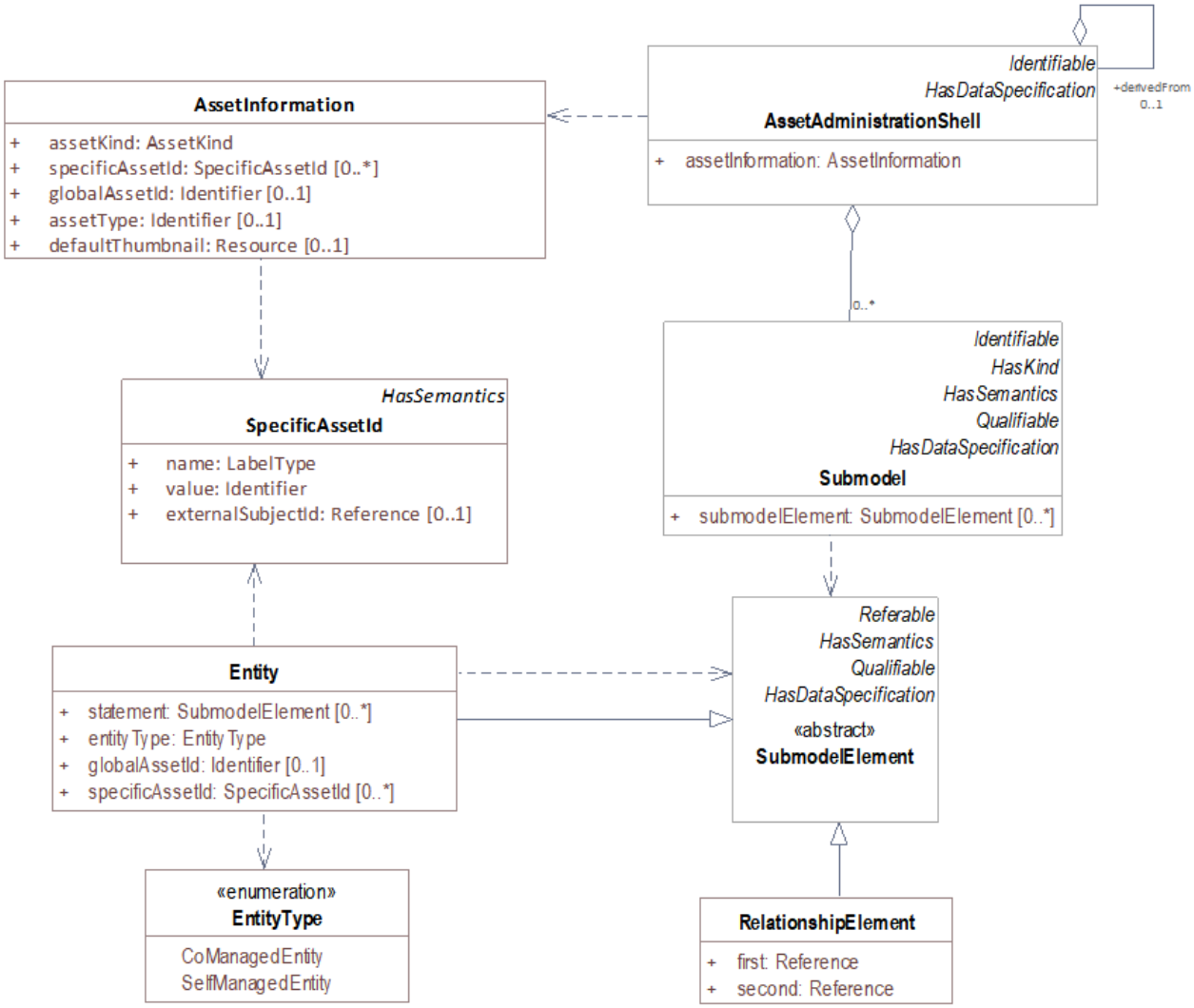


Figure 88. Extract From Metamodel for Composite I4.0 Components

Appendix G: Metamodel UML with Inherited Attributes

In this annex, some UML diagrams are shown together with all attributes inherited for a better overview.

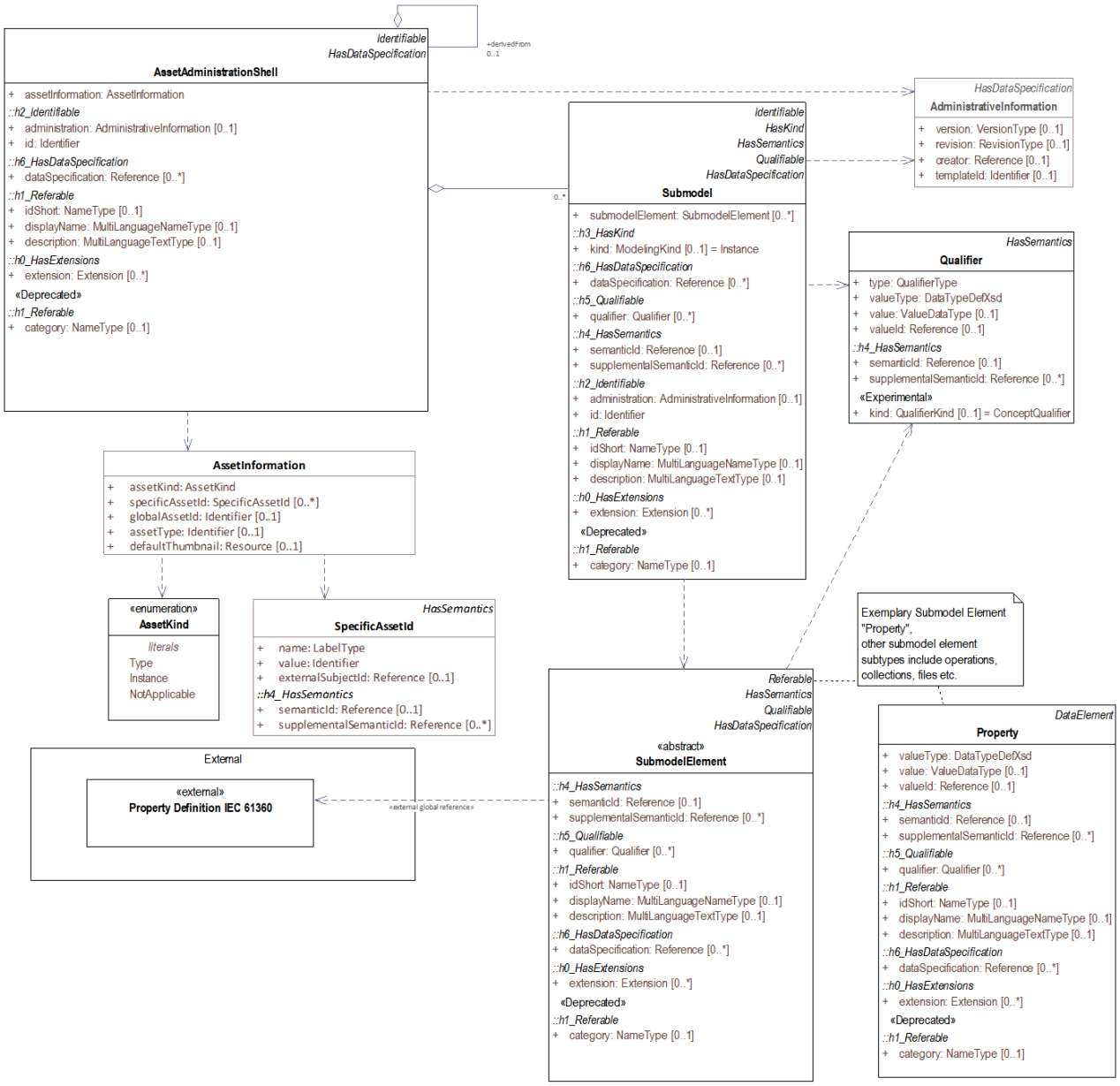


Figure 89. Selected Classes of Metamodel with Inherited Attributes

Note: abstract classes are numbered h0_, h1_ etc. Their aliases however are defined without this prefix. The reason for this naming is that no order for inherited classes can be defined in the tooling used for UML modelling (Enterprise Architect); they are ordered alphabetically.

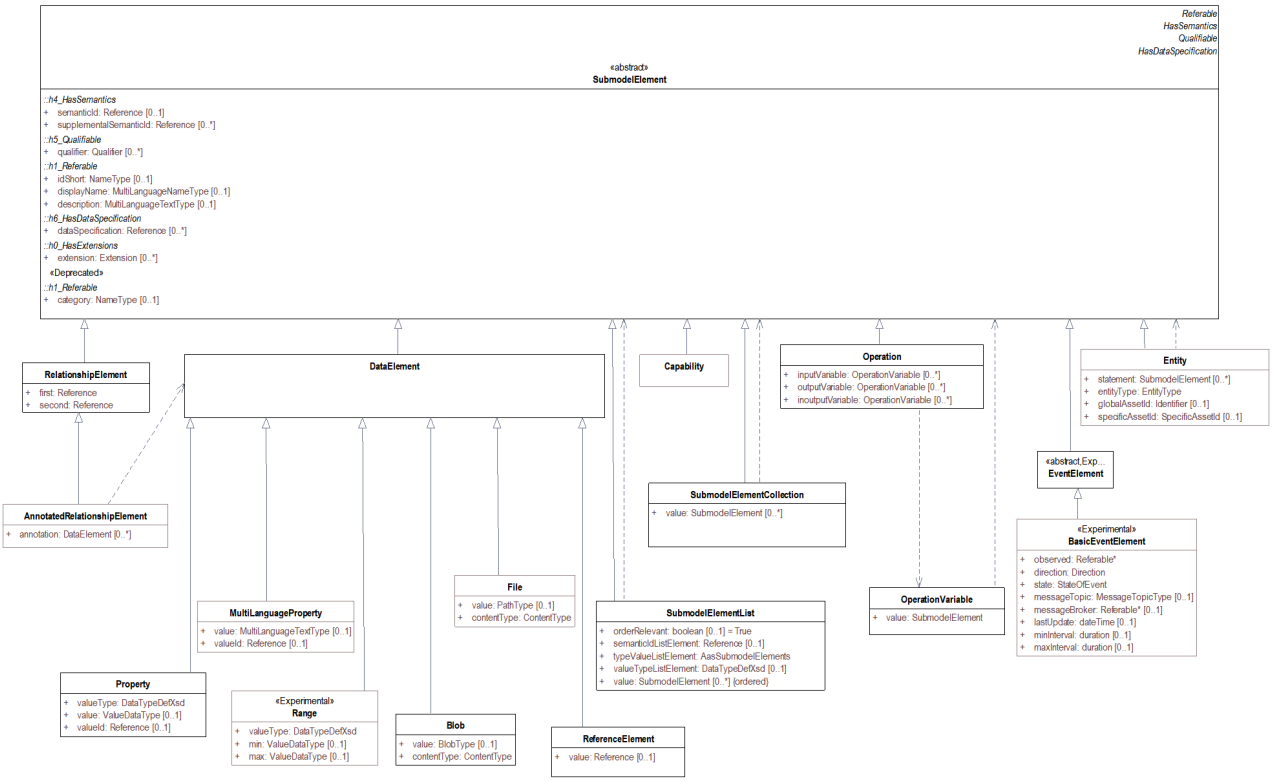


Figure 90. Model for Submodel Elements with Inherited Attributes

Appendix H: Metamodel Changes

H.1. General

This annex lists the changes from version to version of the metamodel, together with major changes in the overall document. Non-backward compatible changes (nc) are marked as such.

nc="x" means non-backward compatible; if no value is added in the table, then the change is backward compatible.

nc="(x)" means that the change made was implicitly contained or stated in the document before and is now being formalized. Therefore, the change is considered to be backward compatible.

Before splitting the document, changes to the data specification templates and the security of the metamodel were listed in separate tables.

Three tables are introduced to explain the changes:

1. changes w.r.t. previous version,
2. new elements in metamodel w.r.t previous version,
3. new, changed, or removed constraints w.r.t previous version.

If there are no changes the corresponding tables are obmitted.

> Note: before V3.0, the security metamodel and the predefined data specifications were also part of this document. Therefore, security metamodel changes were listed using the three subclauses as described above.

H.2. Changes V3.0 vs. V2.0.1

Major Changes:

- Document split into several documents: Part 1 on metamodel of the AAS (this document), Part 5 on the aasx package exchange format, Part 3 series on the predefined data specifications, and Part 4 (security)
- CHANGED: Split of SubmodelElementCollection into SubmodelElementList (with orderRelevant) and SubmodelElementCollection
- CHANGED: Reference type and referredSemanticId added to Reference; Local and Parent attributes removed from Reference. Logical enumeration concept updated. Some renaming and some new enumerations. Constraint added for references. Grammar for text serialization updated.

- CHANGED: idType from Identifier removed, ID now string
- CHANGED: idShort of Referable now optional + Constraints added with respect to ID and idShort, includes that idShort of Submodels etc. no longer need to be unique in the context of an AssetAdministrationShell
- CHANGED: semanticId no longer mandatory but recommended for SubmodelElement. semanticId now recommended not only for submodel instance but also for submodel template
- NEW: supplemental Semantic IDs
- CHANGED: SubmodelElements do no longer inherit from HasKind, only Submodel has distinction between submodel template or submodel instance (including update of tAAS-#18 and -#19)
- CHANGED: Revised concept on handling of Asset and assetIdentificationModel (assetInformation), Asset removed, no Asset/billOfMaterial any longer. Specific asset IDs added.
- CHANGED: Attributes with type "string" were substituted by string types with length restrictions + some more constraints on string handling. LangSringSet handling updated.
- REMOVED: ConceptDictionaries removed, no longer supported
- REMOVED: Views removed, no longer supported
- NEW: Events now experimental parts of normative part (including renaming and smaller changes)
- NEW: Besides type assets and instance assets, assets for which this kind of classification is not applicable are also supported (conformant to IEC 63278-1)
- REMOVED: Security attribute removed from Asset Administration Shell, Access Control remains part of the specification
- ENHANCED: DataTypeIEC61360 extended with values for IRI, IRDI, BLOB, FILE + corresponding new constraints added
- CHANGED: Handling of attributes with xsd-related types like DataTypeDef. New types introduced.
- NEW: hasExtensions introduced
- CHANGED: In some mappings or serializations, the type "Reference" is converted into a single string. In this case, it is now required (and not only recommended) to use the defined string serialization.
- CHANGED: Extracted and not part of this specification any longer; mapping rules for different serializations + Schemata + Example in different serializations
- CHANGED: Referable/category set to <<Deprecated>>
- NEW: Besides <<Deprecated>>, new stereotype <<Experimental>> introduced
- NEW: Qualifier/kind introduced (ValueQualifier, TemplateQualifier, ConceptQualifier)
- CHANGED: Terms and Definitions updated to be conformant to IEC 63278-1 DRAFT, July 2022 (note: this document contains more terms and definitions than IEC 63278-1 and vice versa: not all terms and definitions from IEC 63278-1 are included). Definition of view removed. Definition of service added from

Part 2.

- CHANGED: Description of ModellingKind
- Parent attribute in Referables removed
- NEW: Two new terms introduced: coded value and explicit value
- CHANGED: Updated grammar on how to define semantic identifiers for metamodel elements of this specification
- CHANGED: Blob type changed from byte[0..*] to base64Binary
- NEW: Appendix for Backus-Naur form (BNF) + update of all grammars using BNF and variants so that they are using consistent grammar language
- NEW: Clause on embedded data specifications
- EDITORIAL: Text updated, no kind column any longer in class tables, instead notation of ModelReference<{Referable}>. New table for Primitives/Data Types
- EDITORIAL: New clause "Introduction"
- EDITORIAL: New clause "Matching strategies for semantic identifiers"
- EDITORIAL (REMOVED): Clause on Tooling and Open Source removed
- EDITORIAL: Vector graphics
- NEW: Constraints implicitly contained in text were formalized and numbered (normative)
- NEW: Environment explicitly part of UML (was part of serializations from the beginning)

Bugfixes:

- Corrected Japanese example for xd:string
- HasDataSpecification/embeddedDataSpecs 0..* not 0..1
- Qualifer is and never was abstract (Constraint was), table was correct, UML corrected
- Correct AASd-051: VIEW no longer supported
- Type of SubmodelElementList/typeValueListElement corrected to AasSubmodelElements (before SubmodelElementElements
- Missing table for enumeration ReferenceTypes added
- Bugfix table specifications w.r.t. kind of attribute (from aggr to attr – column kind was removed, see above)

Smaller changes:

- EDITORIAL: Qualifier description updated

- EDITORIAL: Reformulation of constraints dealing with References and Key/type
- EDITORIAL: Examples now with https: and not http:
- EDITORIAL: Footnotes reused
- EDITORIAL: Added explanation for annotated relationship elements
- EDITORIAL: Asset type and asset instance now type asset and instance asset (conformant to IEC 63278-1)
- EDITORIAL: example for langString serialization changed (table 6)

H.3. Metamodel Changes V3.0 VS. V2.0.1

Table 13. Changes

Nc	V3.0 Change w.r.t. V2.0.1	Comment
	anySimpleTypeDef	Type removed, was no longer used in any class definition, was mentioned in text only.
X	Asset	Removed, asset referenced via AssetInformation/globalAssetId only
X	AssetAdministrationShell/asset	Removed, substituted by AssetAdministrationShell/assetInformation (but no reference any longer, instead now aggregation)
X	AssetAdministrationShell/conceptDictionaries	Removed
X	AssetAdministrationShell/security	Removed Note: Security is still part of the Asset Administration Shell, but the Asset Administration Shell and its elements are referenced from Security.
x	AssetAdministrationShell/view	Removed, Views no longer supported
	AssetKind/Instance	Updated description of value "Instance" of enumeration "AssetKind" conformant to IEC 63278-1
	AssetKind/Type	Updated description of value "Type" of enumeration "AssetKind" conformant to IEC 63278-1
X	BasicEvent	Renamed to BasicEventElement and set to <<Experimental>>

Nc	V3.0 Change w.r.t. V2.0.1	Comment
(x)[2]	xref:BlobType[BlobType	Primitive changed from "group of bytes" to Base64Binary
X	ConceptDictionary	Removed
X	Constraint	Abstract class removed. Formula now used in Security part only
x	DataSpecification	No longer inherits from Identifiable. However, same attribute names and types
x	DataSpecification/description	Type changed from LangStringSet to MultiLanguageTextType, thus adding a length constraint
	DataSpecificationContent	Stereotype <<Template>> added
(x)	DataTypeDef	Removed and split into DataTypeDefXsd and DataTypeDefRdf. Some types excluded and not supported because now XML Schema 1.0 reference Before: just string allowing any xsd simple type as string + added prefix xs: or rdf:, resp., to every value in enumeration
X	Entity/asset	Removed, substituted by Entity/globalAssetId and Entity/specificAssetId
x	Event	Renamed to EventElement
x	File/mimeType	Renamed to contentType + Type changed from MimeType to ContentType
x	Formula	Now abstract class Formula now used in Security part only
x	HasKind/kind	Type changed from ModelingKind to ModellingKind
x	Identifiable/identification	Removed Substituted by Identifiable/id
x	IdentifiableElements	Renamed to AasIdentifiables

Nc	V3.0 Change w.r.t. V2.0.1	Comment
x	Identifier	Type changed Before struct class with two attributes: id and idType. Now string data type only. Maximum length defined: 2,000 characters
	IdentifierType	Enumeration removed because no idType any longer
x	Key/idType	removed
x	Key/local	Local attribute removed.
x	Key/value	Type changed from string to Identifier, thus adding a length constraint
(x)	KeyElements	Renamed to KeyTypes The elements remain, except for new SubmodelElementList, and renamed submodel elements Event and BasicEvent to EventElement and BasicEventElement
	KeyType	Enumeration removed because no Key/idType any longer
	LocalKeyType	Enumeration removed because no Key/idType any longer
x	MimeType	Type name changed to ContentType
x	MultiLanguageProperty/value	Type changed from LangStringSet to MultiLanguageTextType, thus adding a length constraint
x	PathType	Same as Identifier, i.e. length constraint added
	Property/valueType	Type changed from DataTypeDef to DataTypeDefXsd
x	Qualifiable/qualifier	Type changed from Constraint to Qualifier
	Qualifier	No longer inherits from abstract class "Constraint"
	Qualifier/valueType	Type changed from DataTypeDef to DataTypeDefXsd
x	QualifierType	Type changed from string to NameType (i.e. length constraint added)

Nc	V3.0 Change w.r.t. V2.0.1	Comment
	Range/valueType	Type changed from DataTypeDef to DataTypeDefXsd
x	Referable/category	Type changed from string to NameType (i.e. length constraint added) Set to deprecated
x	Referable/description	Type changed from string to MultiLanguageTextType, thus adding length constraint
	Referable/idShort	Now optional, was mandatory Type changed from string to NameType (i.e. length constraint added)
x	Referable/parent	Parent attribute removed.
x	ReferableElement/BasicEvent	Renamed to BasicEventElement Now part of AasSubmodelElements
x	ReferableElements	Substituted with enumeration AasSubmodelElements and AasIdentifiables
x	ReferableElements/AccessPermissionRule	Removed from Enumeration, AccessPermissionRule is not referable any longer Not part of new AasReferableNonIdentifiables
x	ReferableElements/Event	Renamed to EventElement Now part of AasSubmodelElements
x	ReferenceTypes/GlobalReference	Renamed to ExternalReference
	RelationshipElement/first	Type changes from model reference Referable to Reference (global or model reference)
	RelationshipElement/second	Type changes from model reference Referable to Reference (global or model reference)
x[1]	SubmodelElement/kind	Removed. SubmodelElement no longer inheritsfrom HasKind
	ValueDataType	Before as specified via DataTypeDef, now any xsd atomic type as specified via DataTypeDefXsd
x	View	Removed

Table 14. New Elements in Metamodel

	New Elements V3.0 vs V2.0.1	Comment
	AasIdentifiables	New enumeration used for References, includes abstract Identifiable Before: Identifiables
	AasReferableNonIdentifiables	New enumeration used for References
	AasReferables	New enumeration used for References, includes abstract Referable
	AasSubmodelElements	New enumeration used for References Before: ReferableElements
	AdministrativeInformation/creator	New optional attribute
	AdministrativeInformation/templateId	New optional attribute
x	AssetAdministrationShell/assetInformation	Substitute for AssetAdministrationShell/asset; no reference any longer, instead aggregation
	AssetInformation	with attributes/functionality from former class Asset because not specific to Asset but AAS
	AssetInformation/assetKind	Former Asset/assetKind
	AssetInformation/assetType	New optional attribute
	AssetInformation/globalAssetId	Former Asset/identification/id
	AssetInformation/specificAssetId	Former Asset/assetIdentificationModel
	AssetInformation/thumbnail	Optional Attribute of new class AssetInformation that was not available in Asset class before
	AssetKind/NotApplicable	New enumeration value
	BasicEventElement	Former BasicEvent New <<Experimental>> submodel element for events
	BasicEventElement/direction	Was part of non-normative part before
	BasicEventElement/lastUpdate	Was part of non-normative part before

	New Elements V3.0 vs V2.0.1	Comment
	BasicEventElement/maxInterval	Was part of non-normative part before Type changed from dateTime to duration
	BasicEventElement/messageBroker	Was part of non-normative part before
	BasicEventElement/messageTopic	Was part of non-normative part before
	BasicEventElement/minInterval	Was part of non-normative part before Type changed from dateTime to duration
	BasicEventElement/observed	Former name: BasicEvent/observed
	BasicEventElement/state	Was part of non-normative part before
	ContentType	Former name: MimeType Maximum length defined: 100 characters
	DataTypeDefRdf	Enumeration for types of Rdf + added prefix rdf: to every value in enumeration
	DataTypeDefXsd	Enumeration that corresponds to anySimpleTypes of XML Schema 1.0 + added prefix xs: to every value in enumeration
	dateTimeStamp	New data type for metamodel as used in EventPayload
	DataSpecification/administration	Was inherited before by Identifiable
	DataSpecification/id	Was inherited before by Identifiable
	DataSpecification/desciption	Was inherited before by Identifiable
	Direction	New Enumeration for BasicEventElement
	Entity/globalAssetId	Former Entity/asset was split into globalAssetId and specificAssetId
	Entity/specificAssetId	Former Entity/asset was split into globalAssetId and specificAssetId
	Environment	New class for entry point for Asset Administration Shells, submodels and concept descriptions.
	EventElement	Former name: Event Set to <<Experimental>>

	New Elements V3.0 vs V2.0.1	Comment
	EventPayload	New experimental class for event payload Was part of non-normative part before
	EventPayload/observableSemanticId	Was part of non-normative part before
	EventPayload/payload	Was part of non-normative part before Type changed from string to BlobType
	EventPayload/source	Was part of non-normative part before Type changed from ModelReference(Referable) to ModelReference(EventElement)
	EventPayload/sourceSemanticId	Was part of non-normative part before
	EventPayload/subjectId	Was part of non-normative part before
	EventPayload/timestamp	Was part of non-normative part before Type changed from dateTimeStamp to dateTime because restriction to types of XML Schema 1.0 that does not contain dateTimeStamp. dateTimeStamp is a derived type of dateTime in XML Schema 1.1
	EventPayload/topic	Was part of non-normative part before Type changed from string to MessageTopicType (i.e. length constraint added, maximum 255 characters)
	Extension	New class, part of new abstract class HasExtensions
	Extension/name	
	Extension/refersTo	
	Extension/valueType	
	Extension/valueType	
	File/contentType	Fromer File/mimeType
	FragmentKeys	New enumeration used for References
	GenericFragmentKeys	New enumeration used for References
	GenericGloballyIdentifiers	New enumeration used for References
	GloballyIdentifiables	New enumeration used for References
	HasExtensions	New abstract class, inherited by Referable
	HasSemantics/supplementalSemanticId	New attribute

	New Elements V3.0 vs V2.0.1	Comment
	Identifiable/id	Substitute for Identifiable/identification
	KeyTypes	Before: KeyElements New submodel element SubmodelElementList added, renamed submodel elements Event and BasicEvent to EventElement and BasicEventElement
	LabelType	New string type with maximum 64 characters
	MessageTopicType	New string type with maximum 255 characters
	ModellingKind	Renamed enumeration, before: ModelingKind
	MultiLanguageNameType	Substitute for LangStringSet with short multi-language strings, maximum 64 characters
	MultiLanguageTextType	Substitute for LangStringSet with long multi-language strings, maximum 64 characters
	NameType	New string type with maximum 128 characters
	Qualifier/kind	New experimental attribute for Qualifier
	QualifierKind	New enumeration for Qualifier/kind
	Referable/displayName	New optional attribute for all referables
	Reference/referredSemanticId	New optional attribute for Reference
	Reference/type	New mandatory attribute for Reference
	ReferenceTypes	New enumeration for Reference/type
	ReferenceTypes/ExternalReference	Enumeration value, was named GlobalReference before
	Resource	new type for AssetInformation/defaultThumbnail
	ShortNameType	New string type with maximum 64 characters
	SpecificAssetId	New type for AssetInformation/specificAssetId
	SpecificAssetId/externalSubjectId	See Attribute Based Access Control (ABAC) for subject concept
	SpecificAssetId/name	New type for AssetInformation/specificAssetId
	SpecificAssetId/value	New type for AssetInformation/specificAssetId
	StateOfEvent	New experimental enumeration for BasicEventElement

	New Elements V3.0 vs V2.0.1	Comment
	SubmodelElementElements	Enumeration for submodel elements (split of ReferableElements)
	SubmodelElementList	Before SubmodelElementCollection was used for lists and collections
	SubmodelElementList/orderRelevant	Similar to SubmodelElementCollection/ordered
	SubmodelElementList/semanticIdListElement	Attribute of new class SubmodelElementList
	SubmodelElementList/typeValueListElement	Attribute of new class SubmodelElementList
	SubmodelElementList/value	Similar to SubmodelElementCollection/value but ordered and with all elements having the same semanticId
	SubmodelElementList/valueTypeListElement	Attribute of new class SubmodelElementList

Table 15. New, Changed or Removed Constraints

Nc	V3.0 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASd-001	Removed	Constraint AASd-001: In case a referable element is not an identifiable element, this ID is mandatory and used for referring to the element in its name space. For namespace part see AASd-022
	AASd-002	Update	Regular expression added Constraint AASd-002: idShort of Referables shall only feature letters, digits, underscore ("*"); starting mandatory with a letter, i.e. [a-zA-Z][a-zA-Z0-9]*.
	AASd-003	Removed	See AASd-022 Constraint AASd-003: idShort of Referables within the same name space shall be unique (case-sensitive).
	AASd-006	Reformulated	Constraint AASd-006: If both, the value and the valueId of a Qualifier are present, the value needs to be identical to the value of the referenced coded value in Qualifier/valueId.

Nc	V3.0 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASd-005	Reformulated	Constraint AASd-005: If AdministrativeInformation/version is not specified, AdministrativeInformation/revision shall also be unspecified. This means that a revision requires a version. If there is no version, there is no revision either. Revision is optional.
	AASd-007	Reformulated	Constraint AASd-007: If both the Property/value and the Property/valueId are present, the value of Property/value needs to be identical to the value of the referenced coded value in Property/valueId.
	AASd-008	Removed	Constraint AASd-008: The submodel element value of an operation variable shall be of kind=Template.
	AASd-010	Renamed	Renamed and reformulated to AASs-010 Not part of this document any longer but of part security
	AASd-011	Renamed	Renamed and reformulated to AASs-011 Not part of this document any longer but of part security
	AASd-012	Reformulated	Constraint AASd-012: If both the MultiLanguageProperty/value and the MultiLanguageProperty/valueId are present, the meaning must be the same for each string in a specific language, as specified in MultiLanguageProperty/valueId
	AASd-014	Reformulated	Entity was changed Constraint AASd-014: Either the attribute globalAssetId or specificAssetId of an Entity must be set if Entity/entityType is set to "SelfManagedEntity". Otherwise, they do not exist.
	AASd-115	Reformulated	Constraint AASd-115: If a first level child element in a SubmodelElementList does not specify a semanticId, the value is assumed to be identical to SubmodelElementList/semanticIdListElement.
	AASd-118	Reformulated	Constraint AASd-118: If a supplemental semantic ID (HasSemantics/supplementalSemanticId) is defined, there shall also be a main semantic ID (HasSemantics/semanticId).
	AASd-119	Reformulated	Constraint AASd-119: If any Qualifier/kind value of a Qualifiable/qualifier is equal to TemplateQualifier and the qualified element inherits from "hasKind", the qualified element shall be of kind Template (HasKind/kind = "Template").

Nc	V3.0 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
(x)	AASd-020	New	Constraint AASd-020: The value of Property/value shall be consistent to the data type as defined in Property/valueType.
(x)	AASd-021	New	Constraint AASd-021: Every qualifiable can only have one qualifier with the same Qualifier/type.
	AASd-022	New	Added case-sensitivity for idShort (since AASd-003 was removed) Constraint AASd-022: idShort of non-identifiable referables within the same name space shall be unique (case-sensitive)
	AASd-129	Reformulated	Constraint AASd-129: If any Qualifier/kind value of a SubmodelElement/qualifier (attribute qualifier inherited via Qualifiable) is equal to TemplateQualifier, the submodel element shall be part of a submodel template, i.e. a Submodel with Submodel/kind (attribute kind inherited via HasKind) value equal to Template.
	AASd-077	New	Constraint AASd-077: The name of an extension (Extension/name) within HasExtensions needs to be unique.
	AASd-090	New	Constraint AASd-090: For data elements, category (inherited by Referable) shall be one of the following values: CONSTANT, PARAMETER or VARIABLE. Default: VARIABLE
	AASd-107	New	Constraint AASd-107: If a first level child element in a SubmodelElementList has a semanticId, it shall be identical to SubmodelElementList/semanticIdListElement.
	AASd-108	New	Constraint AASd-108: All first level child elements in a SubmodelElementList shall have the same submodel element type as specified in SubmodelElementList/typeValueListElement.
	AASd-109	New	Constraint AASd-109: If SubmodelElementList/typeValueListElement is equal to Property or Range, SubmodelElementList/valueTypeListElement shall be set and all first level child elements in the SubmodelElementList shall have the value type as specified in SubmodelElementList/valueTypeListElement.
	AASd-114	New	Constraint AASd-114: If two first level child elements in a SubmodelElementList have a semanticId, they shall be identical.

Nc	V3.0 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASd-115	New	Constraint AASd-115: If a first level child element in a SubmodelElementList does not specify a semanticId, the value is assumed to be identical to SubmodelElementList/semanticIdListElement.
	AASd-116	New	Constraint AASd-116: "globalAssetId" (case-insensitive) is a reserved key. If used as value for SpecificAssetId/name, IdentifierKeyValuePair/value shall be identical to AssetInformation/globalAssetId.
	AASd-117	New	Constraint AASd-117: idShort of non-identifiable Referables not being a direct child of a SubmodelElementList shall be specified.
	AASd-118	New	Because of new attribute supplementalSemanticId for HasSemantics Constraint AASd-118: If a supplemental semantic ID (HasSemantics/supplementalSemanticId) is defined, there shall also be a main semantic ID (HasSemantics/semanticId).
	AASd-119	New	New Qualifier/kind attribute Constraint AASd-119: If any Qualifier/kind value of a Qualifiable/qualifier is equal to TemplateQualifier and the qualified element inherits from "hasKind", the qualified element shall be of kind Template (HasKind/kind = "Template").
	AASd-120	New	Constraint AASd-120: idShort of submodel elements being a direct child of a SubmodelElementList shall not be specified.
	AASd-121	New	Constraint AASd-121: For References, the value of Key/type of the first key of Reference/keys shall be one of GloballyIdentifiables.
	AASd-122	New	Constraint AASd-122: For external references, i.e. References with Reference/type = ExternalReference, the value of Key/type of the first key of Reference/keys shall be one of GenericGloballyIdentifiables.
	AASd-123	New	Constraint AASd-123: For model references, i.e. References with Reference/type = ModelReference, the value of Key/type of the first key of Reference/keys shall be one of AasIdentifiables.
	AASd-124	New	Constraint AASd-124: For external references, i.e. References with Reference/type = ExternalReference, the last key of Reference/keys shall be either one of GenericGloballyIdentifiables or one of GenericFragmentKeys.

Nc	V3.0 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASd-125	New	Constraint AASd-125: For model references, i.e. References with Reference/type = ModelReference with more than one key in Reference/keys, the value of Key/type of each of the keys following the first key of Reference/keys shall be one of FragmentKeys.
	AASd-126	New	Constraint AASd-126: For model references, i.e. References with Reference/type = ModelReference with more than one key in Reference/keys, the value of Key/type of the last Key in the reference key chain may be one of GenericFragmentKeys, or no key at all shall have a value out of GenericFragmentKeys.
	AASd-127	New	Constraint AASd-127: For model references, i.e. References with Reference/type = ModelReference with more than one key in Reference/keys, a key with Key/type FragmentReference shall be preceded by a key with Key/type File or Blob. All other AAS fragments, i.e. Key/type values out of AasSubmodelElements, do not support fragments.
	AAS-128	New	Constraint AASd-128: For model references, i.e. References with Reference/type = ModelReference, the Key/value of a Key preceded by a Key with Key/type=SubmodelElementList is an integer number denoting the position in the array of the submodel element list.
	AASd-129	New	Necessary as supplement for AASd-119 since SubmodelElement does not inherit from HasKind any longer Constraint AASd-129: If any Qualifier/kind value of a SubmodelElement/qualifier (attribute qualifier inherited via Qualifiable) is equal to TemplateQualifier, the submodel element shall be part of a submodel template, i.e. a Submodel with Submodel/kind (attribute kind inherited via HasKind) value equal to Template.
x	AASd-130	New	ensures that encoding is possible and interoperability between different serializations is possible. Constraint AASd-130: An attribute with data type "string" shall consist of these characters only: ^[\x09\x0A\x0D\x20-\uD7FF\uE000-\uFFFD\u00010000-\u0010FFFF]*$.
(x)	AASd-131	New	Constraint AASd-131: The globalAssetId or at least one specificAssetId shall be defined for AssetInformation.

Nc	V3.0 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
(x)	AASd-133	New	Constraint AASd-133: SpecificAssetId/externalSubjectId shall be a global reference, i.e. Reference/type = GlobalReference.
(x)	AASd-134	New	Constraint AASd-134: For an Operation, the idShort of all inputVariable/value, outputVariable/value and inoutputVariable/value shall be unique.
x	AASd-135	New	Constraint AASd-135: AdministrativeInformation/version shall have a length of maximum 4 characters.
x	AASd-136	New	Constraint AASd-136: AdministrativeInformation/revision shall have a length of maximum 4 characters.

H.4. Changes V3.0 Vs. V3.0RC02

Major changes

- Document split into several documents: Part 1 on metamodel of the AAS (this document), Part 5 on the aasx package exchange format, Part 3 series on the predefined data specifications, and Part 4 (security)
- CHANGED: SubmodelElements do not inherit from HasKind any longer, only Submodel has distinction between submodel template or submodel instance (including update of tAAS-#18 and -#19)
- NEW: New Constraint for valid strings (AASd-130)
- NEW: Length constraints added for many string attributes, in most cases by introducing new string types
- CHANGED: renamed ReferenceTypes/GlobalReference to ReferenceTypes/ExternalReference (text serialization of references and constraints updated accordingly)
- CHANGED: Type of globalAsset is now Identifier and not Reference (AssetInformation and Entity)
- CHANGED: Updated text for submodel element collections
- EDITORIAL: Examples for matching references
- CHANGED: Terms and definitions updated to be conformant to IEC 63278-1 DRAFT, July 2022 (note: this document contains more terms and definitions than IEC 63278-1 and vice versa; not all terms and definitions from IEC 63278-1 are included). Definition of view removed. Definition of service added from Part 2.
- CHANGED: Description of ModellingKind
- NEW: Appendix for Backus-Naur form (BNF), including update of all grammars using BNF and variants for consistent grammar language usage

- ENHANCED: Extended grammar (referredSemanticId) for text serialization of <Reference>
- CHANGED: Grammar on how to define semantic identifiers for metamodel elements of this specification: <Character> definition in <Namespace> (before "an unreserved character permitted by DIN SPEC 91406", now regular expression [a..zA..Z-])
- CHANGED: In some mappings or serializations, the type "Reference" is converted into a single string. In this case, it is now required (instead of just recommended) to use the defined string serialization.
- CHANGED: Type of BlobType changed from "group of bytes" to "base64Binary"
- NEW: Two new terms introduced: coded value and explicit value
- REMOVED: Referable/checksum
- CHANGE: EventElements including all classes introduced for Events set to <<Experimental>>
- CHANGE: Referable/category set to <<Deprecated>>
- CHANGE: AdministrativeInformation Class, not data type
- NEW: New stereotype <<Experimental>> introduced besides <<Deprecated>>
- CHANGED: Qualifier/kind set to <<Experimental>>
- CHANGED: enumeration DataTypeDefXsd for data types for valueType attribute (e..g. in Property) as well as enumeration DataTypeDefRdf (for consistency) + restriction to XML Schema 1.0 (not 1.1.)
- EDITORIAL (REMOVED): Clause on Tooling and Open Source
- EDITORIAL: vector graphics
- NEW: Besides type assets and instance assets now also assets for which this kind of classification is not applicable are supported (conformant to IEC 63278-1)
- Bugfixes:
- Corrected Japanese example for xd:string
- UML figure for HasExtensions did not show inheritance (table for Extension was correct)
- HasDataSpecification/embeddedDataSpecs 0..* not 0..1
- Qualifier is and never was abstract (Constraint was), table was correct, UML corrected
- Correct AASd-051: VIEW no longer supported
- Type of SubmodelElementList/typeValueListElement corrected to AasSubmodelElements (before SubmodelElementElements)
- Correct text serialization of <Reference>
- Added missing table for enumeration ReferenceTypes
- AASd-117 it is not the SubmodelElementList having no idShort but its childs

- KeyTypes Table: set of AasReferables added

Smaller changes

- EDITORIAL: Qualifier description updated
- EDITORIAL: Reformulation of constraints dealing with References and Key/type
- EDITORIAL: Examples now with https: and not http:
- EDITORIAL: Footnotes reused
- EDITORIAL: Added explanation for annotated relationship elements
- EDITORIAL: asset type and asset instance now type asset and instance asset (conformant to IEC 63278-1)
- EDITORIAL: example for langString serialization changed (table 6)

H.5. Metamodel Changes V3.0 vs. V3.0RC02

Table 16. Changes

nc	V3.0 Change w.r.t. V3.0RC02	Comment
	AdministrativeInformation	Stereotype <<dataType>> removed
	AssetInformation/externalSubjectId	Not mandatory any longer; now optional
x	AssetInformation/globalAssetId	Type changed from Reference to Identifier
	AssetKind/Type	Updated description of value "Type" of enumeration "AssetKind" conformant to IEC 63278-1
	AssetKind/Instance	Updated description of value "Instance" of enumeration "AssetKind" - conformant to IEC 63278-1
	BasicEventElement	Set to <<Experimental>>
x	BasicEventElement/messageTopic	Type changed from string to MessageTopicType (i.e. length constraint added, maximum 255 characters)
x	BasicEventElement/minInterval	Type changed from dateTime to duration
x	BasicEventElement/maxInterval	Type changed from dateTime to duration
(x) [4]	BlobType	Primitive changed from "group of bytes" to Base64Binary
	ContentType	Maximum length defined: 100 characters

nc	V3.0 Change w.r.t. V3.0RC02	Comment
	decimalBuildInTypes	Removed
	durationBuildInTypes	Removed
x	DataSpecification/description	Type changed from LangStringSet to MultiLanguageTextType; length constraint added
x	DataTypeDefRdf/langString	Added prefix "rdf:", i.e. change from langString to rdf:langString
x	DateTypeDefXsd/dateTimeStamp	Removed since not part of XML Schema 1.0
x	DataTypeDefXsd/dayTimeDuration	Removed since not part of XML Schema 1.0
x	DataTypeDefXsd/yearMonthDuration	Removed since not part of XML Schema 1.0
	Direction	Set to <<Experimental>>
x	Entity/globalAssetId	Type changed from Reference to Identifier
x	Extension/name	Type changed from string to NameType; length constraint added
	EventElement	Set to <<Experimental>>
	EventElement/duration	Type changed from dateTimeStamp to dateTime
	EventPayload	Set to <<Experimental>>
	EventPayload/payload	Type changed from string to BlobType
	EventPayload/source	Type changed from ModelReference(Referable) to ModelReference(EventElement)
	EventPayload/timestamp	Type changed from dateTimeStamp to dateTime because restriction to types of XML Schema 1.0 that does not contain dateTimeStamp. dateTimeStamp is a derived type of dateTime in XML Schema 1.1
	EventPayload/topic	Type changed from string to MessageTopicType (i.e. length constraint added, maximum 255 characters)
x	HasKind/kind	Type changed from ModelingKind to ModellingKind
x	Identifier	Maximum length defined: 2,000 characters
x	Key/value	Type changed from string to Identifier; length constraint added

nc	V3.0 Change w.r.t. V3.0RC02	Comment
x	MultiLanguageProperty/value	Type changed from LangStringSet to MultiLanguageTextType; length constraint added
x	PathType	Same as Identifier; length constraint added
	PrimitiveTypes	Removed
	Qualifier/kind	Set to <<Experimental>>
x	QualifierType	Type changed from string to NameType; length constraint added
	rdfBuildInTypes	Removed
x	Referable/category	Type changed from string to NameType; length constraint added Category set to deprecated
x	Referable/checksum	Removed
x	Referable/displayName	Type changed from string to MultiLanguageNameType; length constraint added Text how to select a suitable display name removed; now explained in Table 2
x	Referable/description	Type changed from string to MultiLanguageTextType; length constraint added
x	Referable/idShort	Type changed from string to NameType; length constraint added
x	ReferenceTypes/GlobalReference	Renamed to ExternalReference
	Resource	Stereotype <<DataType>> removed.
	StateOfEvent	Set to <<Experimental>>
x	SpecificAssetId/name	Type changed from string to LabelType (a string with length constraint)
x	SpecificAssetId/value	Type changed from string to Identifier (because of length constraint)
x[5]	SubmodelElement/kind	Removed. SubmodelElement does not inherit from HasKind any longer

Table 17. New Elements in Metamodel

nc	V3.0 vs. V3.0RC02 New Elements	Comment
	AdministrativeInformation/creator	New optional attribute
	AdministrativeInformation/templateId	New optional attribute
	AssetInformation/assetType	New optional attribute
	AssetKind/NotApplicable	New enumeration value
	LabelType	New string type with maximum 64 characters
	MessageTopicType	New string type with maximum 255 characters
	ModellingKind	Renamed enumeration, before: ModelingKind
	MultiLanguageNameType	Substitute for LangStringSet with short multi-language strings, maximum 64 characters
	MultiLanguageTextType	Substitute for LangStringSet with long multi-language strings, maximum 64 characters
	NameType	New string type with maximum 128 characters
	ReferenceTypes/ExternalReference	Enumeration value: before: GlobalReference
	RevisionType	New type for *AdministrativeInformation/version* with length constraints and regular expression
	ShortNameType	New string type with maximum 64 characters
	VersionType	New type for *AdministrativeInformation/revision* with length constraints and regular expression

Table 18. New, Changed or Removed Constraints

Nc	V3.0 vs. V3.0RC02	New, Update, Removed, Reformulated	Comment
		EDITORIAL	The following constraints were also updated with minor editorial changes: Constraints AASd-121, AASd-122, AASd-123, AASd-124, AASd-125, AASd-126, AASd-127, AASd-129

Nc	V3.0 vs. V3.0RC02	New, Update, Removed, Reformulated	Comment
	AASd-002	Reformulated	Now min length 1, before 2 Constraint AASd-002: idShort of Referables shall only feature letters, digits, underscore ("*"); starting mandatory with a letter, i.e. [a-zA-Z][a-zA-Z0-9]*.
	AASd-003	Removed	See AASd-022 Constraint AASd-003: idShort of Referables within the same name space shall be unique (case-sensitive).
	AASd-005	Reformulated	Constraint AASd-005: If AdministrativeInformation/version is not specified, AdministrativeInformation/revision shall also be unspecified. This means that a revision requires a version. If there is no version, there is no revision. Revision is optional.
	AASd-006	Reformulated	Constraint AASd-006: If both, the value and the valueId of a Qualifier are present, the value needs to be identical to the value of the referenced coded value in Qualifier/valueId.
	AASd-007	Reformulated	Constraint AASd-007: If both the Property/value and the Property/valueId are present, the value of Property/value needs to be identical to the value of the referenced coded value in Property/valueId.
	AASd-012	Reformulated	Constraint AASd-012: if both the MultiLanguageProperty/value and the MultiLanguageProperty/valueId are present, the meaning must be the same for each string in a specific language, as specified in MultiLanguageProperty/valueId.
	AASd-020	Reformulated	Constraint AASd-020: The value of Qualifier/value shall be consistent with the data type as defined in Qualifier/valueType.
	AASd-022	Update	Added case-sensitivity for idShort (since AASd-003 was removed) Constraint AASd-022: idShort of non-identifiable Referables within the same name space shall be unique (case-sensitive)
	AASd-027	Removed	Not needed any longer since Type of idShort was changed to NameType and NameType has a maximum length of 128 characters Constraint AASd-027: idShort of Referables shall have a maximum length of 128 characters

Nc	V3.0 vs. V3.0RC02	New, Update, Removed, Reformulated	Comment
	AASd-077	Reformulated	Constraint AASd-077: the name of an extension (Extension/name) within HasExtensions needs to be unique
	AASd-100	Removed	Since new string types with length constraints were added, this constraint is no longer needed Constraint AASd-100: An attribute with data type "string" is not allowed to be empty
	AASd-109	Reformulated	Constraint AASd-109: If SubmodelElementList/typeValueListElement is equal to Property or Range, SubmodelElementList/-valueTypeListElement shall be set and all first level child elements in the SubmodelElementList shall have the value type as specified in SubmodelElementList/valueTypeListElement.
	AASd-115	Reformulated	Constraint AASd-115: If a first level child element in a SubmodelElementList does not specify a semanticId, the value is assumed to be identical to SubmodelElementList/-semanticIdListElement.
	AASd-117	Bugfix	Constraint AASd-117: idShort of non-identifiable Referables not being a direct child of a SubmodelElementList shall be specified.
	AASd-118	Reformulated	Constraint AASd-118: If a supplemental semantic ID (HasSemantics/supplementalSemanticId) is defined, there shall also be a main semantic ID (HasSemantics/semanticId).
	AASd-119	Reformulated	Constraint AASd-119: If any Qualifier/kind value of a Qualifiable/qualifier is equal to TemplateQualifier and the qualified element inherits from "hasKind", the qualified element shall be of kind Template (HasKind/kind = "Template").
	AASd-120	Reformulated	Constraint AASd-120: idShort of submodel elements being a direct child of a SubmodelElementList shall not be specified.
	AASd-121	Reformulated	Constraint AASd-121: For References the value of Key/type of the first key of Reference/keys shall be one of GloballyIdentifiables.
	AASd-122	Reformulated	Constraint AASd-122: For external references, i.e. References with Reference/type = ExternalReference, the value of Key/type of the first key of Reference/keys shall be one of GenericGloballyIdentifiables.

Nc	V3.0 vs. V3.0RC02	New, Update, Removed, Reformulated	Comment
	AASd-123	Reformulated	Constraint AASd-123: For model references, i.e. References with Reference/type = ModelReference, the value of Key/type of the first key of Reference/keys shall be one of AasIdentifiables.
	AASd-124	Reformulated	Constraint AASd-124: For external references, i.e. References with Reference/type = ExternalReference, the last key of Reference/keys shall be either one of GenericGloballyIdentifiables or one of GenericFragmentKeys.
	AASd-125	Reformulated	Constraint AASd-125: For model references, i.e. References with Reference/type = ModelReference with more than one key in Reference/keys, the value of Key/type of each of the keys following the first key of Reference/keys shall be one of FragmentKeys.
	AASd-126	Reformulated	Constraint AASd-126: For model references, i.e. References with Reference/type = ModelReference with more than one key in Reference/keys, the value of Key/type of the last Key in the reference key chain may be one of GenericFragmentKeys, or no key at all shall have a value out of GenericFragmentKeys.
	AASd-127	Reformulated	Constraint AASd-127: For model references, i.e. References with Reference/type = ModelReference with more than one key in Reference/keys, a key with Key/type FragmentReference shall be preceded by a key with Key/type File or Blob. All other AAS fragments, i.e. Key/type values out of AasSubmodelElements, do not support fragments.
	AASd-129	New	Necessary as supplement for AASd-119, since SubmodelElement does not inherit from HasKind any longer Constraint AASd-129: If any Qualifier/kind value of a SubmodelElement/qualifier (attribute qualifier inherited via Qualifiable) is equal to TemplateQualifier, the submodel element shall be part of a submodel template, i.e. a Submodel with Submodel/kind (attribute kind inherited via HasKind) value to Template.

Nc	V3.0 vs. V3.0RC02	New, Update, Removed, Reformulated	Comment
x	AASd-130	New	Ensures that encoding is possible and interoperability between different serializations is possible. Constraint AASd-130: An attribute with data type "string" shall consist of these characters only: ^[\x09\x0A\x0D\x20-\uD7FF\uE000-\uFFFD\u00010000-\u0010FFFF]*$.
(x)	AASd-131	New	Constraint AASd-131: The globalAssetId or at least one specificAssetId shall be defined for AssetInformation.
(x)	AASd-133	New	Constraint AASd-133: specificAssetId/externalSubjectId shall be a global reference, i.e. Reference/type = ExternalReference.
(x)	AASd-134	New	Constraint AASd-134: For an operation, the idShort of all inputVariable/value, outputVariable/value, and inoutputVariable/value shall be unique.
	AASd-051	Removed	Since category is deprecated, this constraint was removed. Views are no longer supported by metamodel Constraint AASd-051: A *ConceptDescription* shall have one of the following categories: VALUE, PROPERTY, REFERENCE, DOCUMENT, CAPABILITY, RELATIONSHIP, COLLECTION, FUNCTION, EVENT, ENTITY, APPLICATION_CLASS, QUALIFIER. Default: PROPERTY.

H.6. Changes V3.0RC02 vs. V2.0.1

H.7. Metamodel Changes V3.0RC02 vs. V2.0.1 w/o Security Part

Note: if you already implemented the changes made in V3.0RC01, please refer to the corresponding clause in the this annex. This annex is for readers familiar with V2.0.x only.

Major changes:

- CHANGED: Split of SubmodelElementCollection into SubmodelElementList (with orderRelevant) and SubmodelElementCollection

- CHANGED: Reference type and referredSemanticId added to Reference; Local and Parent attributes removed from Reference; logical enumeration concept updated. Some renaming and some new enumerations. Constraint for References.
- CHANGED: Reference/type now as optional part of string serialization of reference
- CHANGED: idType from identifier removed, ID now string
- CHANGED: idShort of Referable now optional + Constraints added with respect to ID and idShort, includes that idShort of Submodels etc. no longer need to be unique in the context of an Asset Administration Shell
- CHANGED: semanticId no longer mandatory for SubmodelElement
- CHANGED: Revised concept on handling of Asset and assetIdentificationModel (assetInformation), Asset removed, no more Asset/billOfMaterial. any longer. Specific asset IDs added.
- REMOVED: ConceptDictionaries removed, because no longer supported
- REMOVED: Views removed, because no longer supported
- NEW: Event and BasicEvent updated and renamed to EventElement and BasicEventElement
- NEW: Checksum introduced for Referables
- REMOVED: security attribute removed from Asset Administration Shell; access control remains part of the specification
- ENHANCED: DataTypeIEC61360 extended with values for IRI, IRDI, BLOB, FILE + corresponding new constraints added
- ENHANCED: Removed and split into DataTypeDefXsd and DataTypeDefRdf. Some types are excluded and not supported
- CHANGED: Mapping rules for different serializations + schemata + example in different serializations extracted and no longer part of this specification
- EDITORIAL: Text updated, no kind column any longer in class tables, instead notation of ModelReference<{Referable}>. New table for Primitives/Data Types
- EDITORIAL: New clause "Introduction"
- EDITORIAL: New clause "Matching strategies for semantic identifiers"
- NEW: Constraints implicitly contained in text were formalized and numbered (normative)
- NEW: Environment explicitly part of UML (was part of serializations from the beginning)
- NEW: supplemental Semantic IDs
- NEW: Qualifier/kind (TemplateQualifier, ConceptQualifier, ValueQualifier)

Bugfixes:

- bugfix annotation AnnotatedRelationship is of type aggr and not ref* (diagram was correct)
- bugfix specification of ValueList and ValueReferencePairType, no data types, normal classes
- bugfix table specifications w.r.t. kind of attribute (from aggr to attr – column kind was removed, see above)
- bugfix data type specification LangStringSet (no diagram and table any longer)
- bugfix enumeration ReferableElements, no ConceptDictionary any longer + new elements like new submodel elements SubmodelElementList added.

Note: ReferableElements was substituted by AasSubmodelElements and Aas Identifiables.

- Entity/globalAssetId diagram (table was correct): Type change from reference of Reference* to Reference

Table 19. Changes w/o Security

nc	V3.0RC02 Change w.r.t. V2.0.1	Comment
	AdministrativeInformation	Bugfix: Stereotype "DataType" added
	AnnotatedRelationship/annotation	Bugfix: Type changed from ModelReference<DataElement> to DataElement
	anySimpleTypeDef	Type removed, was no longer used in any class definition, was mentioned in text only
x	Asset	Removed, asset referenced via AssetInformation/globalAssetId only
x	AssetAdministrationShell/asset	Removed, substituted by AssetAdministrationShell/assetInformation (no reference any longer, instead now aggregation)
x	AssetAdministrationShell/conceptDictionaries	Removed
x	AssetAdministrationShell/security	Removed Note: Security is still part of the Asset Administration Shell, but the Asset Administration Shell and its elements are referenced from Security.
	AssetAdministrationShell/view	Removed, views no longer supported

nc	V3.0RC02 Change w.r.t. V2.0.1	Comment
x	BasicEvent	Renamed to BasicEventElement
x	ConceptDictionary	Removed
x	Constraint	Abstract class removed. Formula now used in Security part only
(x)	DataTypeDef	Removed and split into DataTypeDefXsd and DataTypeDefRdf; some types excluded and not supported (see notes in corresponding clause) Before: just string allowing any xsd simple type as string + added prefix xs: or rdf:, resp. to every value in enumeration
x	Entity/asset	Removed, substituted by Entity/globalAssetId and Entity/specificAssetId
x	Event	Renamed to EventElement
x	Extension/refersTo	Type changed from Reference to ModelReference
x	Extension/valueType	Type changed from DataTypeDef to DataTypeDefXsd
x	File/mimeType	Renamed to contentType + Type changed from MimeType to ContentType
x	Formula	Now abstract class Formula now used in Security part only
x	Formula/dependsOn	Removed, since formula language not yet defined
x	Identifiable/identification	Removed Substituted by Identifiable/id
x	IdentifiableElements	Renamed to AasIdentifiables
x	Identifier	Type changed Before struct class with two attributes: id and idType; now string data type only
	IdentifierType	Enumeration removed, because no idType any longer
x	Key/idType	removed

nc	V3.0RC02 Change w.r.t. V2.0.1	Comment
x	Key/local	Local attribute removed
(x)	KeyElements	Renamed to KeyTypes Note: the elements remain, except for new SubmodelElementList and renamed submodel elements Event and BasicEvent to EventElement and BasicEventElement
	KeyType	Enumeration removed because no Key/idType any longer
	LocalKeyType	Enumeration removed because no Key/idType any longer
x	MimeType	Type name changed to ContentType
	Property/valueType	Type changed from DataTypeDef to DataTypeDefXsd
x	Qualifiable/qualifier	Type changed from Constraint to Qualifier
	Qualifier	No longer inherits from abstract class "Constraint"
	Qualifier/valueType	Type changed from DataTypeDef to DataTypeDefXsd
	Range/valueType	Type changed from DataTypeDef to DataTypeDefXsd
	Referable/idShort	Now optional, was mandatory
x	Referable/parent	Parent attribute removed
x	ReferableElements	Substituted with enumeration AasSubmodelElements and AasIdentifiables
x	ReferableElements/AccessPermissionRule	Removed from enumeration, AccessPermissionRule is no longer referable Not part of new AasReferableNonIdentifiables
x	ReferableElement/BasicEvent	Renamed to BasicEventElement Now part of AasSubmodelElements

nc	V3.0RC02 Change w.r.t. V2.0.1	Comment
(x)	ReferablesElements/ConceptDictionary	Bugfix: ConceptDictionary removed from enumeration, since ConceptDictionary no longer part of specification Not part of new KeyTypes
x	ReferableElements/Event	Renamed to EventElement Now part of AasSubmodelElements
	RelationshipElement/first	Type changes from model reference Referable to Reference (global or model reference)
	RelationshipElement/second	Type changes from model reference Referable to Reference (global or model reference)
	ValueDataType	Before as specified via DataTypeDef, now any xsd atomic type as specified via DataTypeDefXsd
x	View	Removed

Table 20. New Elements in Metamodel w/o Security

nc	V3.0RC02 vs. V2.0.1 New Elements	Comment
	AasSubmodelElements	New enumeration used for References Before: ReferableElements
	AasIdentifiables	New enumeration used for References, includes abstract Identifiable Before: Identifiables
	AasReferableNonIdentifiables	New enumeration used for References
	AasReferables	New enumeration used for References, includes abstract Referable

nc	V3.0RC02 vs. V2.0.1 New Elements	Comment
x	AssetAdministrationShell/assetInformation	substitute for AssetAdministrationShell/asset; no reference any longer, instead now aggregation
	AssetInformation	with attributes/functionality from former class Asset, because not specific to Asset but to AAS
	AssetInformation/assetKind	Former Asset/assetKind
	AssetInformation/globalAssetId	Former Asset/identification/id
	AssetInformation/specificAssetId	Former Asset/assetIdentificationModel
	AssetInformation/thumbnail	Optional Attribute of new class AssetInformation that was not available in Asset class before
	BasicEventElement	Former name: BasicEvent Was part of non-normative part before
	BasicEventElement/direction	Former name: BasicEvent/observed Was part of non-normative part before
	BasicEventElement/lastUpdate	Was part of non-normative part before
	BasicEventElement/messageBroker	Was part of non-normative part before
	BasicEventElement/messageTopic	Was part of non-normative part before
	BasicEventElement/minInterval	Was part of non-normative part before
	BasicEventElement/maxInterval	Was part of non-normative part before

nc	V3.0RC02 vs. V2.0.1 New Elements	Comment
	BasicEventElement/observed	Was part of non-normative part before
	BasicEventElement/state	Was part of non-normative part before
	ContentType	Former name: MimeType
	dateTimeStamp	New data type for metamodel as used in EventPayload
	DataTypeDefRdf	Enumeration for types of Rdf + prefix rdf: added to every value in enumeration
	DataTypeDefXsd	Enumeration consisting of enumerations decimalBuildInTypes, durationBuildInTypes, PrimitiveTypes that correspond to anySimpleTypes of xsd. + added prefix xs: to every value in enumeration
	Direction	New enumeration for BasicEventElement
	Environment	New class for entry point for Asset Administration Shells, submodels and concept descriptions
	EventElement	Former name: Event
	EventPayload	New class for event payload
	EventPayload/observableSemanticId	Was part of non-normative part before
	EventPayload/payload	Was part of non-normative part before
	EventPayload/source	Was part of non-normative part before
	EventPayload/sourceSemanticId	Was part of non-normative part before

nc	V3.0RC02 vs. V2.0.1 New Elements	Comment
	EventPayload/subjectId	Was part of non-normative part before
	EventPayload/timestamp	Was part of non-normative part before
	Extension	New class, part of new abstract class HasExtensions
	FragmentKeys	New enumeration used for References
	GenericFragmentKeys	New enumeration used for References
	GenericGloballyIdentifiers	New enumeration used for References
	GloballyIdentifiables	New enumeration used for References
	HasExtensions	New abstract class, inherited by Referable
	HasSemantics/supplementalSemanticId	New attribute
	Identifiable/id	Substitute for Identifiable/identification
	IdentifierKeyValuePair	New class for AssetInformation/specificAssetId
	KeyTypes	Before: KeyElements New submodel element SubmodelElementList added, submodel elements Event and BasicEvent to EventElement and BasicEventElement renamed
	Qualifier/kind	New attribute for Qualifier
	QualifierKind	New enumeration for Qualifier/kind
	PrimitiveTypes	Enumeration for DataTypeDefXsd
	Referable/checksum	New optional attribute for all referables

nc	V3.0RC02 vs. V2.0.1 New Elements	Comment
	Referable/displayName	New optional attribute for all referables
	Reference/referredSemanticId	New optional attribute for Reference
x	Reference/type	New mandatory attribute for Reference
	ReferenceTypes	New enumeration for Reference/type
	StateOfEvent	New enumeration for BasicEventElement
	SpecificAssetId	New type for AssetInformation/specificAssetId
	SpecificAssetId/name	New type for AssetInformation/specificAssetId
	SpecificAssetId/value	New type for AssetInformation/specificAssetId
	SpecificAssetId/externalSubjectId	New type for AssetInformation/specificAssetId See Attribute Based Access Control (ABAC) for subject concept
	SubmodelElementElements	Enumeration for submodel elements (split of ReferableElements)
	SubmodelElementList	Before SubmodelElementCollection was used for lists and structs
	SubmodelElementList/orderRelevant	Similar to SubmodelElementCollection/ordered
	SubmodelElementList/value	Similar to SubmodelElementCollection/value but ordered and with all elements having the same semanticId
	SubmodelElementList/semanticIdListElement	Attribute of new class SubmodelElementList

nc	V3.0RC02 vs. V2.0.1 New Elements	Comment
	SubmodelElementList/typeValueListElement	Attribute of new class SubmodelElementList
	SubmodelElementList/valueTypeListElement	Attribute of new class SubmodelElementList

Table 21. New, Changed or Removed Constraints w/o Security

Nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASd-001	Removed	Constraint AASd-001: In case of a referable element not being an identifiable element this ID is mandatory and used for referring to the element in its name space. For namespace part see AASd-022
[6]	AASd-003	Update	idShort is case-sensitive and not case-insensitive Constraint AASd-003: *idShort* of *Referables* shall be matched case-sensitive.
	AASd-005	Reformulated	Constraint AASd-005: If *AdministrativeInformation*/version is not specified than also *AdministrativeInformation/revision* shall be unspecified. This means, a revision requires a version. if there is no version there is no revision neither. Revision is optional.
	AASd-008	Removed	Constraint AASd-008: The submodel element value of an operation variable shall be of kind=Template.
	AASd-010	Renamed	Renamed and reformulated to AASs-010 (see NEW)
	AASd-011	Renamed	Renamed and reformulated to AASs-011 (see NEW)
	AASd-012	Reformulated	Constraint AASd-012: If both, the *MultiLanguageProperty/value* and the *MultiLanguageProperty/valueId* are present then for each string in a specific language the meaning must be the same as specified in *MultiLanguageProperty/valueId*

Nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASd-014	Reformulated	Entity was changed Constraint AASd-014: Either the attribute *globalAssetId* or *specificAssetId* of an *Entity* must be set if *Entity/entityType* is set to "*SelfManagedEntity*". They are not existing otherwise.
(x)	AASd-020	New	Constraint AASd-020: The value of Property/*value* shall be consistent to the data type as defined in Property/*valueType*.
(x)	AASd-021	New	Constraint AASd-021: Every qualifiable can only have one qualifier with the same *Qualifier/type.*
	AASd-023	Removed	No Asset any longer that can be referenced as alternative to global reference Constraint AASd-023: AssetInformation/globalAssetId either is a reference to an Asset object or a global reference.
x	AASd-027	New	Constraint AASd-027: *idShort* of *Referables* shall have a maximum length of 128 characters.
x	AASd-076	Removed	Substituted by AASc-002; simplified, no reference to concept description
	AASd-077	New	Constraint AASd-077: The name of an extension within HasExtensions needs to be unique.
x	AASd-076	Removed	Substituted by AASc-002; simplified, no reference to concept description
	AASd-077	New	Constraint AASd-077: The name of an extension within HasExtensions needs to be unique.
	AASd-090	Update	Exception: File and Blob data elements removed. Reformulated. Constraint AASd-090: For data elements category (inherited by Referable) shall be one of the following values: CONSTANT, PARAMETER or VARIABLE. Default: VARIABLE
	AASd-100	New	Constraint AASd-100: An attribute with data type "string" is not allowed to be empty.
	AASd-107	New	Constraint AASd-107: If a first level child element in a SubmodelElementList has a semanticId it shall be identical to SubmodelElementList/semanticIdListElement.

Nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASd-108	New	Constraint AASd-108: All first level child elements in a SubmodelElementList shall have the same submodel element type as specified in SubmodelElementList/typeValueListElement.
	AASd-109	New	Constraint AASd-109: If SubmodelElementList/typeValueListElement equal to Property or Range SubmodelElementList/valueTypeListElement shall be set and all first level child elements in the SubmodelElementList shall have the the value type as specified in SubmodelElementList/valueTypeListElement.
	AASd-114	New	Constraint AASd-114: If two first level child elements in a SubmodelElementList have a semanticId then they shall be identical.
	AASd-115	New	Constraint AASd-115: If a first level child element in a SubmodelElementList does not specify a semanticId then the value is assumed to be identical to SubmodelElementList/semanticIdListElement.
	AASd-116	New	Constraint AASd-116: "globalAssetId" (case-insensitive) is a reserved key. If used as value for SpecificAssetId/name IdentifierKeyValuePair/value shall be identical to AssetInformation/globalAssetId.
	AASd-117	New	Needed because Referable/idShort now optional Constraint AASd-117: idShort of non-identifiable Referables not equal to SubmodelElementList shall be specified (i.e. idShort is mandatory for all Referables except for SubmodelElementLists and all Identifiables).
	AASd-118	New	Because of new attribute supplementalSemanticId for HasSemantics Constraint AASd-118: If there is a supplemental semantic ID (HasSemantics/supplementalSemanticId) defined then there shall be also a main semantic ID (HasSemantics/semanticId).
	AASd-119	New	New qualifier/kind attribute Constraint AASd-119: If any Qualifier/kind value of a Qualifiable/qualifier is equal to TemplateQualifier and the qualified element inherits from "hasKind" then the qualified element shall be of kind Template (HasKind/kind = "Template").

Nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASd-120	New	For new submodel element SubmodelElementList Constraint AASD-120: idShort of submodel elements within a SubmodelElementList shall not be specified.
	AASd-121	New	Constraint AASd-121: For References the type of the first key of Reference/keys shall be one of GloballyIdentifiables.
	AASd-122	New	Constraint AASd-122: For global references, i.e. References with Reference/type = GlobalReference, the type of the first key of Reference/keys shall be one of GenericGloballyIdentifiables.
	AASd-123	New	Constraint AASd-123: For model references, i.e. References with Reference/type = ModelReference, the type of the first key of Reference/keys shall be one of AasIdentifiables.
	AASd-124	New	Constraint AASd-124: For global references, i.e. References with Reference/type = GlobalReference, the last key of Reference/keys shall be either one of GenericGloballyIdentifiables or one of GenericFragmentKeys.
	AASd-125	New	Constraint AASd-125: For model references, i.e. References with Reference/type = ModelReference, with more than one key in Reference/keys the type of the keys following the first key of Reference/keys shall be one of FragmentKeys.
	AASd-126	New	Constraint AASd-126: For model references, i.e. References with Reference/type = ModelReference, with more than one key in Reference/keys the type of the last Key in the reference key chain may be one of GenericFragmentKeys or no key at all shall have a value out of GenericFragmentKey.
	AASd-127	New	Constraint AASd-127: For model references, i.e. References with Reference/type = ModelReference, with more than one key in Reference/keys a key with type FragmentReference shall be preceeded by a key with type File or Blob. All other AAS fragments, i.e. type values out of AasSubmodelElements, do not support fragments.

Nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AA-128	New	Constraint AASd-128: For model references, i.e. References with Reference/type = ModelReference, the Key/value of a Key preceeded by a Key with Key/type=SubmodelElementList is an integer number denoting the position in the array of the submodel element list.

H.8. Metamodel Changes V3.0RC02 vs. V2.0.1 – Data Specification IEC61360

Table 22. Changes w.r.t. Data Specification IEC61360

nc	V3.0RC02 Change w.r.t. V2.0.1	Comment
	DataSpecification	Stereotype <<Template>> added + does not inherit from Identifiable any longer because Data Specification are handled in a different way Some attributes are added to DataSpecification as new attributes like id, administration and description.(see separate entries)
	DataSpecification/category	Removed, was inherited before by Identifiable
	DataSpecification/displayName	Removed, was inherited before by Identifiable
	DataSpecification/idShort	Removed, was inherited before by Identifiable
x	DataSpecificationIEC61360/value	Type changed from ValueDataType to string
	DataSpecificationIEC61360/valueId	Removed, the valueId is identical to the ID of the concept description
	DataSpecificationContent	Stereotype <<Template>> added
x	DataTypeIEC61360	Some new values were added: BLOB, FILE, HTML, IRDI. URL renamed to IRI. See separate entries for individual changes.
x	DataTypeIEC61360/URL	Renamed to IRI

nc	V3.0RC02 Change w.r.t. V2.0.1	Comment
	ValueList/valueReferencePairs	Bugfix, was ValueList/valueReferencePairTypes before
x	ValueReferencePair/value	Type changed from ValueDataType to string

Table 23. New Elements in Metamodel DataSpecification IEC61360

nc	V3.0RC02 vs. V2.0.1	Comment
	DataSpecification/administration	Was inherited before by Identifiable
	DataSpecification/id	Was inherited before by Identifiable
	DataSpecification/desciption	Was inherited before by Identifiable
	DataTypeIEC61360/BLOB	New value
	DataTypeIEC61360/FILE	New value
	DataTypeIEC61360/HTML	New value
	DataTypeIEC61360/IRDI	New value
	DataTypeIEC61360/IRI	Converted Iri to CamelCase and renamed to Iri from URL

Table 24. New, Changed or Removed Constraints Data Specification IEC61360

nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASc-002	New	Updated version of AASd-076, renamed to AASc-002 because applicable to data specification IEC61360 Constraint AASc-002: DataSpecificationIEC61360/preferredName shall be provided at least in English
(x)	AASc-003	New	Constraint AASc-003: For a ConceptDescription with category VALUE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) DataSpecificationIEC61360/value shall be set.

nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
(x)	AASc-004	New	Constraint AASc-004: For a ConceptDescription with category PROPERTY or VALUE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType is mandatory and shall be defined.
(x)	AASc-005	New	Constraint AASc-005: For a ConceptDescription with category REFERENCE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType is STRING by default.
(x)	AASc-006	New	Constraint AASc-006: For a ConceptDescription with category DOCUMENT using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType shall be one of the following values: STRING or URL.
(x)	AASc-007	New	Constraint AASc-007: For a ConceptDescription with category QUALIFIER_TYPE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType is mandatory and shall be defined.
(x)	AASc-008	New	Constraint AASc-008: For a ConceptDescriptions except for a ConceptDescription of category VALUE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/definition is mandatory and shall be defined at least in English.
(x)	AASc-009	New	Constraint AASc-009: If DataSpecificationIEC61360/dataType one of: INTEGER_MEASURE, REAL_MEASURE, RATIONAL_MEASURE, INTEGER_CURRENCY, REAL_CURRENCY, then DataSpecificationIEC61360/unit or DataSpecificationIEC61360/unitId shall be defined.

nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
(x)	AASc-010	New	Constraint AASc-010: If DataSpecificationIEC61360/value is not empty then DataSpecificationIEC61360/valueList shall be empty and vice versa

H.9. Metamodel Changes V3.0RC02 vs. V2.0.1 – Security Part

Changes:

- Removed, because deprecated: policy decision point, policy enforcement point, and policy information points are not part of information model but of server infrastructure hosting the Asset Administration Shells
- Removed: Certificate Handling not part of information model but of server infrastructure hosting the Asset Administration Shells

Table 25. Changes w.r.t. Security

nc	V3.0RC02 Change w.r.t. V2.0.1	Comment
x	AccessControlPolicyPoints/policyAdministrationoint	Type changed from PolicyAdministrationPoint to AccessControl
x	AccessControlPolicyPoints/policyDecisionPoint	Removed
x	AccessControlPolicyPoints/policyEnforcementPoint	Removed
x	AccessControlPolicyPoints/policyInformationPoint	Removed
x	AccessPermissionRule	No longer inherits from Referable No longer inherits from Qualifiable
x	BlobCertificate	Removed
x	Certificate	Removed
x	Formula	Now abstract class, only used in security part (no longer used in Qualifiables)
x	Formula/dependsOn	Removed attribute
x	PolicyAdministrationPoint	Removed

nc	V3.0RC02 Change w.r.t. V2.0.1	Comment
x	policyDecisionPoint	Removed
x	policyEnforcementPoint	Removed
x	policyInformationPoints	Removed
x	Security/certificate	Removed
x	Security/requiredCertificateExtension	Removed

Table 26. New Elements in Metamodel Security

nc	V3.0RC02 vs. V2.0.1	Comment
	AccessPermissionRule/constraint	Substitute for inherited attributes from Qualifiable

Table 27. New, Changed or Removed Constraints Security

nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASd-015	Removed	Renamed to AASs-015 (see NEW)
	AASs-009	Removed	Removed since class PolicyAdministrationPoint was removed Constraint AASs-009: either there is an external policy administration point endpoint defined (PolicyAdministrationPoint/externalPolicyDecisionPoints=true) or the AAS has its own access control
	AASs-010	NEW	Reformulation of AASd-010 Constraint AASs-010: the property referenced in Permission/permission shall have the category "CONSTANT".
	AASs-011	NEW	Reformulation of AASd-011 Constraint AASs-011: the property referenced in Permission/permission shall be part of the submodel that is referenced within the "selectablePermissions" attribute of "AccessControl".

nc	V3.0RC02 vs. V2.0.1	New, Update, Removed, Reformulated	Comment
	AASs-015	NEW	Constraint AASs-015: every data element in SubjectAttributes/subjectAttributes shall be part of the submodel that is referenced within the "selectableSubjectAttributes" attribute of "AccessControl".

H.10. Changes V3.0RC02 vs. V3.0RC01

H.11. Metamodel Changes V3.0RC02 vs. V3.0RC01 w/o Security Part

Major changes:

- CHANGED: Split of SubmodelElementCollection into SubmodelElementList (with orderRelevant) and SubmodelElementCollection
- CHANGED: Reference type and referredSemanticId added to Reference; Local and Parent attributes removed from Reference. Logical enumeration concept updated. Some renaming; constraints added for references
- CHANGED: Reference/type now as optional part of string serialization of reference
- CHANGED: idType from identifier removed, ID now string.
- CHANGED: idShort of Referable now optional + Constraints added with respect to id and idShort
- REMOVED: AssetInformation/billOfMaterial removed
- REMOVED: Asset removed
- REMOVED: Views removed, because no longer supported
- NEW: Event and BasicEvent updated and renamed to EventElement and BasicEventElement
- NEW: Checksum introduced for Referables
- REMOVED: security attribute removed from Asset Administration Shell; access control remains part of the specification
- ENHANCED: DataTypeIEC61360 extended with values for IRI, IRDI, BLOB, FILE + corresponding new constraints added
- ENHANCED: Removed and split into DataTypeDefXsd and DataTypeDefRdf; some types are excluded and not supported

- CHANGED: Mapping rules for different serializations + schemata + example in different serializations extracted and no longer part of this specification
- EDITORIAL: Text updated, no kind column any longer in class tables, instead notation of ModelReference<{Referable}>. New table for Primitives/Data Types
- EDITORIAL: New clause "Introduction"
- EDITORIAL: New clause "Matching strategies for semantic identifiers"
- NEW: Environment
- NEW: supplemental Semantic IDs
- NEW: Qualifier/kind
- CHANGED: Renaming of IdentifierKeyValuePair used in AssetInformation to SpecificAssetId

Bugfixes:

- bugfix annotation AnnotatedRelationship is of type aggr and not ref* (diagram was correct)
- bugfix specification of ValueList and ValueReferencePairType, no data types, normal classes
- bugfix table specifications w.r.t. kind of attribute (from aggr to attr – column kind was removed, see above)
- bugfix data type specification LangStringSet (no diagram and table any longer)
- bugfix enumeration ReferableElements, no ConceptDictionary any longer + new elements like new submodel elements SubmodelElementList added
- Entity/globalAssetId diagram (table was correct): Type change from reference of Reference to Reference (from Reference* to Reference)

Table 28. Changes w/o Security

nc	V3.0RC02 Change w.r.t. V3.0RC01	Comment
	AdministrativeInformation	Bugfix: Stereotype "DataType" added
	AnnotatedRelationship/annotation	Type changed from ModelReference<DataElement> to DataElement
x	Asset	Removed, asset referenced via globalAssetId only

nc	V3.0RC02 Change w.r.t. V3.0RC01	Comment
x	AssetAdministrationShell/security	Removed Note: Security is still part of the Asset Administration Shell, but the Asset Administration Shell and its elements are referenced from Security
	AssetAdministrationShell/view	Removed, views not longer supported
x	AssetInformation/billOfMaterial	Removed
x	AssetInformation/defaultThumbnail	Type changed from File to Resource
x	AssetInformation/specificAssetId	Type changed from IdentifierKeyValuePair to SpecificAssetId
x	BasicEvent	Renamed to BasicEventElement
x	Constraint	Abstract class removed. Formula now used in Security part only
(x)	DataTypeDef	Split into DataTypeDefXsd and DataTypeDefRdf. Some types excluded and not supported (see notes in corresponding clause) Before: just string allowing all anySimpleTypes of xsd and langString of rdf
	Entity/globalAssetId	Bugfix: Type change from reference of Reference to Reference (from Reference* to Reference)
x	Event	Renamed to EventElement
x	Extension/refersTo	Type changed from Reference to ModelReference<Referable>
x	File/mimeType	Renamed to contentType + Type name changed from MimeType to ContentType
x	Formula	Now abstract class now used in Security part only
x	Formula/dependsOn	Removed since formula language not yet defined

nc	V3.0RC02 Change w.r.t. V3.0RC01	Comment
x	Identifiable/identification	Removed Substituted by Identifiable/id
(x)	IdentifiableElements	Renamed to AasIdentifiables
x	Identifier	Type changed Before struct class with two attributes: id and idType; now string data type only
x	IdentifierKeyValuePair	Renamed to SpecificAssetId and change of attribute "key" to "name"
	IdentifierType	Enumeration removed because no idType any longer
x	Key/idType	removed
(x)	KeyElements	Renamed to KeyTypes Note: the elements remain, except for new SubmodelElementList and renamed submodel elements Event and BasicEvent to EventElement and BasicEventElement
	KeyType	Enumeration removed because no Key/idType any longer
	LocalKeyType	Enumeration removed because no Key/idType any longer
x	MimeType	Type name changed to ContentType
	Property/valueType	Type changed from DataTypeDef to DataTypeDefXsd
x	Qualifiable/qualifier	Type changed from Constraint to Qualifier
	Qualifier	Does not inherit from abstract class "Constraint" any longer
	Qualifier/valueType	Type changed from DataTypeDef to DataTypeDefXsd
	Range/valueType	Type changed from DataTypeDef to DataTypeDefXsd
	Referable/idShort	Now optional, was mandatory

nc	V3.0RC02 Change w.r.t. V3.0RC01	Comment
x	ReferableElements	Substituted with enumeration AasSubmodelElements and AasIdentifiables
x	ReferableElements/AccessPermissionRule	Removed from enumeration, AccessPermissionRule is no longer referable Not part of new AasReferableNonIdentifiables
x	ReferableElement/BasicEvent	Renamed to BasicEventElement Now part of AasSubmodelElements
(x)	ReferablesElements/ConceptDictionary	Bugfix: ConceptDictionary removed from enumeration since ConceptDictionary no longer part of specification Not part of new KeyTypes
x	ReferableElements/Event	Renamed to EventElement Now part of AasSubmodelElements
	RelationshipElement/first	Type changes from model reference Referable to Reference (global or model reference)
	RelationshipElement/second	Type changes from model reference Referable to Reference (global or model reference)
	ValueDataType	Before as specified via DataTypeDef, now any xsd atomic type as specified via DataTypeDefXsd + Prefix xs: added to every value in list
x	ValueList/valueReferencePairType	Bugfix: renamed to ValueList/valueReferencePairs
x	View	removed

Table 29. New Elements in Metamodel w/o Security

nc	V3.0RC02 vs. V2.0RC01 New Elements	Comment
	AasSubmodelElements	New enumeration used for References Before ReferableElements
	AasIdentifiables	New enumeration used for References, includes abstract Identifiable Before: Identifiables

nc	V3.0RC02 vs. V2.0RC01 New Elements	Comment
	AasReferableNonIdentifiables	New enumeration used for References
	AasReferables	New enumeration used for References, includes abstract Referable
	BasicEventElement	Former name: BasicEvent
	BasicEventElement/direction	
	BasicEventElement/lastUpdate	
	BasicEventElement/messageBroker	
	BasicEventElement/messageTopic	
	BasicEventElement/minInterval	
	BasicEventElement/maxInterval	
	BasicEventElement/observed	Former name: BasicEvent/observed
	BasicEventElement/state	
	ContentType	Former name: MimeType
	DataTypeDefRdf	Enumeration for types of Rdf + prefix rdf: added to every value in enumeration
	DataTypeDefXsd	Enumeration consisting of enumerations decimalBuildInTypes, durationBuildInTypes, PrimitiveTypes that correspond to anySimpleTypes of xsd. + prefix xs: added to every value in enumeration
	dateTimeStamp	New data type for metamodel as used in EventPayload
	decimalBuildInTypes	Enumeration for DataTypeDef
	Direction	New enumeration for BasicEventElement
	durationBuildInTypes	Enumeration for DataTypeDef
	Environment	New class for entry point for Asset Administration Shells, submodels and concept descriptions
	EventElement	Former name: Event
	EventPayload	New class for event payload
	EventPayload/observableReference	

nc	V3.0RC02 vs. V2.0RC01 New Elements	Comment
	EventPayload/observableSemanticId	
	EventPayload/payload	
	EventPayload/source	
	EventPayload/sourceSemanticId	
	EventPayload/subjectId	
	EventPayload/timestamp	
	EventPayload/topic	
	File/contentType	Former name: mimeType
	FragmentKeys	New enumeration used for References
	GenericFragmentKeys	New enumeration used for References
	GenericGloballyIdentifiers	New enumeration used for References
	GloballyIdentifiables	New enumeration used for References
	HasSemantics/supplementalSemanticId	New attribute
	Identifiable/id	Substitute for Identifiable/identification
	KeyTypes	Before: KeyElements New submodel element SubmodelElementList added, renamed submodel elements Event and BasicEvent to EventElement and BasicEventElement
	ModelReference	New class inheriting from Reference
x	Reference/type	New mandatory attribute of Reference
	Reference/referredSemanticId	New optional attribute of Reference
	PrimitiveTypes	Enumeration for DataTypeDefXsd
	Qualifier/kind	New attribute for Qualifier
	QualifierKind	New enumeration for Qualifier/kind
	Referable/checksum	
	SpecificAssetId	Before: IdentifierKeyValuePair, was renamed
	SpecificAssetId/name	Before: IdentifierKeyValuePair/key, was renamed

nc	V3.0RC02 vs. V2.0RC01 New Elements	Comment
	SpecificAssetId/value	Before: IdentifierKeyValuePair/value
	SpecificAssetId/externalSubjectId	Before: IdentifierKeyValuePair/externalSubjectId
	StateOfEvent	New enumeration for BasicEventElement
	SubmodelElementElements	Enumeration for submodel elements (split of ReferableElements into SubmodelElementElements and IdentifiableElements)
	SubmodelElementList	Before SubmodelElementCollection was used for lists and structs
	SubmodelElementList/orderRelevant	Similar to SubmodelElementCollection/ordered
	SubmodelElementList/value	Similar to SubmodelElementCollection/value but ordered and with all elements having the same semanticId
	SubmodelElementList/semanticIdListElement	Attribute for new class SubmodelElementList
	SubmodelElementList/typeValueListElement	Attribute for new class SubmodelElementList
	SubmodelElementList/valueTypeListElement	Attribute for new class SubmodelElementList

Table 30. New, Changed or Removed Constraints w/o Security

Nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
[7]	AASd-003	Update	idShort is case-sensitive and not case-insensitive Constraint AASd-003: *idShort* of *Referables* shall be matched case-sensitive.
	AASd-005	Reformulated	Constraint AASd-005: If AdministrativeInformation/version is not specified than also AdministrativeInformation/revision shall be unspecified. This means, a revision requires a version. if there is no version there is no revision neither. Revision is optional.
	AASd-008	Removed	Constraint AASd-008: The submodel element value of an operation variable shall be of kind=Template.

Nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-023	Removed	No Asset any longer that can be referenced as alternative to global reference Constraint AASd-023: AssetInformation/globalAssetId either is a reference to an Asset object or a global reference.
	AASd-026	Removed	SubmodelElementCollection was split into SubmodelElementList and SubmodelElementRecord. No attribute allowDuplicates any longer. Constraint AASd-026: If allowDuplicates==false then it is not allowed that the collection contains several elements with the same semantics (i.e. the same semanticId).
x	AASd-027	New	Constraint AASd-027: *idShort* of *Referables* shall have a maximum length of 128 characters.
	AASd-050	Update	Version information in data specification ID updated to /3/0/RC02. hasDataSpecification corrected to HasDataSpecification Constraint AASd-050: If the *DataSpecificationContent* DataSpecificationIEC61360 is used for an element then the value of H*asDataSpecification/dataSpecification* shall contain the global reference to the IRI of the corresponding data specification template *https://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/3/0/RC02*.
(x)	AASd-050b	New	Constraint AASd-050b: If the DataSpecificationContent DataSpecificationPhysicalUnit is used for an element then the value of HasDataSpecification/dataSpecification shall contain the global reference to the IRI of the corresponding data specification template https://admin-shell.io/DataSpecificationTemplates/DataSpecificationPhysicalUnit0/3/0/RC02.
	AASd-052a	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-052a: If the semanticId of a Property references a ConceptDescription then the ConceptDescription/category shall be one of following values: VALUE, PROPERTY.

Nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-052b	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-052b: If the semanticId of a MultiLanguageProperty references a ConceptDescription then the ConceptDescription/category shall be one of following values: PROPERTY.
	AASd-053	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-053: If the semanticId of a Range submodel element references a ConceptDescription then the ConceptDescription/category shall be one of following values: PROPERTY.
	AASd-054	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-054: If the semanticId of a ReferenceElement submodel element references a ConceptDescription then the ConceptDescription/category shall be one of following values: REFERENCE.
	AASd-055	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-055: If the semanticId of a RelationshipElement or an AnnotatedRelationshipElement submodel element references a ConceptDescription then the ConceptDescription/category shall be one of following values: RELATIONSHIP.
	AASd-056	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-056: If the semanticId of a Entity submodel element references a ConceptDescription then the ConceptDescription/category shall be one of following values: ENTITY. The ConceptDescription describes the elements assigned to the entity via Entity/statement.
	AASd-057	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-057: The semanticId of a File or Blob submodel element shall only reference a ConceptDescription with the category DOCUMENT.

Nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-058	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-058: The semanticId of a Capability submodel element shall only reference a ConceptDescription with the category CAPABILITY.
	AASd-059	Removed	removed, still recommended; would be renamed to AASc if still needed SubmodelElementCollection was split into SubmodelElementList and SubmodelElementCollection. AASd-092 and AASd-093 contain it. Constraint AASd-059: If the semanticId of a SubmodelElementCollection references a ConceptDescription then the category of the ConceptDescription shall be ~~**COLLECTION or**~~ ENTITY.
	AASd-060	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-060: If the semanticId of a Operation submodel element references a ConceptDescription then the category of the ConceptDescription shall be one of the following values: FUNCTION.
	AASd-061	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-061: If the semanticId of a Event submodel element references a ConceptDescription then the category of the ConceptDescription shall be one of the following values: EVENT.
	AASd-062	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-062: If the *semanticId* of a *Property* references a *ConceptDescription* then the *ConceptDescription/category* shall be one of following values: APPLICATION_CLASS.
	AASd-063	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-063: If the semanticId of a Qualifier references a ConceptDescription then the ConceptDescription/category shall be one of following values: QUALIFIER.

Nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-064	Removed	Removed because there are not VIEWs any longer Constraint AASd-064: If the *semanticId* of a *View* references a *ConceptDescription* then the category of the *ConceptDescription* shall be *VIEW*.
	AASd-065	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-065: If the semanticId of a Property or MultiLanguageProperty references a ConceptDescription with the category VALUE then the value of the property is identical to DataSpecificationIEC61360/value and the valueId of the property is identical to DataSpecificationIEC61360/valueId.
	AASd-066	Removed	removed, still recommended; would be renamed to AASc if still needed Update because of renaming of ValueReferencePairType into ValueReferencePair Constraint AASd-066: If the semanticId of a Property or MultiLanguageProperty references a ConceptDescription with the category PROPERTY and DataSpecificationIEC61360/valueList is defined the value and valueId of the property is identical to one of the value reference pair types references in the value list, i.e. ValueReferencePair/value or ValueReferencePair/valueId, resp.
	AASd-067	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-067: If the semanticId of a MultiLanguageProperty references a ConceptDescription then DataSpecificationIEC61360/dataType shall be STRING_TRANSLATABLE.
	AASd-068	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-068: If the semanticId of a Range submodel element references a ConceptDescription then DataSpecificationIEC61360/dataType shall be a numerical one, i.e. REAL_* or RATIONAL_*.

Nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-069	Removed	removed, still recommended; would be renamed to AASc if still needed Constraint AASd-069: If the semanticId of a Range references a ConceptDescription then DataSpecificationIEC61360/levelType shall be identical to the set \{Min, Max}.
(x)	AASd-070	Renamed	Now AASc-004.
(x)	AASd-071	Renamed	Now AASc-005
(x)	AASd-072	Renamed	Now AASc-006.
(x)	AASd-073	Renamed	Now AASc-007
(x)	AASd-074	Renamed	Now AASc-008
	AASd-075	Removed	Content now documented as separate constraints Constraint AASd-075: For all ConceptDescriptions using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) values for the attributes not being marked as mandatory or optional in tables depending on its category are ignored and handled as undefined.
	AASd-076	Removed	Substituted by AASc-002. Simplified, no reference to concept description
	AASd-080	Removed	No *Key/type* GlobalReference any longer Constraint AASd-080: In case *Key/type == GlobalReference idType* shall not be any LocalKeyType (*IdShort, FragmentId).*
	AASd-081	Removed	No *Key/idType* any longer Constraint AASd-081: In case Key/type==AssetAdministrationShell Key/idType shall not be any LocalKeyType (IdShort, FragmentId).
	AASd-090	Update	Exception: File and Blob data elements removed. Reformulated. Constraint AASd-090: For data elements category (inherited by Referable) shall be one of the following values: CONSTANT, PARAMETER or VARIABLE. Default: VARIABLE

Nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-092	Removed	removed, still recommended; would be renamed to AASc and updated if still needed SubmodelElementCollection was split into SubmodelElementList and SubmodelElementCollection (here: SubmodelElementCollection) Constraint AASd-092: If the semanticId of a SubmodelElementCollection with SubmodelElementCollection/allowDuplicates == false references a ConceptDescription then the ConceptDescription/category shall be ENTITY.
	AASd-093	Removed	removed, still recommended; would be renamed to AASc and updated if still needed SubmodelElementCollection was split into SubmodelElementList and SubmodelElementStruct (here: SubmodelElementList) Constraint AASd-093: If the semanticId of a SubmodelElementCollection with SubmodelElementCollection/allowDuplicates == true references a ConceptDescription then the ConceptDescription/category shall be COLLECTION.
	AASd-107	New	Constraint AASd-107: If a first level child element in a SubmodelElementList has a semanticId it shall be identical to SubmodelElementList/semanticIdListElement.
	AASd-108	New	Constraint AASd-108: All first level child elements in a SubmodelElementList shall have the same submodel element type as specified in SubmodelElementList/typeValueListElement.
	AASd-109	New	Constraint AASd-109: If SubmodelElementList/typeValueListElement equal to Property or Range SubmodelElementList/valueTypeListElement shall be set and all first level child elements in the SubmodelElementList shall have the the value type as specified in SubmodelElementList/valueTypeListElement.
	AASd-114	New	Constraint AASd-114: If two first level child elements in a SubmodelElementList have a semanticId then they shall be identical.

Nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-115	New	Constraint AASd-115: If a first level child element in a SubmodelElementList does not specify a semanticId then the value is assumed to be identical to SubmodelElementList/semanticIdListElement.
	AASd-116	New	Constraint AASd-116: "globalAssetId" (case-insensitive) is a reserved key. If used as value for SpecificAssetId/name IdentifierKeyValuePair/value shall be identical to AssetInformation/globalAssetId.
	AASd-117	New	Needed because Referable/idShort now optional Constraint AASd-117: idShort of non-identifiable Referables not equal to SubmodelElementList shall be specified (i.e. idShort is mandatory for all Referables except for SubmodelElementLists and all Identifiables).
	AASd-118	New	Constraint AASd-118: If there is a supplemental semantic ID (HasSemantics/supplementalSemanticId) defined then there shall be also a main semantic ID (HasSemantics/semanticId).
	AASd-119	New	New Qualifier/kind attribute Constraint AASd-119: If any Qualifier/kind value of a Qualifiable/qualifier is equal to TemplateQualifier and the qualified element inherits from "hasKind" then the qualified element shall be of kind Template (HasKind/kind = "Template").
	AASd-120	New	For new submodel element SubmodelElementList Constraint AASD-120: idShort of submodel elements within a SubmodelElementList shall not be specified.
	AASd-121	New	Constraint AASd-121: For References the type of the first key of Reference/keys shall be one of GloballyIdentifiables.
	AASd-122	New	Constraint AASd-122: For global references, i.e. References with Reference/type = GlobalReference, the type of the first key of Reference/keys shall be one of GenericGloballyIdentifiables.
	AASd-123	New	Constraint AASd-123: For model references, i.e. References with Reference/type = ModelReference, the type of the first key of Reference/keys shall be one of AasIdentifiables.

Nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-124	New	Constraint AASd-124: For global references, i.e. References with Reference/type = GlobalReference, the last key of Reference/keys shall be either one of GenericGloballyIdentifiables or one of GenericFragmentKeys.
	AASd-125	New	Constraint AASd-125: For model references, i.e. References with Reference/type = ModelReference, with more than one key in Reference/keys the type of the keys following the first key of Reference/keys shall be one of FragmentKeys.
	AASd-126	New	Constraint AASd-126: For model references, i.e. References with Reference/type = ModelReference, with more than one key in Reference/keys the type of the last Key in the reference key chain may be one of GenericFragmentKeys or no key at all shall have a value out of GenericFragmentKey.
	AASd-127	New	Constraint AASd-127: For model references, i.e. References with Reference/type = ModelReference, with more than one key in Reference/keys a key with type FragmentReference shall be preceeded by a key with type File or Blob. All other AAS fragments, i.e. type values out of AasSubmodelElements, do not support fragments.
	AAS-128	New	Constraint AASd-128: For model references, i.e. References with Reference/type = ModelReference, the Key/value of a Key preceeded by a Key with Key/type=SubmodelElementList is an integer number denoting the position in the array of the submodel element list.

H.12. Metamodel Changes V3.0RC02 vs. V3.0RC01 – Data Specification IEC61360

Table 31. Changes w.r.t. Data Specification IEC61360

nc	V3.0RC02 Change w.r.t. V3.0RC01	Comment
	DataSpecification	Stereotype <<Template>> added + does not inherit from Identifiable any longer because Data Specification are handled differently Some attributes are added to DataSpecification as new attributes like id, administration and description
	DataSpecification/category	Removed, was inherited before by Identifiable
	DataSpecification/displayName	Removed, was inherited before by Identifiable
	DataSpecification/idShort	Removed, was inherited before by Identifiable
	DataSpecificationIEC61360/unitId	Type changes from Reference to GlobalReference
x	DataSpecificationIEC61360/value	Type changed from ValueDataType to string
	DataSpecificationIEC61360/valueId	Removed, the valueId is identical to the ID of the concept description
	DataSpecificationContent	Stereotype <<Template>> added
x	DataTypeIEC61360	Some new values were added: BLOB, FILE, HTML, IRDI; URL renamed to IRI See separate entries for individual changes
x	DataTypeIEC61360/URL	Renamed to IRI
	ValueList/valueReferencePairs	Bugfix, was ValueList/valueReferencePairTypes before
x	ValueReferencePair/value	Type changed from ValueDataType to string

Table 32. New Elements in Metamodel DataSpecification IEC61360

nc	V3.0RC02	Comment
x	ValueReferencePair/valueId	Type changed from Reference to GlobalReference
	DataSpecification/administration	Was inherited before by Identifiable
	DataSpecification/id	Was inherited before by Identifiable
	DataSpecification/desciption	Was inherited before by Identifiable
	DataTypeIEC61360/BLOB	New value

nc	V3.0RC02	Comment
	DataTypeIEC61360/FILE	New value
	DataTypeIEC61360/HTML	New value
	DataTypeIEC61360/IRDI	New value
	DataTypeIEC61360/IRI	Converted Iri to CamelCase and renamed to Iri from URL

Table 33. New, Changed or Removed Constraints Data Specification IEC61360

nc	V3.0RC02	New, Update, Removed, Reformulated	Comment
	AASc-002	New	Updated version of AASd-076, renamed to AASC-002 because applicable to data specification IEC61360 Constraint AASc-002: Data¬Specification¬IEC61360-/preferredName shall be provided at least in English
(x)	AASc-003	New	Constraint AASc-003: For a ConceptDescription with category VALUE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) DataSpecificationIEC61360/value shall be set.
(x)	AASc-004	New	Constraint AASc-004: For a ConceptDescription with category PROPERTY or VALUE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType is mandatory and shall be defined.
(x)	AASc-005	New	Constraint AASc-005: For a ConceptDescription with category REFERENCE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType is STRING by default.

nc	V3.0RC02	New, Update, Removed, Reformulated	Comment
(x)	AASc-006	New	Constraint AASc-006: For a ConceptDescription with category DOCUMENT using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType shall be one of the following values: STRING or URL.
(x)	AASc-007	New	Constraint AASc-007: For a ConceptDescription with category QUALIFIER_TYPE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType is mandatory and shall be defined.
(x)	AASc-008	New	Constraint AASc-008: For a ConceptDescriptions except for a ConceptDescription of category VALUE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/definition is mandatory and shall be defined at least in English.
(x)	AASc-009	New	Constraint AASc-009: If DataSpecificationIEC61360/dataType one of: INTEGER_MEASURE, REAL_MEASURE, RATIONAL_MEASURE, INTEGER_CURRENCY, REAL_CURRENCY, then DataSpecificationIEC61360/unit or DataSpecificationIEC61360/unitId shall be defined.
(x)	AASc-010	New	Constraint AASc-010: If DataSpecificationIEC61360/value is not empty then DataSpecificationIEC61360/valueList shall be empty and vice versa

H.13. Metamodel Changes V3.0RC02 vs. V3.0RC01 – Security Part

Changes:

- Removed, because deprecated: policy decision point, policy enforcement point, and policy information points are not part of information model but of server infrastructure hosting the Asset Administration Shells

- Removed: Certificate Handling not part of information model but of server infrastructure hosting the Asset Administration Shells

Table 34. Changes w.r.t. Security

nc	V3.0RC02 Change w.r.t. V3.0RC01	Comment
x	AccessControlPolicyPoints/policyAdministrationoint	Type changed from PolicyAdministrationPoint to AccessControl
x	AccessControlPolicyPoints/policyDecisionPoint	Removed
x	AccessControlPolicyPoints/policyEnforcementPoint	Removed
x	AccessControlPolicyPoints/policyInformationPoint	Removed
x	AccessPermissionRule	No longer inherits from referable No longer inherits from qualifiable
x	BlobCertificate	Removed
x	Certificate	Removed
x	Formula	Now abstract class, only used in security part (no longer used in Qualifiables)
x	Formula/dependsOn	Removed attribute
x	PolicyAdministrationPoint	Removed
x	policyDecisionPoint	Removed
x	policyEnforcementPoint	Removed
x	policyInformationPoints	Removed
x	Security/certificate	Removed
x	Security/requiredCertificateExtension	Removed

Table 35. New Elements in Metamodel Security

nc	V3.0RC02 vs. V3.0RC01	Comment
	AccessPermissionRule/constraint	Substitute for inherited attributes from Qualifiable

Table 36. New, Changed or Removed Constraints Security

nc	V3.0RC02 vs. V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASs-009	Removed	Removed since class PolicyAdministrationPoint was removed Constraint AASs-009: Either there is an external policy administration point endpoint defined (PolicyAdministrationPoint/externalPolicyDecisionPoints=true) or the AAS has its own access control
	AASs-015	Updated	Constraint AASs-015: Every data element in SubjectAttributes/subjectAttributes shall be part of the submodel that is referenced within the "selectableSubjectAttributes" attribute of "AccessControl".

H.14. Changes V3.0RC01 vs. V2.0.1

H.15. Metamodel Changes V3.0RC01 w/o Security Part

Major changes:

- idShort of Submodels etc. no longer need to be unique in the context of an Asset Administration Shell
- Constraints implicitly contained in text were formalized and numbered
- Revised concept on handling of Asset and assetIdentificationModel (assetInformation)
- ConceptDictionaries not supported any longer
- semanticId no longer mandatory for SubmodelElement
- More than one bill of material for assetInformation in Asset Administration Shell
- Local attribute in References removed
- Parent attribute in Referables removed
- Abstract class HasExtension introduced
- AASX file exchange format:no splitting of an Asset Administration Shell allowed any longer (i.e. relationship type aas-spec-split removed)

Table 37. Changes w.r.t. V2.0 w/o Security

nc	V3.0RC01 Change w.r.t. V2.0.1	Comment
	anySimpleTypeDef	Type removed, was not used in any class definition any longer, was mentioned in text only
x	AssetAdministrationShell/asset	Removed, substituted by AssetAdministrationShell/assetInformation (no reference any longer, instead now aggregation)
x	Asset/assetKind	Attribute "assetKind" moved to AssetAdministrationShell/AssetInformation
x	Asset/assetIdentificationModel	Attribute "assetIdentificationModel" removed, substituted by AssetInformation /IdentifierKeyValuePairs
x	Asset/billOfMaterial	Attribute "billOfMaterial" moved to AssetAdministrationShell/AssetInformation
x	AssetAdministrationShell/conceptDictionaries	Removed
	ConceptDescription/isCaseOf	Text changed, no global reference requested, just reference
x	ConceptDictionary	Removed
x	Entity/asset	Removed, substituted by Entity/globalAssetId and Entity/specificAssetId
x	Key/local	Local attribute removed
x	Referable/parent	Parent attribute removed

Table 38. New Elements in Metamodel V3.0RC01 w/o Security

nc	V3.0RC01 vs. V2.0.1	Comment
x	AssetAdministrationShell/assetInformation	Substitute for AssetAdministrationShell/asset (no reference any longer, instead aggregation)
	AssetInformation	with attributes/functionality from former class Asset because not specific to Asset but AAS
	AssetInformation/thumbnail	Optional Attribute of new class AssetInformation that was not available in Asset class before
x	Entity/globalAssetId	Substitute for Entity/asset (together with Entity/specificAssetId)

nc	V3.0RC01 vs. V2.0.1	Comment
x	Entity/specificAssetId	Substitute for Entity/asset (together with Entity/globalAssetId)
	Extension	New class, part of new abstract class HasExtensions
	HasExtensions	New abstract class, inherited by Referable
	IdentifierKeyValuePair	New class for AssetInformation/specificAssetId
	Referable/displayName	New optional attribute for all referables

Table 39. New, Changed or Removed Constraints w/o Security

nc	V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-001	Removed	Constraint AASd-001: In case of a referable element not being an identifiable element this id is mandatory and used for referring to the element in its name space. For namespace part see AASd-022
x	AASd-002	Update	reformulated, formula added *idShort* of *Referable*s shall only feature letters, digits, underscore (""*); starting mandatory with a letter. I.e. [a-zA-Z][a-zA-Z0-9*]+
	AASd-010	Reformulated	Constraint AASd-010: The property has the category "CONSTANT". Reformulated to Constraint AASd-010: The property referenced in Permission/permission shall have the category "CONSTANT".
	AASd-011	Reformulated	Constraint AASd-011: The property referenced in Permission/permission shall be part of the submodel that is referenced within the "selectablePermissions" attribute of "AccessControl".
	AASd-012	Reformulated	Constraint AASd-012: If both the MultiLanguageProperty/value and the MultiLanguageProperty/valueId are present then for each string in a specific language the meaning must be the same as specified in MultiLanguageProperty/valueId

nc	V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-014	Reformulated	Entity was changed Constraint AASd-014: Either the attribute globalAssetId or specificAssetId of an *Entity* must be set if *Entity/entityType* is set to "*SelfManagedEntity*". They are not existing otherwise.
(x)	AASd-020	New	Constraint AASd-020: The value of Property/*value* shall be consistent to the data type as defined in Property/*valueType*.
(x)	AASd-021	New	Constraint AASd-021: Every qualifiable can only have one qualifier with the same *Qualifier/type.*
(x)	AASd-022	New	Part from AASd-001 after split Constraint AASd-022: idShort of non-identifiable referables shall be unique in its namespace.
(x)	AASd-026	New	Constraint AASd-026: If allowDuplicates==false then it is not allowed that the collection contains several elements with the same semantics (i.e. the same semanticId).
(x)	AASd-050	New	Constraint AASd-050: If the DataSpecificationContent DataSpecificationIEC61360 is used for an element then the value of hasDataSpecification/dataSpecification shall contain the global reference to the IRI of the corresponding data specification template http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0.
(x)	AASd-051	New	Constraint AASd-051: A ConceptDescription shall have one of the following categories: VALUE, PROPERTY, REFERENCE, DOCUMENT, CAPABILITY, RELATIONSHIP, COLLECTION, FUNCTION, EVENT, ENTITY, APPLICATION_CLASS, QUALIFIER, VIEW. Default: PROPERTY.
(x)	AASd-052a	New	Constraint AASd-052a: If the semanticId of a Property references a ConceptDescription then the ConceptDescription/category shall be one of following values: VALUE, PROPERTY.
(x)	AASd-052b	New	Constraint AASd-052b: If the semanticId of a MultiLanguageProperty references a ConceptDescription then the ConceptDescription/category shall be one of following values: PROPERTY.

nc	V3.0RC01	New, Update, Removed, Reformulated	Comment
(x)	AASd-053	New	Constraint AASd-053: If the semanticId of a Range submodel element references a ConceptDescription then the ConceptDescription/category shall be one of following values: PROPERTY.
(x)	AASd-054	New	Constraint AASd-054: If the semanticId of a ReferenceElement submodel element references a ConceptDescription then the ConceptDescription/category shall be one of following values: REFERENCE.
(x)	AASd-055	New	Constraint AASd-055: If the semanticId of a RelationshipElement or an AnnotatedRelationshipElement submodel element references a ConceptDescription then the ConceptDescription/category shall be one of following values: RELATIONSHIP.
(x)	AASd-056	New	Constraint AASd-056: If the semanticId of a Entity submodel element references a ConceptDescription then the ConceptDescription/category shall be one of following values: ENTITY. The ConceptDescription describes the elements assigned to the entity via Entity/statement.
(x)	AASd-057	New	Constraint AASd-057: The semanticId of a File or Blob submodel element shall only reference a ConceptDescription with the category DOCUMENT.
(x)	AASd-058	New	Constraint AASd-058: The semanticId of a Capability submodel element shall only reference a ConceptDescription with the category CAPABILITY.
(x)	AASd-059	New	Constraint AASd-059: The semanticId of a SubmodelElementCollection submodel element shall only reference a ConceptDescription with the category COLLECTION or ENTITY.
(x)	AASd-060	New	Constraint AASd-060: If the semanticId of a Operation submodel element references a ConceptDescription then the category of the ConceptDescription shall be one of the following values: FUNCTION.
(x)	AASd-061	New	Constraint AASd-061: If the semanticId of a Event submodel element references a ConceptDescription then the category of the ConceptDescription shall be one of the following values: EVENT.

nc	V3.0RC01	New, Update, Removed, Reformulated	Comment
(x)	AASd-062	New	Constraint AASd-062: If the *semanticId* of a *Property* references a *ConceptDescription* then the *ConceptDescription/category* shall be one of following values: APPLICATION_CLASS.
(x)	AASd-063	New	Constraint AASd-063: If the semanticId of a Qualifier references a ConceptDescription then the ConceptDescription/category shall be one of following values: QUALIFIER.
(x)	AASd-064	New	Constraint AASd-064: If the *semanticId* of a *View* references a *ConceptDescription* then the category of the *ConceptDescription* shall be *VIEW*.
(x)	AASd-065	New	Constraint AASd-065: If the semanticId of a Property or MultiLanguageProperty references a ConceptDescription with the category VALUE then the value of the property is identical to DataSpecificationIEC61360/value and the valueId of the property is identical to DataSpecificationIEC61360/valueId.
(x)	AASd-066	New	Constraint AASd-066: If the semanticId of a Property or MultiLanguageProperty references a ConceptDescription with the category PROPERTY and DataSpecificationIEC61360/valueList is defined the value and valueId of the property is identical to one of the value reference pair types references in the value list, i.e. ValueReferencePair/value or ValueReferencePair/valueId, resp.
(x)	AASd-067	New	Constraint AASd-067: If the semanticId of a MultiLanguageProperty references a ConceptDescription then DataSpecificationIEC61360/dataType shall be STRING_TRANSLATABLE.
(x)	AASd-068	New	Constraint AASd-068: If the semanticId of a Range submodel element references a ConceptDescription then DataSpecificationIEC61360/dataType shall be a numerical one, i.e. REAL_* or RATIONAL_*.
(x)	AASd-069	New	Constraint AASd-069: If the semanticId of a Range references a ConceptDescription then DataSpecificationIEC61360/levelType shall be identical to the set \{Min, Max}.

nc	V3.0RC01	New, Update, Removed, Reformulated	Comment
(x)	AASd-070	New	Constraint AASd-070: For a ConceptDescription with category PROPERTY or VALUE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType is mandatory and shall be defined.
(x)	AASd-071	New	Constraint AASd-071: For a ConceptDescription with category REFERENCE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType is STRING by default.
(x)	AASd-072	New	Constraint AASd-072: For a ConceptDescription with category DOCUMENT using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType shall be one of the following values: STRING or URL.
(x)	AASd-073	New	Constraint AASd-073: For a ConceptDescription with category QUALIFIER using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/dataType is mandatory and shall be defined.
(x)	AASd-074	New	Constraint AASd-074: For all ConceptDescriptions except for ConceptDescriptions of category VALUE using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) - DataSpecificationIEC61360/definition is mandatory and shall be defined at least in English.
(x)	AASd-075	New	Constraint AASd-075: For all ConceptDescriptions using data specification template IEC61360 (http://admin-shell.io/DataSpecificationTemplates/DataSpecificationIEC61360/2/0) values for the attributes not being marked as mandatory or optional in tables Table 6, Table 7, Table 8 and Table 9.depending on its category are ignored and handled as undefined.

nc	V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-077	New	Constraint AASd-077: The name of an extension within HasExtensions needs to be unique.
(x)	AASd-080	New	Constraint AASd-080: In case *Key/type == GlobalReference idType* shall not be any LocalKeyType (*IdShort, FragmentId).*
	AASd-081	New	Constraint AASd-081: In case *Key/type==AssetAdministrationShell Key/idType* shall not be any *LocalKeyType* (*IdShort, FragmentId).*
(x)	AASd-092	New	Constraint AASd-092: If the semanticId of a SubmodelElementCollection with SubmodelElementCollection/allowDuplicates == false references a ConceptDescription then the ConceptDescription/category shall be ENTITY.
(x)	AASd-093	New	Constraint AASd-093: If the semanticId of a SubmodelElementCollection with SubmodelElementCollection/allowDuplicates == true references a ConceptDescription then the ConceptDescription/category shall be COLLECTION.
	AASd-100	New	Constraint AASd-100: An attribute with data type "string" is not allowed to be empty.

H.16. Metamodel Changes V3.0RC01 – Security Part

Major changes:

- Constraints for security part renamed from pattern AASd- to AASs-.
- Only bugfixes

Table 40. New, Changed or Removed Constraints Security

nc	V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASd-010	Removed	Renamed to AASs-010 (see NEW)

nc	V3.0RC01	New, Update, Removed, Reformulated	Comment
	AASs-010	NEW	Reformulation of AASd-010 Constraint AASs-010: The property referenced in Permission/permission shall have the category "CONSTANT".
	AASd-011	Removed	Renamed to AASs-011 (see NEW)
	AASs-011	NEW	Reformulation of AASd-011 Constraint AASs-011: The property referenced in Permission/permission shall be part of the submodel that is referenced within the "selectablePermissions" attribute of "AccessControl".
	AASd-015	Removed	Renamed to AASs-015 (see NEW)
	AASs-015	NEW	Constraint AASd-015: The data element SubjectAttributes/subjectAttribute shall be part of the submodel that is referenced within the "selectableSubjectAttributes" attribute of "AccessControl".

H.17. Changes V2.0.1 vs. V2.0

H.18. Metamodel Changes V2.0.1 w/o Security Part

Major changes:

- Only bugfixes

Table 41. Changes w.r.t. V2.0.1 w/o Security

nc	V2.0.1 Change w.r.t. V2.0	Comment
	DataTypeIEC61360/INTEGER_COUNT	Bugfix, was missing
	DataTypeIEC61360/INTEGER_MEASURE	Bugfix, was missing
	DataTypeIEC61360/INTEGER_CURRENCY	Bugfix, was missing

nc	V2.0.1 Change w.r.t. V2.0	Comment
	hasDataSpecification	Bugfix, is abstract class – was mixed up with DataSpecification class that is not abstract
	DataSpecification	Bugfix, is not abstract
	AnnotatedRelationshipElement/annotation	Bugfix, Annotation ist not a reference to Data Elements

Table 42. New, Changed or Removed Constraints w/o Security

nc	V2.0.1	New, Update, Removed	Comment
	AASd-001	update	idShort now mandatory Constraint AASd-001: an identifiable element this id is mandatory and used for referring to the element in its name space. Constraint AASd-001: In case of a referable element not being an identifiable element this ID is used for referring to the element in its name space.
	AASd-013	Removed	Constraint AASd-013: In case of a range with kind=Instance either the min or the max value or both need to be defined.

H.19. Changes V2.0 vs. V1.0

H.20. Metamodel Changes V2.0 w/o Security Part

Major changes:

- Composite I4.0 Components supported via new Entity submodel element and billOfMaterial
- Event submodel element introduced
- Capability submodel element introduced
- Annotatable relationship submodel element introduced
- MultiLanguageProperty submodel element introduced
- Range submodel element introduced
- Data Specification Template IEC61360 extended for Values, ValueLists and Ranges
- Referencing of fragments within a file etc. now also supported

Table 43. Changes w.r.t. V1.0 w/o Security

nc	V2.0 Change w.r.t. V1.0	Comment
(x) [8]	anySimpleTypeDef	Type now starts with capital letter: AnySimpleTypeDef Type changed from string to values representing xsd-type anySimpleType
	Asset	Does not inherit from HasKind any longer (but attribute kind remains)
	Asset/kind	Now of type "AssetKind" instead of "Kind". Instead of value Type and Instance now value Template and Instance
	AssetAdministrationShell/security	Now optional to support passive AAS of type 1
	Code	Data type removed, no longer used
x	DataSpecificationIEC61360/shortName	Type changed from string to LangStringSet Cardinality changed from mandatory to optional
x	DataSpecificationIEC61360/sourceOfDefinition	Type changed from langString to string
(x) [9]	DataSpecificationIEC61360/dataType	Type changed from string to Enumeration Cardinality changed from mandatory to optional
x	DataSpecificationIEC61360/code	Attribute code removed
	DataSpecificationIEC61360/definition	Cardinality changed from mandatory to optional
	HasDataSpecification	Was abstract before
	HasDataSpecification/hasDataSpecification	Renamed to HasDataSpecification/dataSpecification
x	HasKind/kind	Now of type "ModellingKind" instead of "Kind". Values changed: Type now Template; Instance remains
x	File/value	File name not without but with extension
x	Identifiable/description	Type changed from langString to LangStringSet
x	IdentifierType/URI	URI renamed to IRI

nc	V2.0 Change w.r.t. V1.0	Comment
	Kind	Type Kind removed and substituted by types AssetKind and ModellingKind
x	OperationVariable	No longer inherits from SubmodelElement
	Property/value	Type changed from anySimpleTypeDef to ValueDataType
x	Qualifier/qualifierType	Renamed to Qualifier/type
x	Qualifier/qualifierValue	Renamed to Qualifier/value Type changed from AnySimpleTypeDef to ValueDataType
x	Qualifier/qualifierValueId	Renamed to Qualifier/valueId
x	Referable/idShort	Now mandatory, was optional (but with constraints for defined elements)
x	Reference/key	Cardinality changed from 0..* to 1..*

Table 44. New Elements in Metamodel V1.0 w/o Security

V2.0	Comment
AnnotatedRelationshipElement	New submodel element, inheriting from RelationshipElement
Asset/billOfMaterial	New attribute
AssetKind	New enumeration type
BasicEvent	New submodel element, inherits from Event
Capability	New submodel element
DataSpecificationIEC61360/valueList	For value lists (string)
DataSpecificationIEC61360/value	For coded and explicit values
DataSpecificationIEC61360/valueId	For coded values
DataSpecificationIEC61360/levelType	For Ranges
DataSpecificationPhysicalUnit	New data specification template
DataTypeIEC61360	New enumeration type
Entity	New submodel Element

V2.0	Comment
EntityType	New enumeration type
IdentifierType	Is a subset of KeyType Enumeration
KeyElements/FragmentReference	New value FragmentReference as part of KeyElements Enumeration
LocalKeyType	Is a subset of KeyType Enumeration
LocalKeyType/FragmentId	New value for KeyType Enumeration (via subset LocalKeyType)
LangStringSet	New type, used for example in MultiLanguageProperty
LevelType	New enumeration type
ModellingKind	New enumeration type
MultiLanguageProperty	New submodel element
Qualifier/valueType	New attribute to be consistent with valueType of Property etc.
Range	New submodel element
ReferableElements/BasicEvent	New enumeration value
ReferableElements/Capability	New enumeration value
ReferableElements/Event	New enumeration value
ReferableElements/MultiLanguageProperty	New enumeration value
ReferableElements/Range	New enumeration value
ValueDataType	New type, used for example for Property value
ValueList	New class
ValueReferencePairType	New class

Table 45. New, Changed or Removed Constraints w/o Security

nc	V2.0	New, Update, Removed	Comment
	AASd-007	update	Reformulated Constraint AASd-007: if both, the value and the valueId are present then the value needs to be identical to the value of the referenced coded value in valueId.

nc	V2.0	New, Update, Removed	Comment
	AASd-008	update	Reformulated Constraint AASd-008: The submodel element value of an operation variable shall be of kind=Template.
	AASd-025	removed	Redundant to AASd-015 Constraint AASd-025: The data element shall be part of the submodel that is referenced within the "selectableSubjectAttributes" attribute of "AccessControl".

H.21. Metamodel Changes V2.0 – Security Part

Table 46. Changes Metamodel w.r.t. V1.0 Security

nc	V2.0 Change w.r.t. V1.0	Comment
x	AccessControl/selectableEnvironmentAttributes	Type changed from Submodel to Submodel*
	AccessPermissionRule/permissionsPerObject	Cardinality now consistent for figure and table: 0..*
x	AccessPermissionRule/targetSubjectAttributes	Cardinality changed from 1..* to 1
	Certificate	Was abstract, now not abstract and contains attributes (see in table New)
x	PermissionKind/allow	Now PermissionKind/Allow starts with capital letter for enumeration values
x	PermissionKind/deny	Now PermissionKind/Deny starts with capital letter for enumeration values
x	PermissionKind/not applicable	Now PermissionKind/NotApplicable starts with capital letter for enumeration values
x	PermissionKind/Undefined	Now PermissionKind/Undefined starts with capital letter for enumeration values
	PermissionsPerObject	Name now consistent in figure and table (in table PermissionPerObject, needs to be PermissionsPerObject)

nc	V2.0 Change w.r.t. V1.0	Comment
x	PolicyAdministrationPoint/externalAccessControl	Type changed from Endpoint to Boolean, cardinality 1
x	PolicyInformationPoints/externalInformationPoint	Type changed from Endpoint to Boolean, cardinality 1 externalInformationPoint renamed to externalInformationPoints
x	Security/trustAnchor	Renamed to Security/certificate

Table 47. New Elements in Metamodel w.r.t. Security

V2.0	Comment
BlobCertificate	New class inheriting from Certificate
Certificate	Abstract class: was foreseen in V1.0 but not yet modelled
Security/requiredCertificateExtension	New attribute
PolicyEnforcementPoint	Was foreseen in V1.0 but not yet modelled
PolicyEnforcementPoint/externalPolicyEnforcementPoint	
PolicyDecisionPoint	Was foreseen in V1.0 but not yet modelled
PolicyDecisionPoint/externalPolicyDecisionPoint	

[4] Derived Schemata used Base64Binary and not hexBinary, therefore this change is considered to be backward compatible for most applications.

[5] Since *HasKind/kind* had the default *Instance,* this change has no impact if the attribute was omitted for submodel instances.

[6] Every model valid for V3.0RC02 is still valid in V3.0RC01, however there might be implementations that need to be changed if they assumed that the user can type case-insensitive names and get all elements that match the name in an case-insensitive way.

[7] Every model valid for V3.0RC02 is still valid in V3.0RC01, however there might be implementations that need to be changed if they assumed that the user can type case-insensitive names and get all elements that match the name in an case-insensitive way.

[8] There was an implicit constraint restricting the values to the values in the enumeration. This is now formalized.

[9] There was an implicit constraint that only IEC61360 data types are allowed to be used. This is now formalized.

Bibliography

[1] "Recommendations for implementing the strategic initiative INDUSTRIE 4.0", acatech, April 2013. [Online]. Available: https://www.acatech.de/Publikation/recommendations-for-implementing-the-strategic-initiative-industrie-4-0-final-report-of-the-industrie-4-0-working-group/

[2] "Implementation Strategy Industrie 4.0: Report on the results of the Industrie 4.0 Platform"; BITKOM e.V. / VDMA e.V., /ZVEI e.V., April 2015. [Online]. Available: https://www.bitkom.org/noindex/Publikationen/2016/Sonstiges/Implementation-Strategy-Industrie-40/2016-01-Implementation-Strategy-Industrie40.pdf

[3] DIN SPEC 91345:2016-04 "Referenzarchitekturmodell Industrie 4.0 (RAMI4.0) / Reference Architecture Model Industrie 4.0 (RAMI4.0) / Modèle de reference de l'architecture de l'industrie 4.0 (RAMI4.0)", ICS 03.100.01; 25.040.01; 35.240.50, April 2016. [Online]. Available: https://www.beuth.de/en/technical-rule/din-spec-91345-en/250940128

[4] "Structure of the Administration Shell, continuation of the development of the reference model for the Industrie 4.0 component", Plattform Industrie 4.0, Working Paper, April 2016. [Online]. Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/structure-of-the-administration-shell.html

[5] "Which criteria do Industrie 4.0 products need to fulfil? Guideline 2020", Federal Ministry for Economic Affairs and Energy (BMWi), July 2020. [Online]. Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/criteria-industrie-40-products_2020.html

[6] (German) "Beispiele zur Verwaltungsschale der Industrie 4.0-Komponente – Basisteil"; ZVEI e.V., Whitepaper, November 2016. [Online]. Available: https://www.zvei.org/presse-medien/publikationen/beispiele-zur-verwaltungsschale-der-industrie-40-komponente-basisteil/

[7] "Aspects of the research roadmap in application scenarios", Plattform Industrie 4.0, working paper, April 2016. [Online]. Available: http://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/aspects-of-the-research-roadmap.html

[8] (German) "Fortschreibung der Anwendungsszenarien der Plattform Industrie 4.0"; Plattform Industrie 4.0, Ergebnispapier, October 2016. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/fortschreibung-anwendungsszenarien.html

[9] "Security in RAMI4.0", Plattform Industrie 4.0, Berlin, technical overview, April 2016. [Online]. Available: http://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/security-rami40-en.html

[10] "Die Deutsche Normungs-Roadmap Industrie 4.0 / The German standardization roadmap Industrie 4.0", DKE Deutsche Kommission Elektrotechnik, Elektronik Informationstechnik im DIN und VDE, Version 2.0, 2015. [Online]. Available: http://www.din.de/de/forschung-und-innovation/industrie4-0/roadmap-industrie40-

[11] "Weiterentwicklung des Interaktionsmodells für Industrie 4.0-Komponenten", Plattform Industrie 4.0, discussion paper, November 2016. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/interaktionsmodell-i40-komponenten-it-gipfel.html

[13] "Relationships between I4.0 Components – Composite Components and Smart Production", Plattform Industrie 4.0, Berlin, working paper, June 2017. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/hm-2018-relationship.html

[14] "Industrie 4.0 Plug-and-Produce for Adaptable Factories"; Plattform Industrie 4.0, Berlin, working paper, June 2017. [Online]. Available: http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/Industrie-40-%20Plug-and-Produce.html

[15] "Security der Verwaltungsschale / Security of the Administration Shell", Plattform Industrie 4.0, Berlin, working paper, April 2017. [Online]. Available: http://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/security-der-verwaltungsschale.html

[16] DIN SPEC 92000:2019-09 "Data Exchange on the Base of Property Value Statements (PVSX)", 2019 September.

[17] "(German) Verwaltungsschale in der Praxis. Wie definiere ich Teilmodelle, beispielhafte Teilmodelle und Interaktion zwischen Verwaltungsschalen", Version 1.0, April 2019, Plattform Industrie 4.0 in Kooperation mit VDI/VDE-GMA Fachausschuss 7.20, Federal Ministry for Economic Affairs and Energy (BMWi), Available: https://www.plattform-i40.de/PI40/Redaktion/DE/Downloads/Publikation/2019-verwaltungsschale-in-der-praxis.html

[18] (German) "I4.0-Sprache. Vokabular, Nachrichtenstruktur und semantische Interaktionsprotokolle der I4.0-Sprache", Plattform Industrie 4.0 in Kooperation mit VDI/VDE-GMA Fachausschuss 7.20, April 2018. [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/DE/Downloads/Publikation/hm-2018-sprache.html

[19] "The Structure of the Administration Shell: TRILATERAL PERSPECTIVES from France, Italy and Germany", March 2018, [Online]. Available: https://www.plattform-i40.de/I40/Redaktion/EN/Downloads/Publikation/hm-2018-trilaterale-coop.html

[20] "Industrial automation systems and integration — Exchange of characteristic data — Part 10: Characteristic data exchange format", Technical Specification ISO/TS 29002-10:2009(E), 2009

[21] "Smart Manufacturing - Reference Architecture Model Industry 4.0 (RAMI4.0)", IEC PAS 63088, International Electrotechnical Commission (IEC), 2017

[22] "System.IO.Packaging Namespace", MSDN, Accessed: 2019-01-26 [Online]. Available: https://msdn.microsoft.com/en-us/library/system.io.packaging(v=vs.110).aspx

[23] DIN SPEC 16593-1 "Reference Model for Industrie 4.0 Service Architectures – Part 1: Basic Concepts of an Interaction-based Architecture", Beuth-Verlag: Berlin, Germany, 2018. [Online]. Available: https://www.beuth.de/en/technical-rule/din-spec-16593-1/287632675

[24] ISO 13854-42 "Standard data element types with associated classification scheme – Part 1: Definitions – Principles and methods" Edition 4.0, 2017-07

[25] IEC 61360-1 "Standard data element types with associated classification scheme – Part 1: Definitions – Principles and methods", Edition 4.0, 2017-07

[26] ISO/TS 29002-10:2009(E) "Industrial automation systems and integration — Exchange of characteristic data — Part 10: Characteristic data exchange format", First edition 2009-12-01

[27] A. Bayha, J. Bock, B. Boss, C. Diedrich, S. Malakuti "Describing Capabilities of Industrie 4.0 Components". Nov. 2020. Plattform Industrie 4.0. [Online] Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Capabilities_Industrie40_Components.html

[28] AutomationML Association: "Application Recommendations: Asset Administration Shell Representation (AR 004E)", Version 1.0.0, 20.11.2019, [Online]. Available: https://www.automationml.org/o.red.c/dateien.html

[29] H. Knublauch, D. Knotokostas "Shapes Constraint Language (SHACL)" W3C Recommendation, 2017, [Online]. Available: https://www.w3.org/TR/shacl/

[30] "I4AAS – Industrie 4.09 Asset Administration Shell". June 2021. [Online] Available: https://opcfoundation.org/markets-collaboration/I4AAS/

[31] DIN SPEC 91406:2019 "Automatische Identifikation von physischen Objekten und Informationen zum physischen Objekt in IT-Systemen, insbesondere IoT-Systemen/Automatic identification of physical objects and information on physical objects in IT systems, particularly IoT systems". December 2019

[32] F. Manola, E. Miller "RDF 1.1 Primer" W3C Recommendation, 2014, [Online]. Available: https://www.w3.org/TR/rdf11-primer/

[33] T. R. Gruber "A translation approach to portable ontology specifications." Knowledge acquisition 5.2 (1993): 199-220. [Online]. Available: https://tomgruber.org/writing/ontolingua-kaj-1993.htm

[34] "The Industrial Internet of Things Vocabulary". Technical Report. Version 2.3. October 10, 2020. Industrial Internet Consortium. IIC:IIVOC:V2.3:20201025 [Online] Available: https://www.iiconsortium.org/vocab/

[36] T. Preston-Werner "Semantic Versioning". Version 2.0.0. Accessed: 2020-11-13. [Online] Available: https://semver.org/spec/v2.0.0.html

[37] "Details of the Asset Administration Shell – Interoperability at Runtime – Exchanging Information via

Application Programming Interfaces". See [46].

[38] "Asset Administration Shell. Reading Guide". Plattform Industrie 4.0 in cooperation with IDTA. November 2020. See [46].

[39] "Submodel Templates of the Asset Administration Shell - Generic Frame for Technical Data for Industrial Equipment in Manufacturing", Version 1.1, Nov. 2020, Plattform Industrie 4.0 [Online] Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Submodel_templates-Asset_Administration_Shell-Technical_Data.html

[40] "Submodel Templates of the Asset Administration Shell - ZVEI Digital Nameplate for industrial equipment", Version 1.0, Nov. 2020, Plattform Industrie 4.0 [Online] Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Submodel_templates-Asset_Administration_Shell-digital_nameplate.html

[41] OPC 30270: OPC UA for Asset Administration Shell (AAS). 2021-06-04. [Online]. Available: https://reference.opcfoundation.org/v104/I4AAS/v100/docs/

[42] OPC Unified Architecture Specification. Part 5 Information Model. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-unified-architecture

[43] OPC UA Information Models. [Online]. Available: https://opcfoundation.org/developer-tools/specifications-opc-ua-information-models

[44] IEC 63278-1 "Asset Administration Shell for industrial applications – Part 1: Asset Administration Shell structure". 95/925/CDV

[45] "Registered AAS Submodel Templates". Industrial Digital Twin Association. Available: https://industrialdigitaltwin.org/en/content-hub/submodels

[46] "Asset Administration Shell Specifications – Quicklinks to Different Versions & Reading Guide". [Online]. Available: https://www.plattform-i40.de/IP/Redaktion/EN/Standardartikel/specification-administrationshell.html

[47] (German) "I4.0-Sprache. Vokabular, Nachrichtenstruktur und semantische Interaktionsprotokolle der I4.0-Sprache", Discussion Paper. Plattform Industrie 4.0 [Online] Available: https://www.plattform-i40.de/IP/Redaktion/DE/Downloads/Publikation/hm-2018-sprache.html

[48] "How to create a submodel template specification". Guideline. December 2022. Industrial Digital Twin Association. Available: https://industrialdigitaltwin.org/wp-content/uploads/2022/12/I40-IDTA-WS-Process-How-to-write-a-SMT-FINAL-.pdf

[49] Vincent Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller and Karen Scarfone, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations", NIST Special Publication 800-162, Jan. 2014. [Online]. Available: http://dx.doi.org/10.6028/NIST.SP.800-162

[50] "Secure Download Service", Discussion Paper. Oct. 2020, Plattform Industrie 4.0 [Online] Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/secure_downloadservice.html

[51] "AAS Repository. Repository for Information and Code for the Asset Administration Shell". https://github.com/admin-shell-io