March 2023

Specification of the Asset Adminitration Shell

Part 1: Metamodel

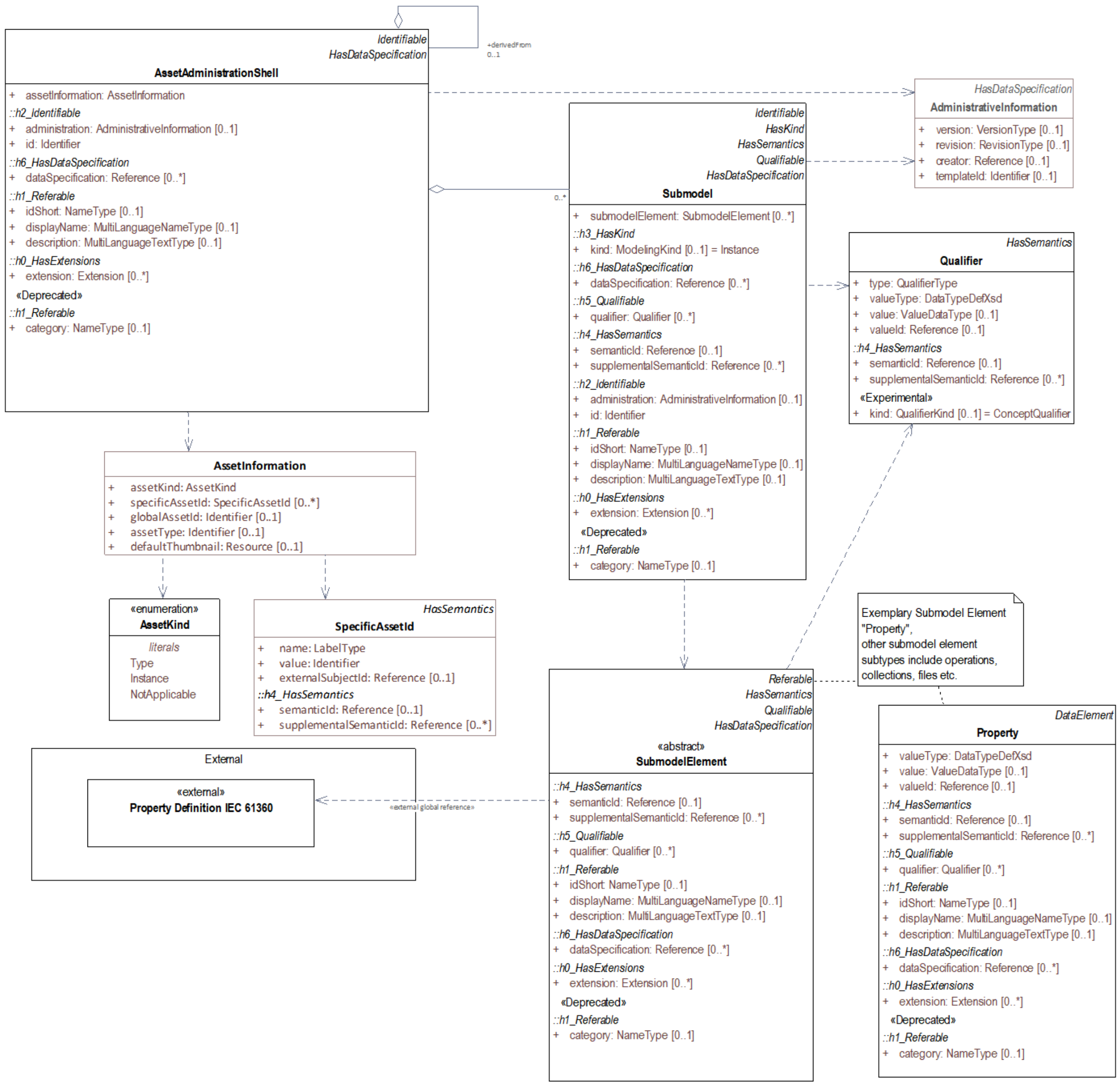
SPECIFICATION

IDTA Number: 01001

Asset Administration Shell: Classes

Overview

In this annex, some UML diagrams are shown together with all attributes inherited for a better overview.



Identifiable
HasDataSpecification
AssetAdministrationShell
+ assetInformation: AssetInformation
::h2_Identifiable
+ administration: AdministrativeInformation [0..1]
+ id: Identifier
::h6_HasDataSpecification
+ dataSpecification: Reference [0..*]
::h1_Referable
+ idShort: NameType [0..1]
+ displayName: MultiLanguageNameType [0..1]
+ description: MultiLanguageTextType [0..1]
::h0_HasExtensions
+ extension: Extension [0..*]
«Deprecated»
::h1_Referable
+ category: NameType [0..1]
+derivedFrom
0..1
0..*
HasDataSpecification
AdministrativeInformation
+ version: VersionType [0..1]
+ revision: RevisionType [0..1]
+ creator: Reference [0..1]
+ templateId: Identifier [0..1]
Identifiable
HasKind
HasSemantics
Qualifiable
HasDataSpecification
Submodel
+ submodelElement: SubmodelElement [0..*]
::h3_HasKind
+ kind: ModelingKind [0..1] = Instance
::h6_HasDataSpecification
+ dataSpecification: Reference [0..*]
::h5_Qualifiable
+ qualifier: Qualifier [0..*]
::h4_HasSemantics
+ semanticId: Reference [0..1]
+ supplementalSemanticId: Reference [0..*]
::h2_Identifiable
+ administration: AdministrativeInformation [0..1]
+ id: Identifier
::h1_Referable
+ idShort: NameType [0..1]
+ displayName: MultiLanguageNameType [0..1]
+ description: MultiLanguageTextType [0..1]
::h0_HasExtensions
+ extension: Extension [0..*]
«Deprecated»
::h1_Referable
+ category: NameType [0..1]
HasSemantics
Qualifier
+ type: QualifierType
+ valueType: DataTypeDefXsd
+ value: ValueDataType [0..1]
+ valueId: Reference [0..1]
::h4_HasSemantics
+ semanticId: Reference [0..1]
+ supplementalSemanticId: Reference [0..*]
«Experimental»
+ kind: QualifierKind [0..1] = ConceptQualifier
AssetInformation
+ assetKind: AssetKind
+ specificAssetId: SpecificAssetId [0..*]
+ globalAssetId: Identifier [0..1]
+ assetType: Identifier [0..1]
+ defaultThumbnail: Resource [0..1]
«enumeration»
AssetKind
literals
Type
Instance
NotApplicable
HasSemantics
SpecificAssetId
+ name: LabelType
+ value: Identifier
+ externalSubjectId: Reference [0..1]
::h4_HasSemantics
+ semanticId: Reference [0..1]
+ supplementalSemanticId: Reference [0..*]
Exemplary Submodel Element "Property", other submodel element subtypes include operations, collections, files etc.
External
«external»
Property Definition IEC 61360
«external global reference»
Referable
HasSemantics
Qualifiable
HasDataSpecification
«abstract»
SubmodelElement
::h4_HasSemantics
+ semanticId: Reference [0..1]
+ supplementalSemanticId: Reference [0..*]
::h5_Qualifiable
+ qualifier: Qualifier [0..*]
::h1_Referable
+ idShort: NameType [0..1]
+ displayName: MultiLanguageNameType [0..1]
+ description: MultiLanguageTextType [0..1]
::h6_HasDataSpecification
+ dataSpecification: Reference [0..*]
::h0_HasExtensions
+ extension: Extension [0..*]
«Deprecated»
::h1_Referable
+ category: NameType [0..1]
DataElement
Property
+ valueType: DataTypeDefXsd
+ value: ValueDataType [0..1]
+ valueId: Reference [0..1]
::h4_HasSemantics
+ semanticId: Reference [0..1]
+ supplementalSemanticId: Reference [0..*]
::h5_Qualifiable
+ qualifier: Qualifier [0..*]
::h1_Referable
+ idShort: NameType [0..1]
+ displayName: MultiLanguageNameType [0..1]
+ description: MultiLanguageTextType [0..1]
::h6_HasDataSpecification
+ dataSpecification: Reference [0..*]
::h0_HasExtensions
+ extension: Extension [0..*]
«Deprecated»
::h1_Referable
+ category: NameType [0..1]

Figure 1. 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.

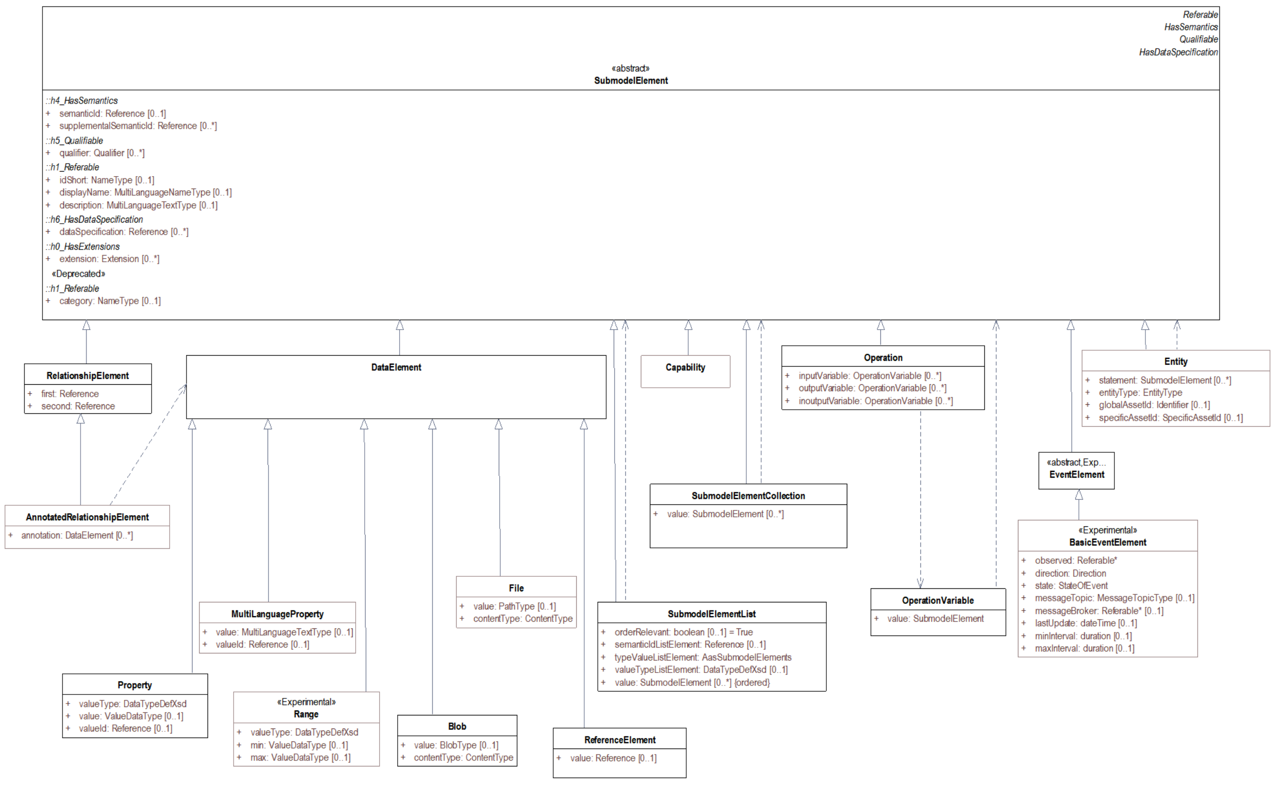

Referable
HasSemantics
Qualifiable
HasDataSpecification
«abstract»
SubmodelElement
::h4_HasSemantics
+ semanticId: Reference [0..1]
+ supplementalSemanticId: Reference [0..*]
::h5_Qualifiable
+ qualifier: Qualifier [0..*]
::h1_Referable
+ idShort: NameType [0..1]
+ displayName: MultiLanguageNameType [0..1]
+ description: MultiLanguageTextType [0..1]
::h6_HasDataSpecification
+ dataSpecification: Reference [0..*]
::h0_HasExtensions
+ extension: Extension [0..*]
«Deprecated»
::h1_Referable
+ category: NameType [0..1]
RelationshipElement
+ first: Reference
+ second: Reference
DataElement
Capability
Operation
+ inputVariable: OperationVariable [0..*]
+ outputVariable: OperationVariable [0..*]
+ inoutputVariable: OperationVariable [0..*]
Entity
+ statement: SubmodelElement [0..*]
+ entityType: EntityType
+ globalAssetId: Identifier [0..1]
+ specificAssetId: SpecificAssetId [0..1]
«abstract,Exp...
EventElement
AnnotatedRelationshipElement
+ annotation: DataElement [0..*]
SubmodelElementCollection
+ value: SubmodelElement [0..*]
«Experimental»
BasicEventElement
+ observed: Referable*
+ direction: Direction
+ state: StateOfEvent
+ messageTopic: MessageTopicType [0..1]
+ messageBroker: Referable* [0..1]
+ lastUpdate: dateTime [0..1]
+ minInterval: duration [0..1]
+ maxInterval: duration [0..1]
File
+ value: PathType [0..1]
+ contentType: ContentType
MultiLanguageProperty
+ value: MultiLanguageTextType [0..1]
+ valueId: Reference [0..1]
SubmodelElementList
+ orderRelevant: boolean [0..1] = True
+ semanticIdListElement: Reference [0..1]
+ typeValueListElement: AasSubmodelElements
+ valueTypeListElement: DataTypeDefXsd [0..1]
+ value: SubmodelElement [0..*] {ordered}
OperationVariable
+ value: SubmodelElement
Property
+ valueType: DataTypeDefXsd
+ value: ValueDataType [0..1]
+ valueId: Reference [0..1]
«Experimental»
Range
+ valueType: DataTypeDefXsd
+ min: ValueDataType [0..1]
+ max: ValueDataType [0..1]
Blob
+ value: BlobType [0..1]
+ contentType: ContentType
ReferenceElement
+ value: Reference [0..1]

Figure 2. Model for Submodel Elements with Inherited Attributes

Common used classes

Administrative Information

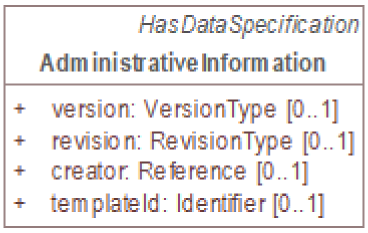

HasDataSpecification
AdministrativeInformation
+ version: VersionType [0..1]
+ revision: RevisionType [0..1]
+ creator: Reference [0..1]
+ templateId: Identifier [0..1]

Figure 3. 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

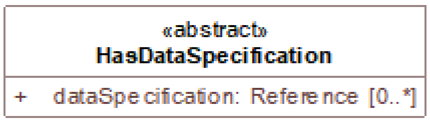


Figure 4. Metamodel of HasDataSpecification

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

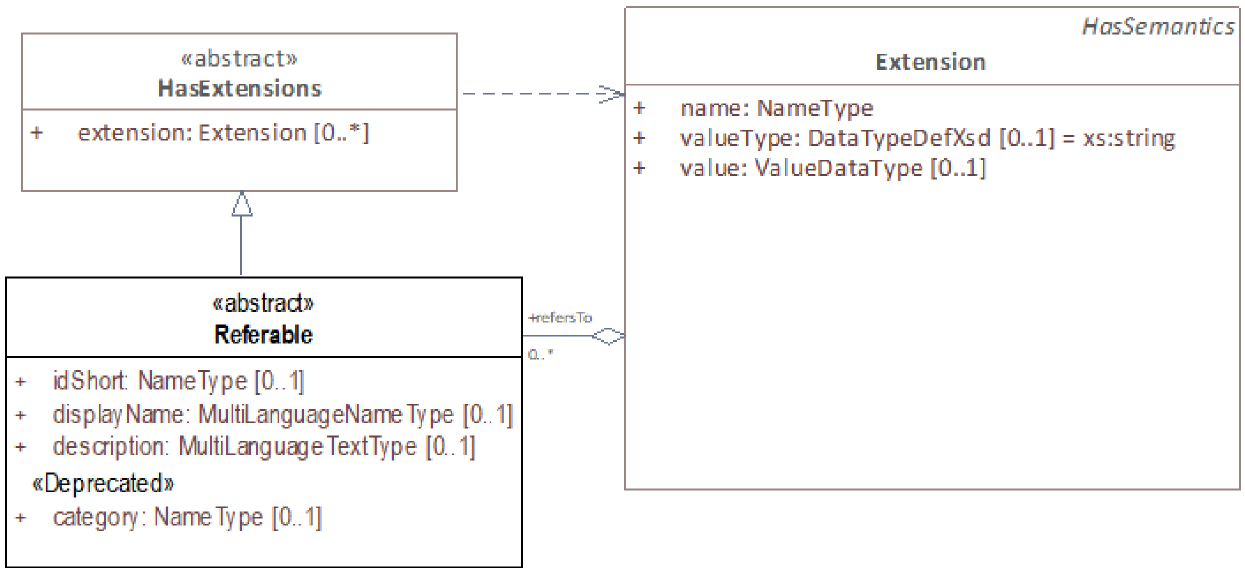

«abstract»
HasExtensions
+ extension: Extension [0..*]
HasSemantics
Extension
+ name: NameType
+ valueType: DataTypeDefXsd [0..1] = xs:string
+ value: ValueDataType [0..1]
«abstract»
Referable
+ idShort: NameType [0..1]
+ displayName: MultiLanguageNameType [0..1]
+ description: MultiLanguageTextType [0..1]
«Deprecated»
+ category: NameType [0..1]
+refersTo
0..*

Figure 5. 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..*

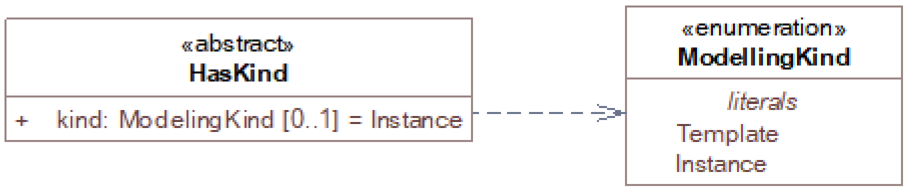


Figure 6. Metamodel of HasKind

Has Kind

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:	—

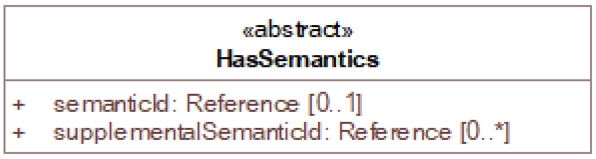
Attribute	Explanation	Type	Card.
kind	Kind of the element: either type or instance Default Value = *Instance*	ModellingKind	0..1

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



«abstract»
HasSemantics
+ semanticId: Reference [0..1]
+ supplementalSemanticId: Reference [0..*]

Figure 7. 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

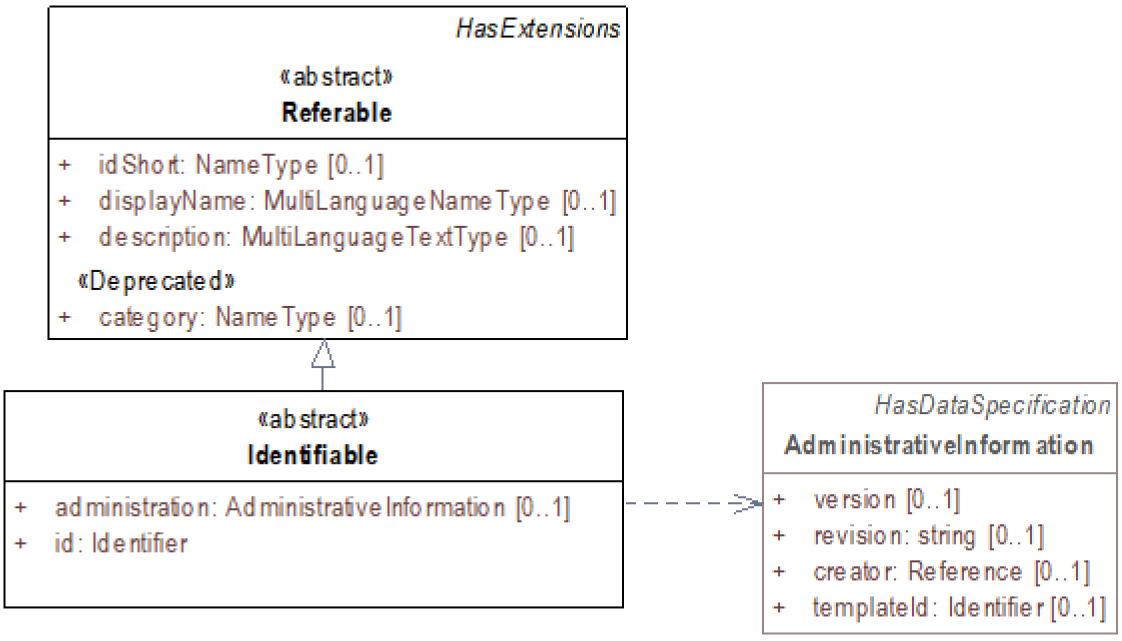

HasExtensions
«abstract»
Referable
+ idShort: NameType [0..1]
+ displayName: MultiLanguageNameType [0..1]
+ description: MultiLanguageTextType [0..1]
«Deprecated»
+ category: NameType [0..1]
«abstract»
Identifiable
+ administration: AdministrativeInformation [0..1]
+ id: Identifier
HasDataSpecification
AdministrativeInformation
+ version [0..1]
+ revision: string [0..1]
+ creator: Reference [0..1]
+ templateId: Identifier [0..1]

Figure 8. 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

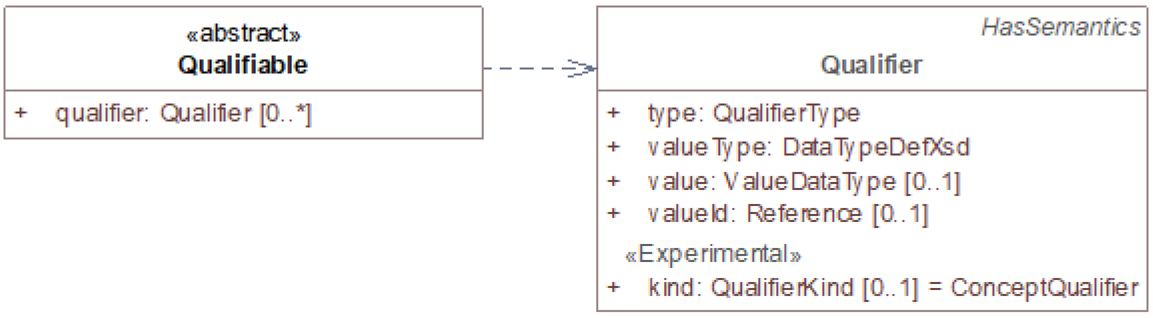

«abstract»
Qualifiable
+ qualifier: Qualifier [0..*]
HasSemantics
Qualifier
+ type: QualifierType
+ valueType: DataTypeDefXsd
+ value: ValueDataType [0..1]
+ valueId: Reference [0..1]
«Experimental»
+ kind: QualifierKind [0..1] = ConceptQualifier

Figure 9. 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

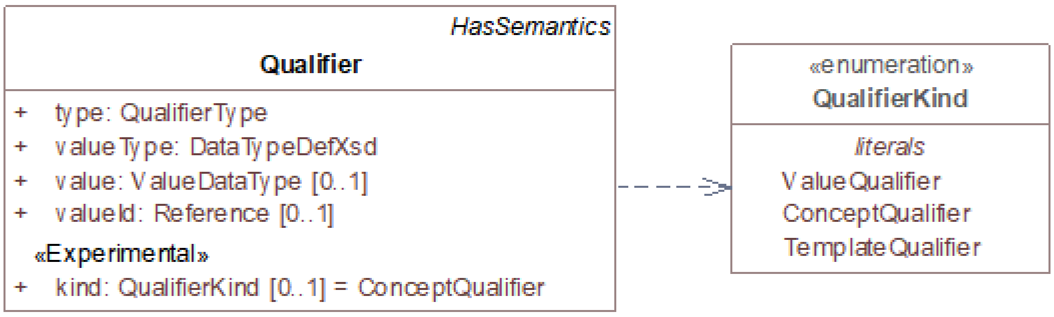


Figure 10. Metamodel of Qualifiers

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

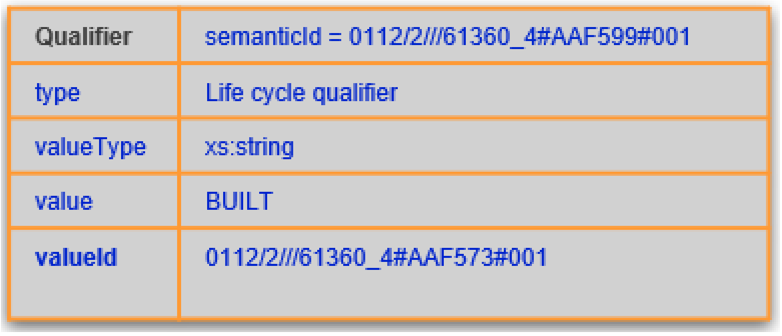
Set of:	—

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].



Qualifier	semanticId = 0112/2///61360_4#AAF599#001
type	Life cycle qualifier
valueType	xs:string
value	BUILT
valueId	0112/2///61360_4#AAF573#001

Figure 11. Example: Qualifier from IEC CDD

Referable

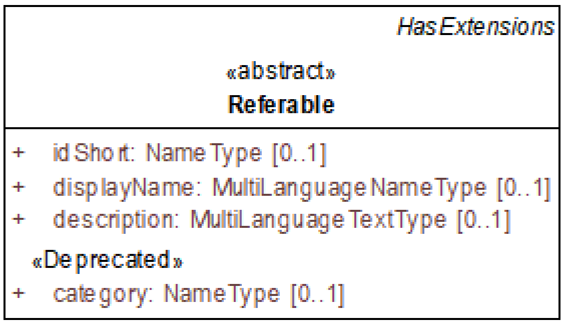

HasExtensions
«abstract»
Referable
+ idShort: NameType [0..1]
+ displayName: MultiLanguageNameType [0..1]
+ description: MultiLanguageTextType [0..1]
«Deprecated»
+ category: NameType [0..1]

Figure 12. 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 1. Categories[1] 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.

[1] Note: categories of referables are deprecated.

Asset Administration Shell

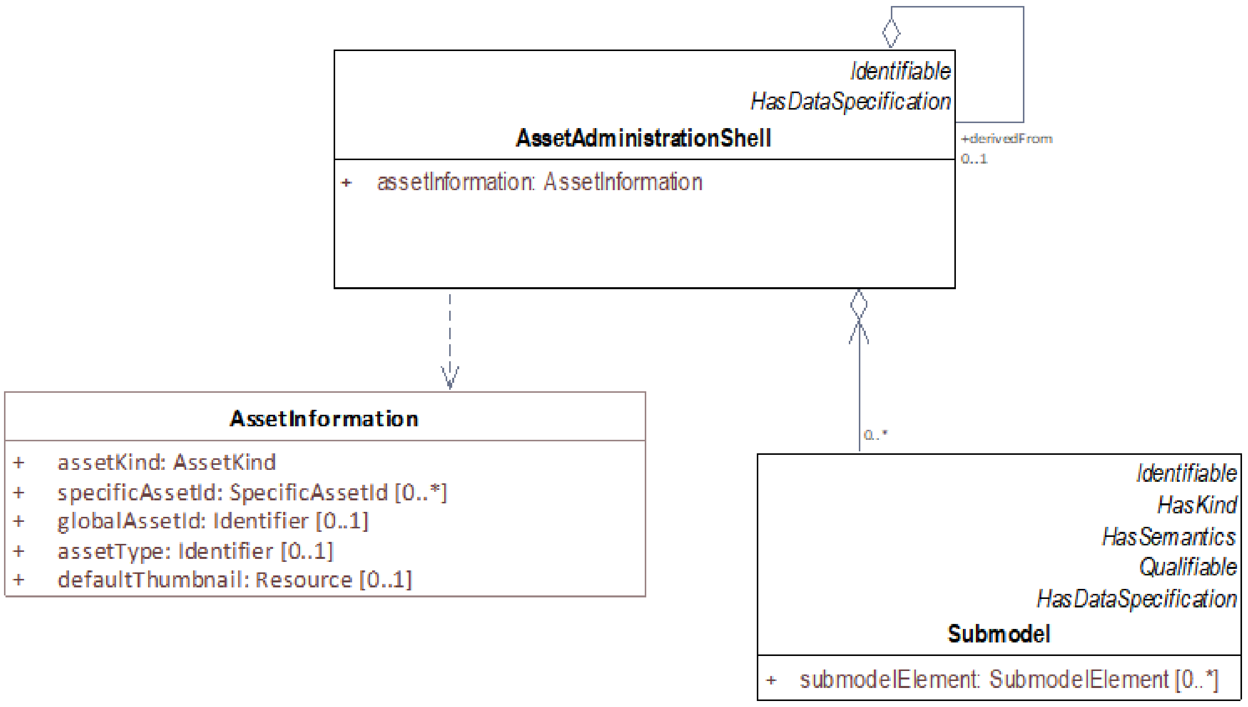

Identifiable
HasDataSpecification
AssetAdministrationShell
+ assetInformation: AssetInformation
+derivedFrom
0..1
AssetInformation
+ assetKind: AssetKind
+ specificAssetId: SpecificAssetId [0..*]
+ globalAssetId: Identifier [0..1]
+ assetType: Identifier [0..1]
+ defaultThumbnail: Resource [0..1]
0..*
Identifiable
HasKind
HasSemantics
Qualifiable
HasDataSpecification
Submodel
+ submodelElement: SubmodelElement [0..*]

Figure 13. Metamodel of an AssetAdministrationShell

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

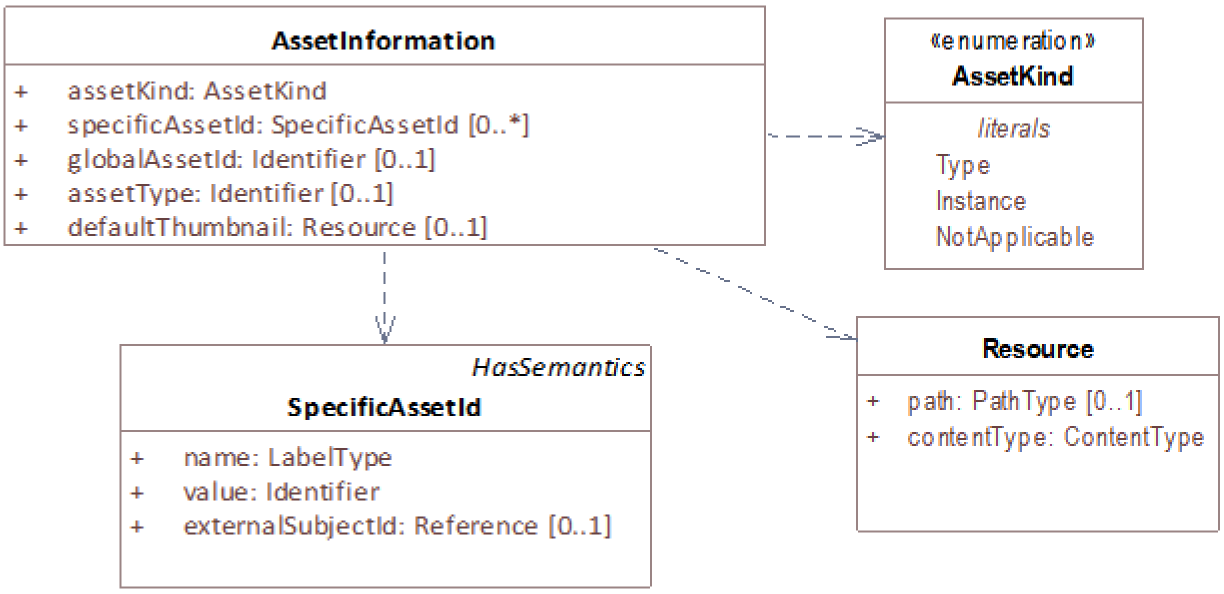
The *derivedFrom* attribute is used to establish a relationship between two Asset Administration Shells that are derived from each other. For more detailed information on the *derivedFrom* concept, see Clause 4.2.

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..*

Asset Information



AssetInformation
+ assetKind: AssetKind
+ specificAssetId: SpecificAssetId [0..*]
+ globalAssetId: Identifier [0..1]
+ assetType: Identifier [0..1]
+ defaultThumbnail: Resource [0..1]
«enumeration»
AssetKind
literals
Type
Instance
NotApplicable
HasSemantics
SpecificAssetId
+ name: LabelType
+ value: Identifier
+ externalSubjectId: Reference [0..1]
Resource
+ path: PathType [0..1]
+ contentType: ContentType

Figure 14. 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

Attribute	Explanation	Type	Card.
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..*
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
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.

Submodel

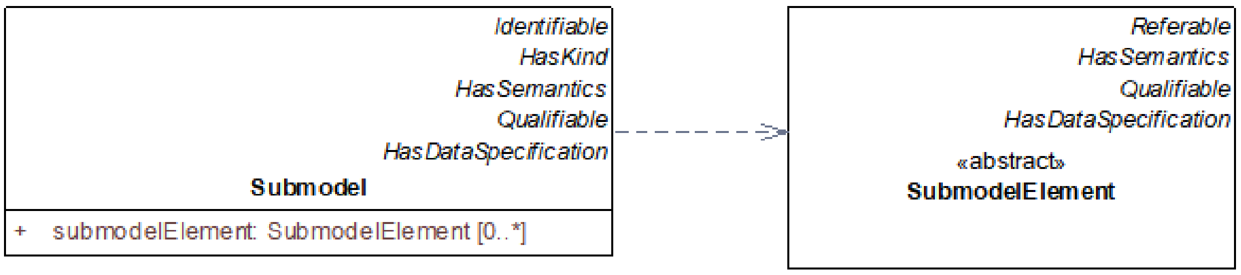


Figure 15. Metamodel of Submodel

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..*

Submodel Element

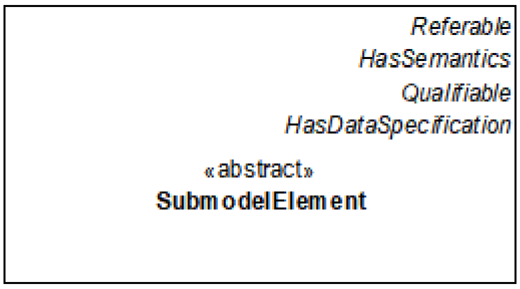


Figure 16. Metamodel of Submodel Element

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.**

Submodel Element Types

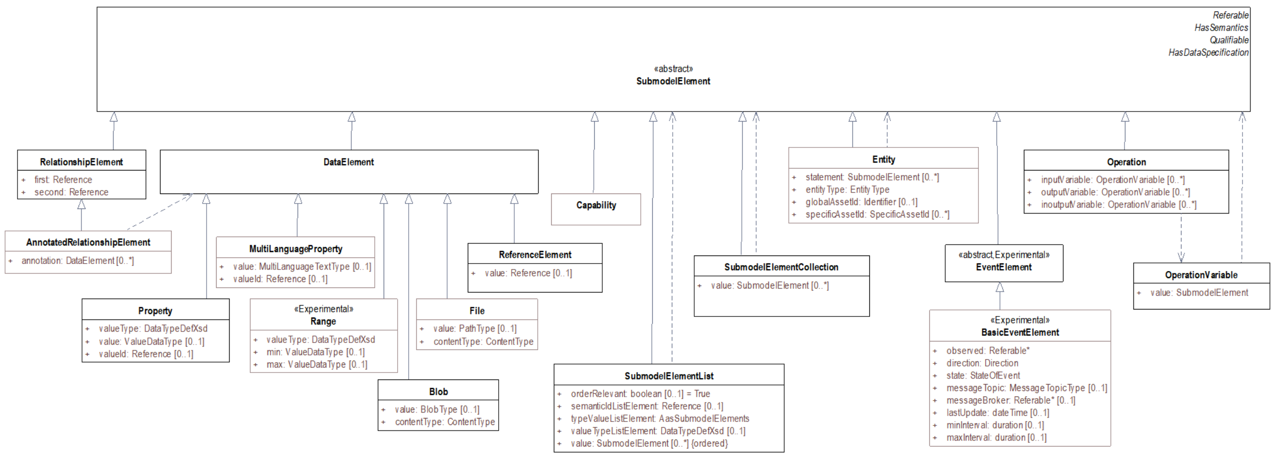

Referable
HasSemantics
Qualifiable
HasDataSpecification
«abstract»
SubmodelElement
RelationshipElement
+ first: Reference
+ second: Reference
DataElement
Capability
Entity
+ statement: SubmodelElement [0..*]
+ entity Type: Entity Type
+ globalAssetId: Identifier [0..1]
+ specificAssetId: SpecificAssetId [0..*]
Operation
+ inputVariable: OperationVariable [0..*]
+ outputVariable: OperationVariable [0..*]
+ inoutputVariable: OperationVariable [0..*]
AnnotatedRelationshipElement
+ annotation: DataElement [0..*]
MultiLanguageProperty
+ value: MultiLanguageTextType [0..1]
+ valueId: Reference [0..1]
ReferenceElement
+ value: Reference [0..1]
SubmodelElementCollection
+ value: SubmodelElement [0..*]
«abstract,Experimental»
EventElement
OperationVariable
+ value: SubmodelElement
Property
+ valueType: DataTypeDefXsd
+ value: ValueDataType [0..1]
+ valueId: Reference [0..1]
«Experimental»
Range
+ valueType: DataTypeDefXsd
+ min: ValueDataType [0..1]
+ max: ValueDataType [0..1]
File
+ value: PathType [0..1]
+ contentType: ContentType
«Experimental»
BasicEventElement
+ observed: Referable*
+ direction: Direction
+ state: StateOfEvent
+ messageTopic: MessageTopicType [0..1]
+ messageBroker: Referable* [0..1]
+ lastUpdate: dateTime [0..1]
+ minInterval: duration [0..1]
+ maxInterval: duration [0..1]
Blob
+ value: BlobType [0..1]
+ contentType: ContentType
SubmodelElementList
+ orderRelevant: boolean [0..1] = True
+ semanticIdListElement: Reference [0..1]
+ typeValueListElement: AasSubmodelElements
+ valueTypeListElement: DataTypeDefXsd [0..1]
+ value: SubmodelElement [0..*] {ordered}

Figure 17. 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).

Annotated Relationship Element

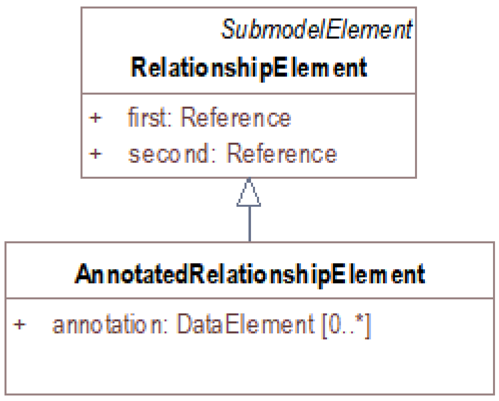


Figure 18. Metamodel of Annotated Relationship Elements

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

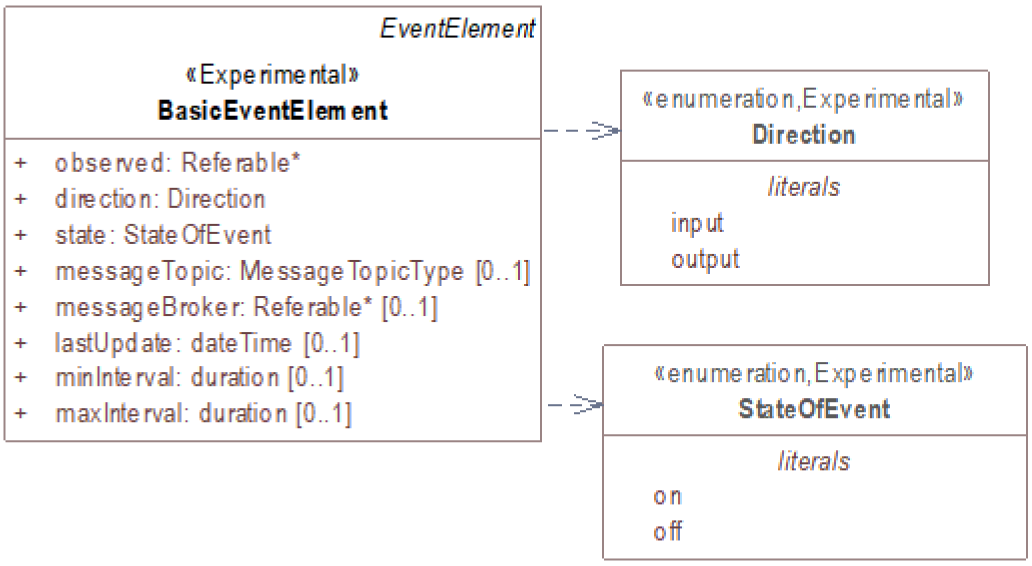

EventElement
«Experimental»
BasicEventElement
+ observed: Referable*
+ direction: Direction
+ state: StateOfEvent
+ messageTopic: MessageTopicType [0..1]
+ messageBroker: Referable* [0..1]
+ lastUpdate: dateTime [0..1]
+ minInterval: duration [0..1]
+ maxInterval: duration [0..1]
«enumeration,Experimental»
Direction
literals
input
output
«enumeration,Experimental»
StateOfEvent
literals
on
off

Figure 19. 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.

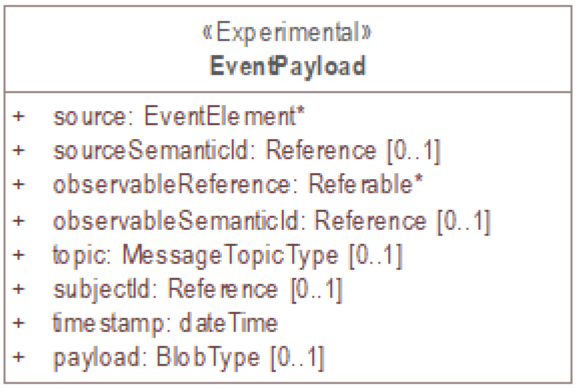

«Experimental»
EventPayload
+ source: EventElement*
+ sourceSemanticId: Reference [0..1]
+ observableReference: Referable*
+ observableSemanticId: Reference [0..1]
+ topic: MessageTopicType [0..1]
+ subjectId: Reference [0..1]
+ timestamp: dateTime
+ payload: BlobType [0..1]

Figure 20. 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

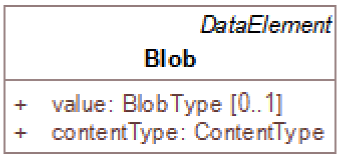


Figure 21. Metamodel of Blobs

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

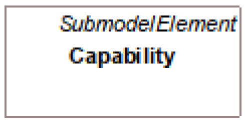


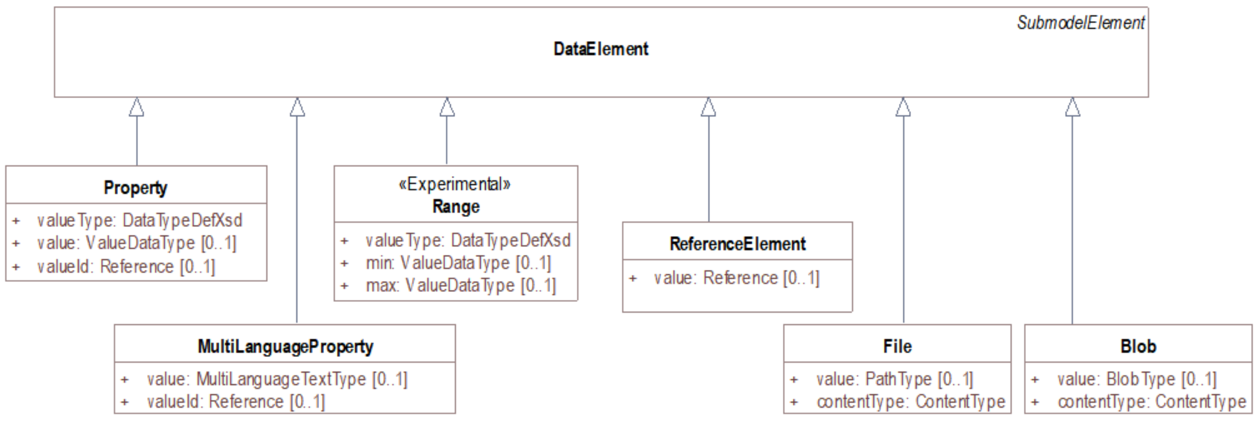
Figure 22. Metamodel of Capabilities

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



SubmodelElement
DataElement
Property
+ valueType: DataTypeDefXsd
+ value: ValueDataType [0..1]
+ valueId: Reference [0..1]
«Experimental»
Range
+ valueType: DataTypeDefXsd
+ min: ValueDataType [0..1]
+ max: ValueDataType [0..1]
ReferenceElement
+ value: Reference [0..1]
MultiLanguageProperty
+ value: MultiLanguageTextType [0..1]
+ valueId: Reference [0..1]
File
+ value: PathType [0..1]
+ contentType: ContentType
Blob
+ value: BlobType [0..1]
+ contentType: ContentType

Figure 23. 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

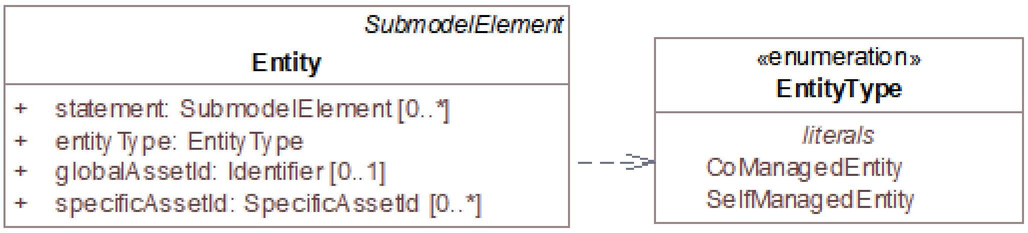

SubmodelElement
Entity
+ statement: SubmodelElement [0..*]
+ entityType: EntityType
+ globalAssetId: Identifier [0..1]
+ specificAssetId: SpecificAssetId [0..*]
«enumeration»
EntityType
literals
CoManagedEntity
SelfManagedEntity

Figure 24. 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

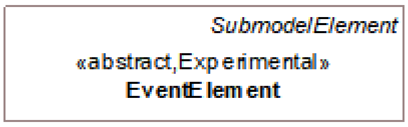


Figure 25. Metamodel of Events

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

Attribute	Explanation	Type	Card.

File

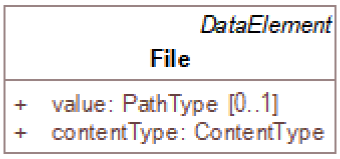


Figure 26. Metamodel of File Submodel Element

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

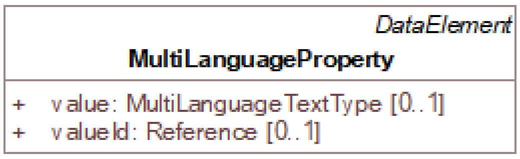

DataElement
MultiLanguageProperty
+ value: MultiLanguageTextType [0..1]
+ valueId: Reference [0..1]

Figure 27. 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

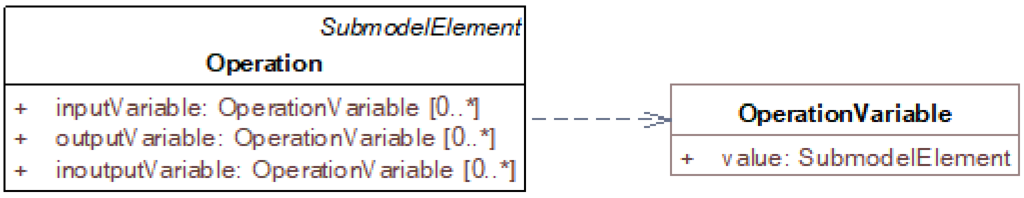

SubmodelElement
Operation
+ inputVariable: OperationVariable [0..*]
+ outputVariable: OperationVariable [0..*]
+ inoutputVariable: OperationVariable [0..*]
OperationVariable
+ value: SubmodelElement

Figure 28. 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

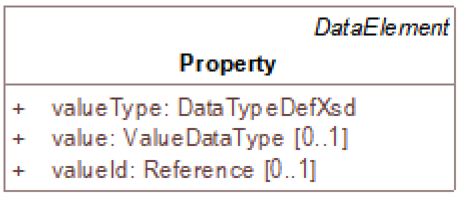

DataElement
Property
+ valueType: DataTypeDefXsd
+ value: ValueDataType [0..1]
+ valueId: Reference [0..1]

Figure 29. 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

Attribute	Explanation	Type	Card.
value	The value of the property instance	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

Range

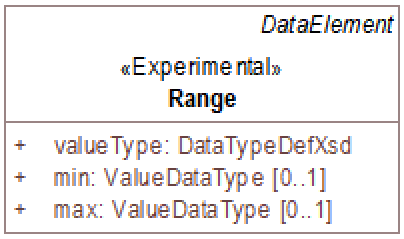

DataElement
«Experimental»
Range
+ valueType: DataTypeDefXsd
+ min: ValueDataType [0..1]
+ max: ValueDataType [0..1]

Figure 30. 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

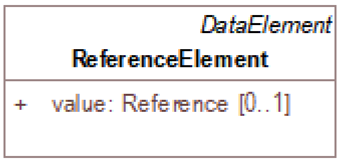


Figure 31. Metamodel of Reference Elements

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

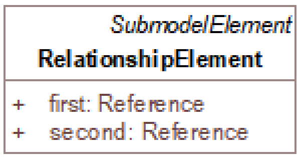


Figure 32. Metamodel of Relationship Elements

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
second	Reference to the second element in the relationship taking the role of the object	Reference	1

Submodel Element Collection

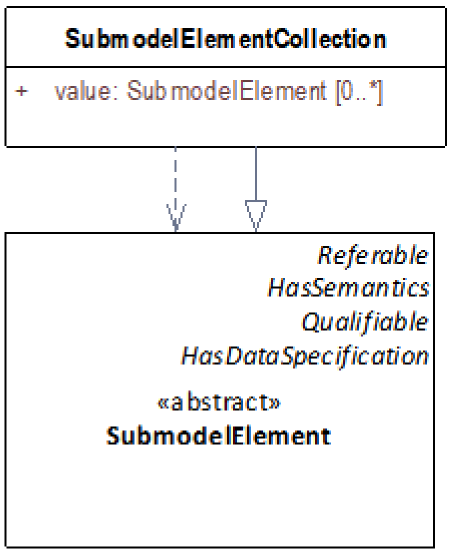


Figure 33. Metamodel of Submodel Element Collections

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

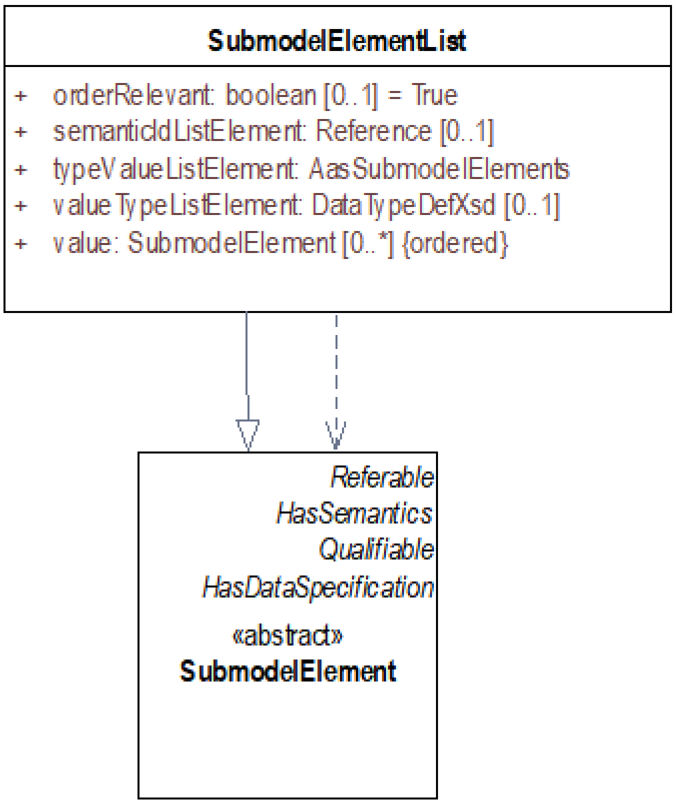

SubmodelElementList
+ orderRelevant: boolean [0..1] = True
+ semanticIdListElement: Reference [0..1]
+ typeValueListElement: AasSubmodelElements
+ valueTypeListElement: DataTypeDefXsd [0..1]
+ value: SubmodelElement [0..*] {ordered}
Referable
HasSemantics
Qualifiable
HasDataSpecification
«abstract»
SubmodelElement

Figure 34. 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

Concept Description

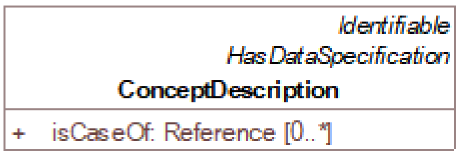


Figure 35. Metamodel of Concept Descriptions

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.

Environment

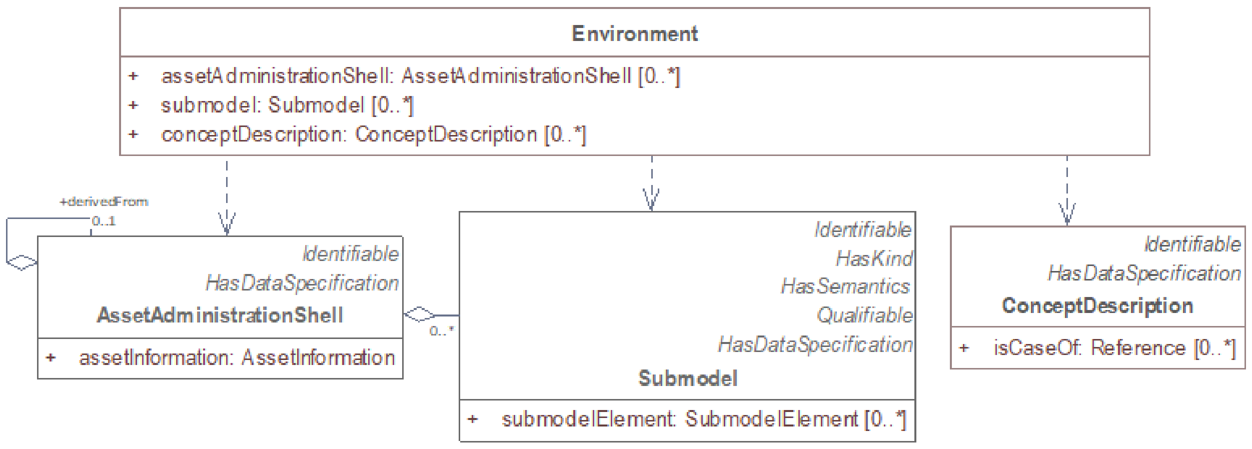

Environment
+ assetAdministrationShell: AssetAdministrationShell [0..*]
+ submodel: Submodel [0..*]
+ conceptDescription: ConceptDescription [0..*]
+derivedFrom
0..1
Identifiable
HasDataSpecification
AssetAdministrationShell
+ assetInformation: AssetInformation
0..*
Identifiable
HasKind
HasSemantics
Qualifiable
HasDataSpecification
Submodel
+ submodelElement: SubmodelElement [0..*]
Identifiable
HasDataSpecification
ConceptDescription
+ isCaseOf: Reference [0..*]

Figure 36. 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..*

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

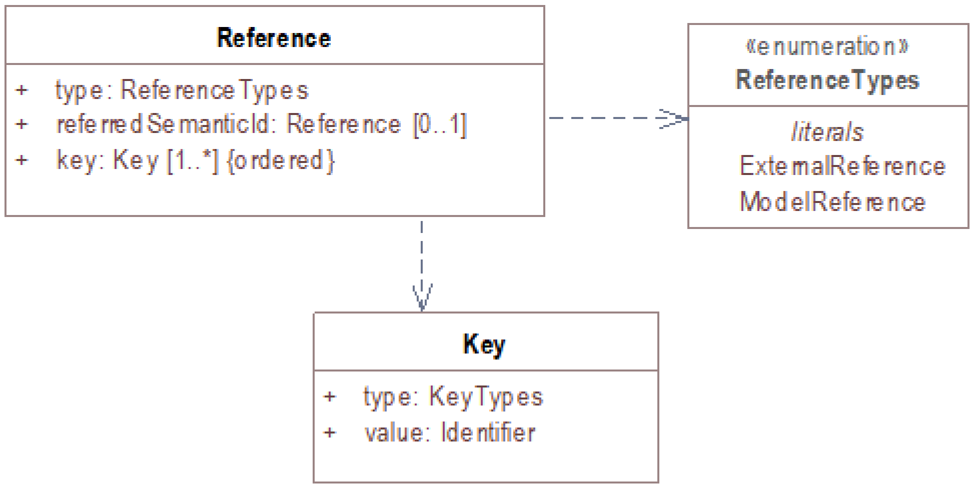


Figure 37. Metamodel of Reference

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
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

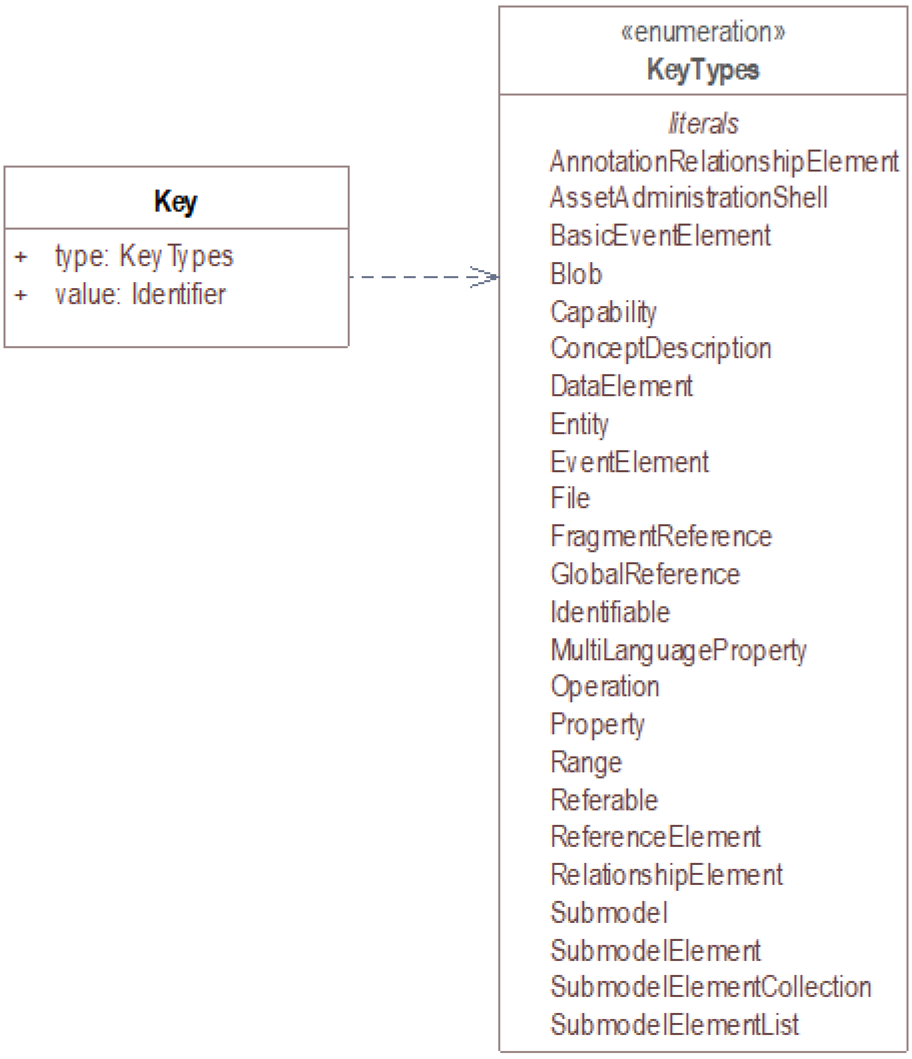


Figure 38. Metamodel of Keys

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

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

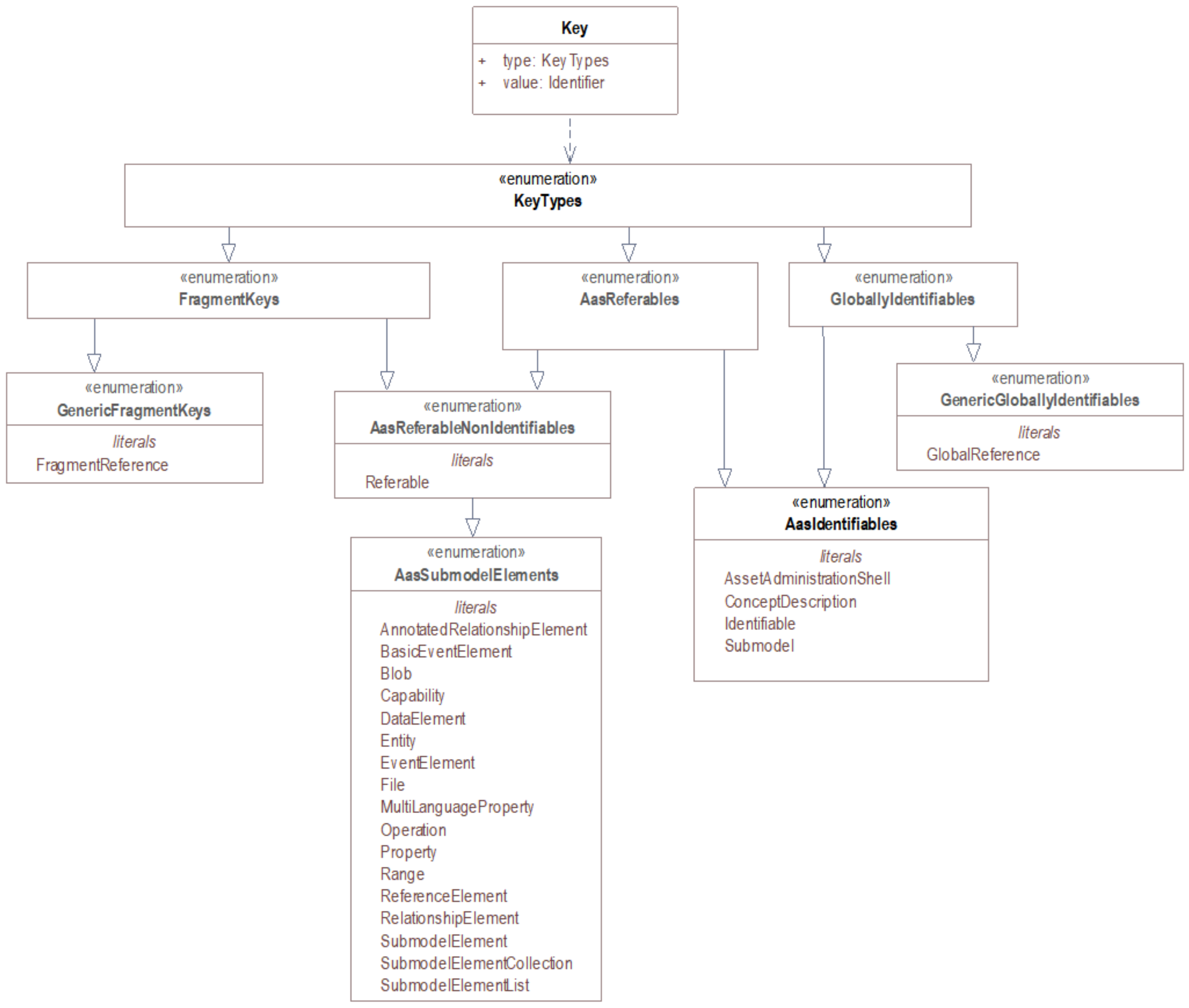


Figure 39. Logical Model for Keys of References (non-normative)

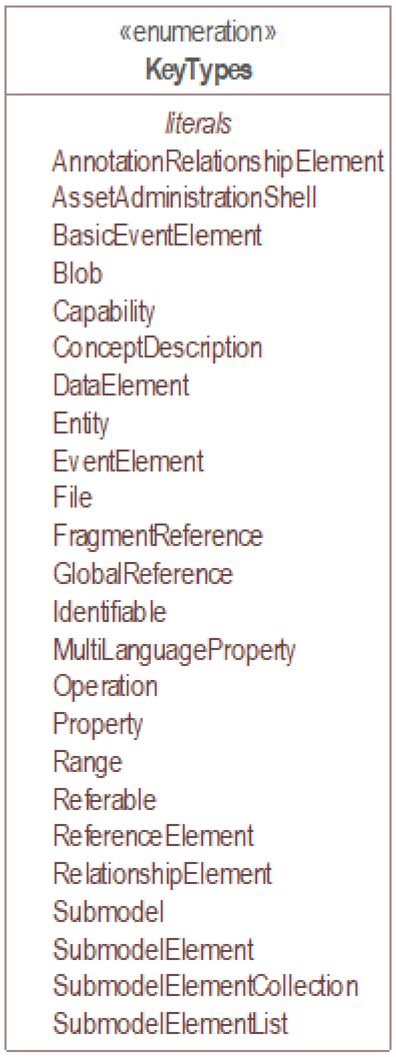


Figure 40. Metamodel of KeyTypes Enumeration

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.

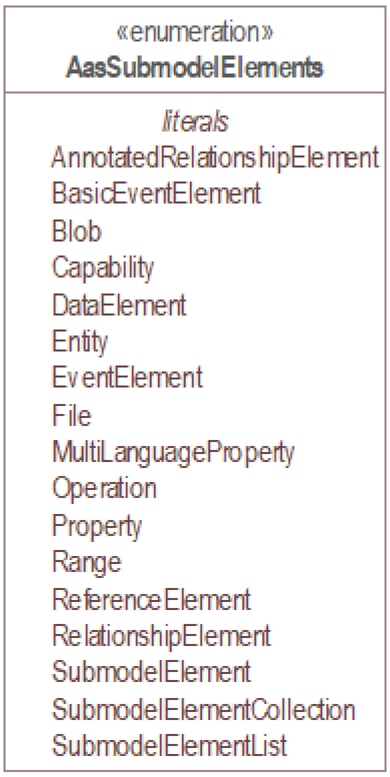


Figure 41. Metamodel of AasSubmodelElements Enumeration

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.

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)[1]. 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)[2].

See Clause 5.3.12.6 for constraints on types.

Table 2. 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"[3]
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 3. 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.[4]	application/pdf image/jpeg

Primitive	Definition	Value Examples
Identifier	*string* with max 2,000 and min 1 characters	https://cust/123456 0173-1#02-BAA120#008
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[5].

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)[6]. Additionally, the type "langString" with the semantics as defined in the Resource Description Framework (RDF)[7] is used. "langString" is a string value tagged with a language code.

Note 1: RDF[8] 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.

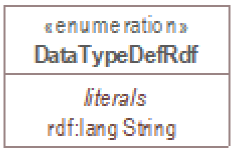


Figure 42. DefTypeDefRdf Enumeration

The enumeration is derived from Figure 54.

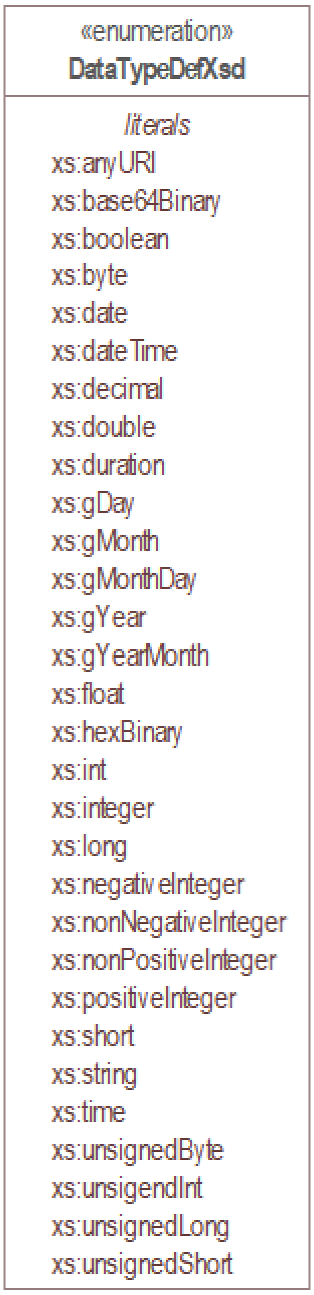


Figure 43. Data TypeDefXsd Enumeration

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 4. Data Types with Examples[9]

	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 12678967543233293879283742983742 9837429 +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"[10]
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 7345683746578364857368475638745

	Data Type	Value Range	Sample Values
	xs:nonNegativeInteger	Integer numbers ¬0	0 1 7345683746578364857368475 63
	xs:negativeInteger	Integer numbers <0	-1 • 234872638476283764827364872 63
	xs:nonPositiveInteger	Integer numbers ¬0	-1 0 -938458374985739874987989873
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, @end and ²de 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: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:float	see https://www.w3.org/TR/xmlschema11-2/#float
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
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
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

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

Literal	Explanation
rdf:langString	String with a language tag

RDF requires IETF BCP 47[11] 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.

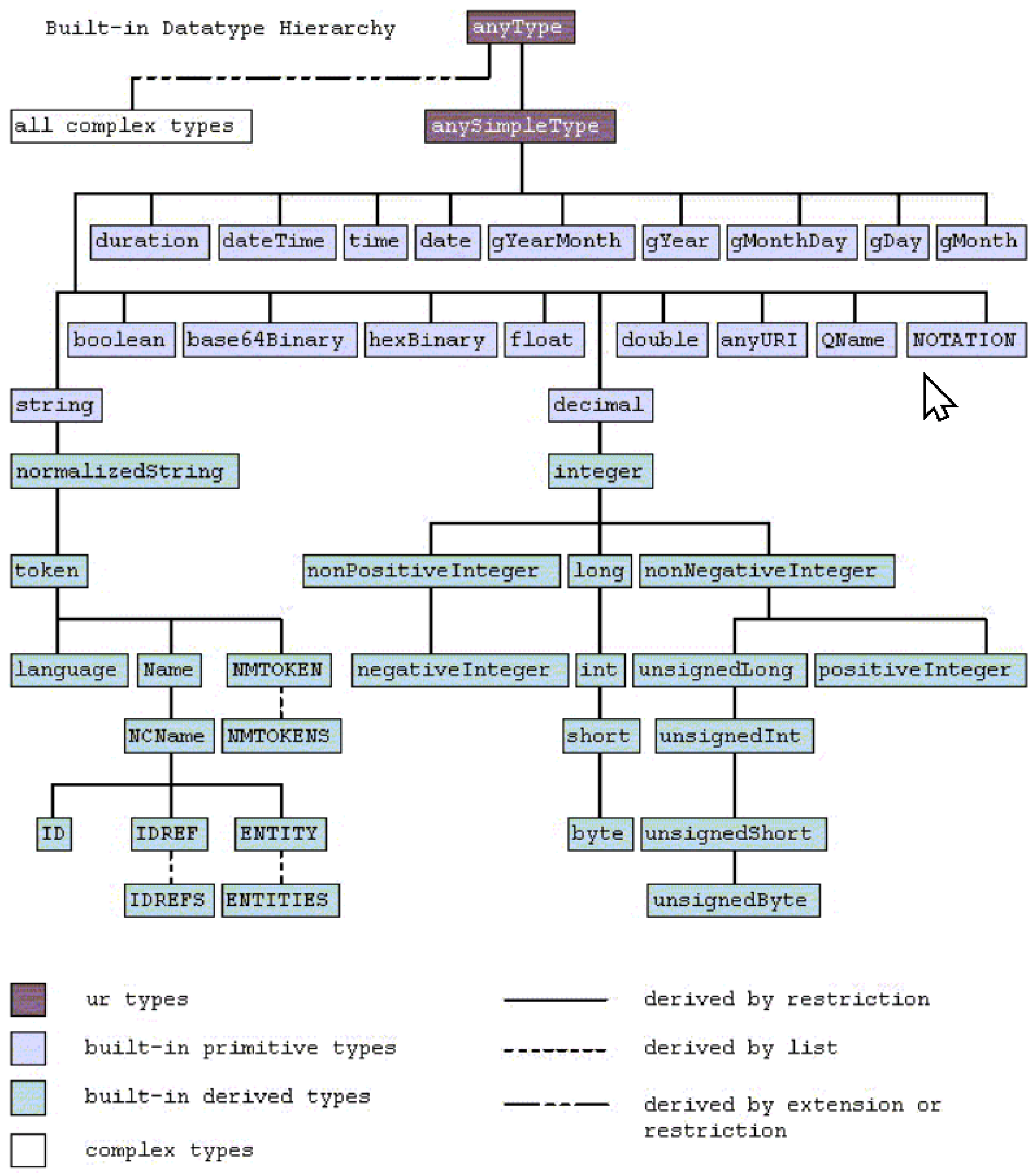


Figure 44. Built-In Types of XML Schema Definition 1.0 (XSD)[12]

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

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

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

[5] E.g. Property/valueType

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

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

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

[9] 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

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

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

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[3].

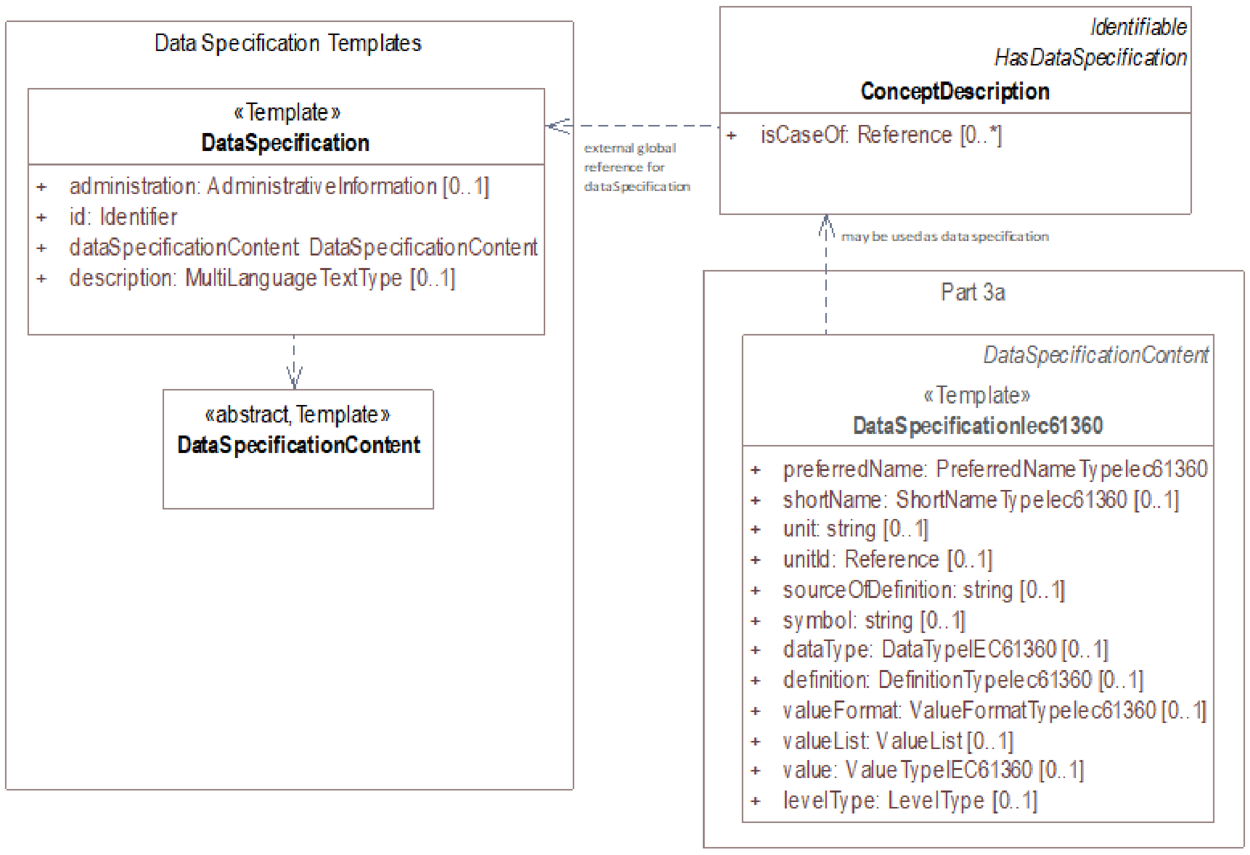
[3] https://www.w3.org/TR/xml/#charsets

Data Specification Templates (normative)

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[3] that, for example, can be used for concept descriptions for single properties.



Data Specification Templates
«Template»
DataSpecification
+ administration: AdministrativeInformation [0..1]
+ id: Identifier
+ dataSpecificationContent: DataSpecificationContent
+ description: MultiLanguageTextType [0..1]
«abstract, Template»
DataSpecificationContent
external global reference for dataSpecification
Identifiable
HasDataSpecification
ConceptDescription
+ isCaseOf: Reference [0..*]
may be used as data specification
Part 3a
DataSpecificationContent
«Template»
DataSpecificationIec61360
+ preferredName: PreferredNameTypeIec61360
+ shortName: ShortNameTypeIec61360 [0..1]
+ unit: string [0..1]
+ unitId: Reference [0..1]
+ sourceOfDefinition: string [0..1]
+ symbol: string [0..1]
+ dataType: DataTypeIEC61360 [0..1]
+ definition: DefinitionTypeIec61360 [0..1]
+ valueFormat: ValueFormatTypeIec61360 [0..1]
+ valueList: ValueList [0..1]
+ value: ValueTypeIEC61360 [0..1]
+ levelType: LevelType [0..1]

Figure 45. 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.

Data Specification Template

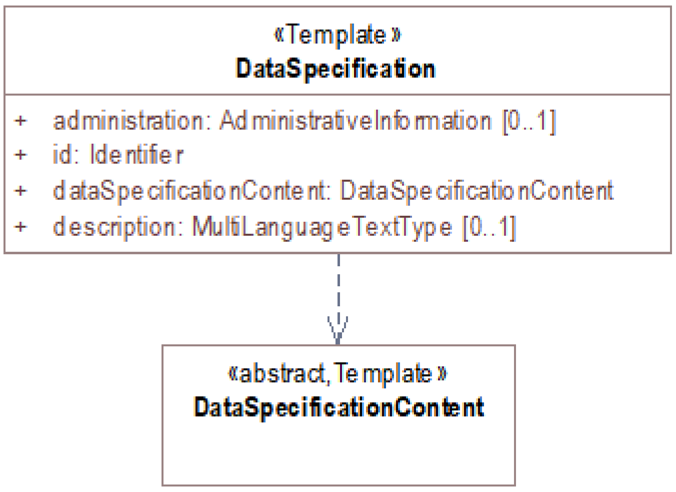

«Template»
DataSpecification
+ administration: AdministrativeInformation [0..1]
+ id: Identifier
+ dataSpecificationContent: DataSpecificationContent
+ description: MultiLanguageTextType [0..1]
«abstract,Template»
DataSpecificationContent

Figure 46. 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

Attribute	Explanation	Type	Card.
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.

[3] 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.