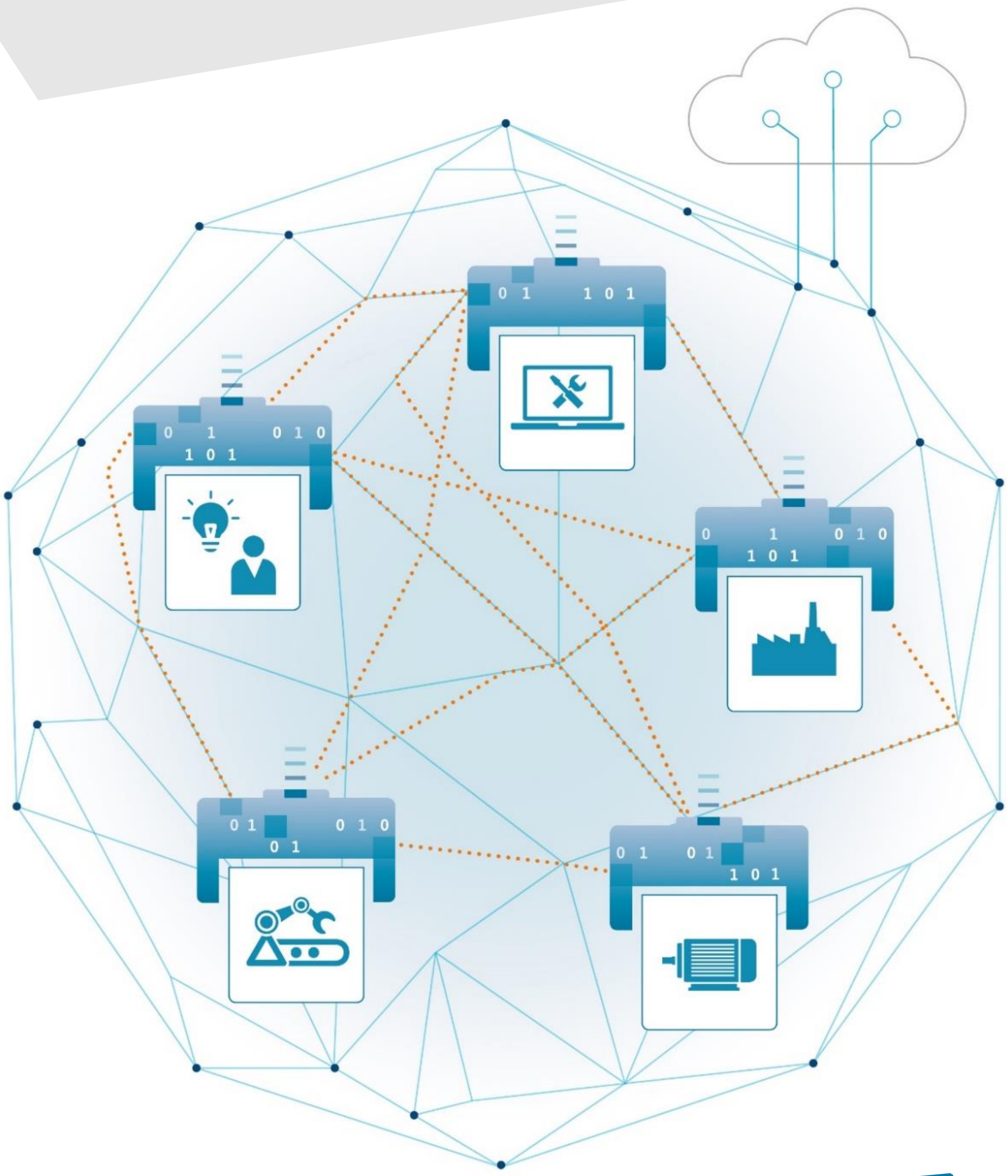
in cooperation with

SPECIFICATION

# Details of the Asset Administration Shell



**Part 2** – Interoperability at Runtime – Exchanging Information via Application Programming Interfaces (Version 1.0RC01)

# Imprint

**Publisher**

Federal Ministry for Economic Affairs

and Energy (BMWi)

Public Relations

10119 Berlin

www.bmwi.de

**Text and editing**

Plattform Industrie 4.0

Bertolt-Brecht-Platz 3

10117 Berlin

**Design and production**

The Plattform Industrie 4.0 secretariat, Berlin

**Status**

November 2020

**Illustrations**

Plattform Industrie 4.0; Anna Salari, designed by freepik (Title)

# Contents

# 1 Preamble

## 1.1 Editorial notes

This document was developed from December 2019 to November 2020 by the sub working groups “Asset Administration Shell” and “Infrastructure of the Asset Administration Shell” of the Platform Industrie 4.0 Working Group “Reference Architectures, Standards and Norms“.

This documents is part 2 of the document series “Details of the Asset Administration Shell” [1].

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

This specification is versioned using Semantic Versioning 2.0.0 and follows the semver specification [4].

## 1.2 Scope of this Document

This document specifies the interfaces as well as the APIs in selected technologies for the Asset Administration Shells and its submodels.

Note: In this first version of the document no technology specific mappings are yet included.

## 1.3 Structure of the Document

The technology neutral specification of the interfaces of the Asset Administration Shell can be found in Clause 4 to Clause 10. General topics are discussed in the Clause before, in Clause 3.

In the Annex the tables used to specify operations and interfaces are explained. Additionally, the UML notation used is presented.

## 1.4 Terms & Definitions

Forward notice
Definition of terms are only valid in a certain context. The current glossary applies to the context of this document. Definitions already defined in Part 1 ([3]) are only repeated if they are essential for this document.

**asset administration shell (AAS)**

standardized *digital representation* of the *asset*

Note 1 to entry: Asset Administration Shell and Administration Shell are used synonymously.
Note 2: Each administration shell can contain one or multiple sub models
Note 3: The administration shell can be passive, re-active, or pro-active
Note 4: The administration shell exists within one phase or across different phases of the lifecycle.
Note 5: Assets are part of an I4.0 component in an I4.0 system

→ [SOURCE: Glossary Industrie 4.0]

**interface**

defined connection point of a functional unit which can be connected to other functional units

Note 1: “Defined” means that the requirements and the assured properties of this connection point are described.

Note 2: The connection between the interfaces of function units is also called an interface.
Note 3: In an information system, the defined exchange of information takes place at this point.
Note 4: Interface places certain requirements on the connection that is to be made.
Note 5: Interface demands certain features.

[Source: Glossary Industrie 4.0
DUDEN (modified)
ISO/IEC 13066-1:2011(en), 2.15 (modified)
DIN EN 60870-5-6:2009-11 (modified)
DIN IEC 60625-1:1981-05 (modified)]

**operation**

executable realization of a function

| | |
|---|---|
| Note 1 to entry: | The term method is synonym to operation in the IT domain |
| Note 2 to entry: | an operation has a name and a list of parameters [ISO 19119:2005, 4.1.3] |

[SOURCE: Glossary Industrie 4.0 (work in progress)]

**service**

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

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

[SOURCE: Glossary Industrie 4.0]

**submodel**

model that is technically separated from another sub model and that is included in the *asset administration shell*

Note 1: Each submodel refers to a well-defined domain or subject matter. Submodels can become standardized and thus become submodel templates.
Note 2: Submodels can have different life cycles.
Note 3: The concept of template and instance applies to submodels.

→ [SOURCE: Glossary Industrie 4.0 (work in progress)]

**submodel element**

element suitable for the description and differentiation of assets

| | |
|---|---|
| Note 1 to entry: | extends the definition of properties |
| Note 2 to entry: | could describe operations, relationships, and files |

→ SOURCE: Glossary Industrie 4.0 (work in progress)]

## 1.5 Abbreviations

| Abbreviation | Description |
|---|---|
| AAS | Asset Administration Shell |
| AASX | Package file format for the AAS |
| AML | AutomationML |
| API | Application Programming Interface |
| BITKOM | Bundesverband Informationswirtschaft, Telekommunikation und neue Medien e. V. |
| BLOB | Binary Large Object |
| CDD | Common Data Dictionary |
| GUID | Globally unique identifier |
| I4.0 | Industrie 4.0 |
| ID | Identifier |
| IEC | International Electrotechnical Commission |
| IRDI | International Registration Data Identifier |
| ISO | International Organization for Standardization |
| JSON | JavaScript Object Notation |
| MIME | Multipurpose Internet Mail Extensions |
| OPC | Open Packaging Conventions (ECMA-376, ISO/IEC 29500-2) |
| OPC | Open Platform Communications |
| OPCF | OPC Foundation |
| OPC UA | OPC Unified Architecture |
| PDF | Portable Document Format |
| RAMI4.0 | Reference Architecture Model Industrie 4.0 |
| RDF | Resource Description Framework |
| REST | Representational State Transfer |
| RFC | Request for Comment |
| ROA | Ressource Oriented Architecture |
| SOA | Service Oriented Architecture |
| UML | Unified Modeling Language |
| URI, URL, URN | Uniform Resource Identifier, Locator, Name |

| Abbreviation | Description |
|---|---|
| VDI | Verein Deutscher Ingenieure e.V. |
| VDE | Verband der Elektrotechnik Elektronik Informationstechnik e. V. |
| VDMA | Verband Deutscher Maschinen- und Anlagenbau e.V. |
| W3C | World Wide Web Consortium |
| XML | eXtensible Markup Language |
| ZIP | archive file format that supports lossless data compression |
| ZVEI | Zentralverband Elektrotechnik- und Elektronikindustrie e. V. |

# 2 Introduction

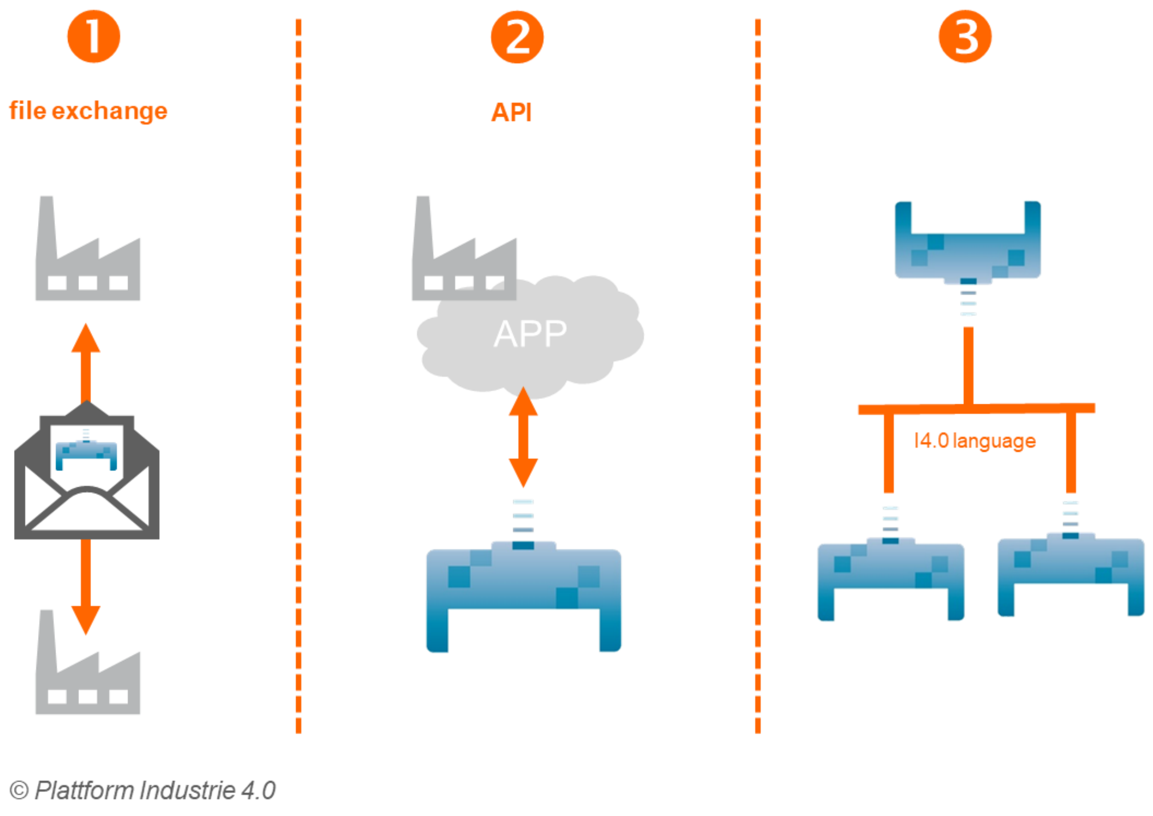
In this document APIs for enabling the access to the information an Asset Administration Shell provides are defined. The underlying information model is as defined in [2].

Since an API can be specified in different technologies like http/REST, MQTT and OPC UA the specification offers a technology neutral specification of the interfaces.

In this version of the specification this technology neutral specification of the interfaces is defined.

Whereas in part 1 of the specification series of the Asset Administration Shell ([2]) it was mainly file exchange that was considered it is the API that allows online access to information provided by the AAS that is subject of this specification (see Figure 1).

**Figure 1 Types of Information Exchange via Asset Administration Shells**



# 3 General

## 3.1 Services, Interfaces and Interface Operations

For this document the Industrie 4.0 Service illustrated in Figure 2 is used for a uniform understanding and naming. It basically distinguishes between associated concepts on several levels (from left to right):

- technology-neutral level: concepts that are independent from selected technologies.
- technology-specific level: concepts that are instantiated for a given technology and/or architectural style (e.g. http/REST, OPC UA, MQTT)
- implementation level:  concepts that are related to an implementation architecture that comprises one or more technologies (e. g. C#, C++, Java, Python)
- runtime level: concepts that are related to identifiable components in an operational Industrie 4.0 system.

The concepts that are dealt with in this document are those of the technology-neutral and technology-specific level. However, in order to avoid terminological and conceptual misunderstandings, the whole Industrie 4.0 service model is provided here.

The technology-neutral level comprises the following concepts:

- Service: A service describes a demarcated scope of functionality (including its informational and non-functional aspects), which is offered by an entity or organization via interfaces.
- Interface: This is the most important concept as it is understood to be the unit of reusability across services and the unit of standardization when being mapped to application programming interfaces (API) in the technology-specific level. One interface may be mapped to several APIs depending on the technology and architectural style being used, e.g. http/REST or OPC UA, whereby these API mappings also need to be standardized for the sake of interoperability.
- Interface-Operation: An interface is specified by means of interface operations according to specified interaction policies and patterns.

The technology-specific level comprises the following concepts:

- Service Specification: specification of a service according to the notation, architectural style and constraints of a selected technology. Among others, it comprises and refers to the list of APIs that forms this service specification. These may be I4.0-defined standard APIs but also other, proprietary APIs.
    - Note: Such a technology-specific service specification may but not need to be derived from the “service” described in the technology-neutral form. It is up to the system architect and service engineer to tailor the technology-specific service according to the needs of the use cases to be supported.
- API (Application programming Interface): Specification of the set of operations and events that forms an API in a selected technology. It is derived from the interface description on the technology-neutral level. Hence, if there are several selected technologies, one interface may be mapped to several APIs.
- API-Operation: specification of the operations (procedures) that may be called through an API. It is derived from the interface operation description on the technology-neutral level. Hence, if there are several selected technologies, one interface operation may be mapped to several API-operations.

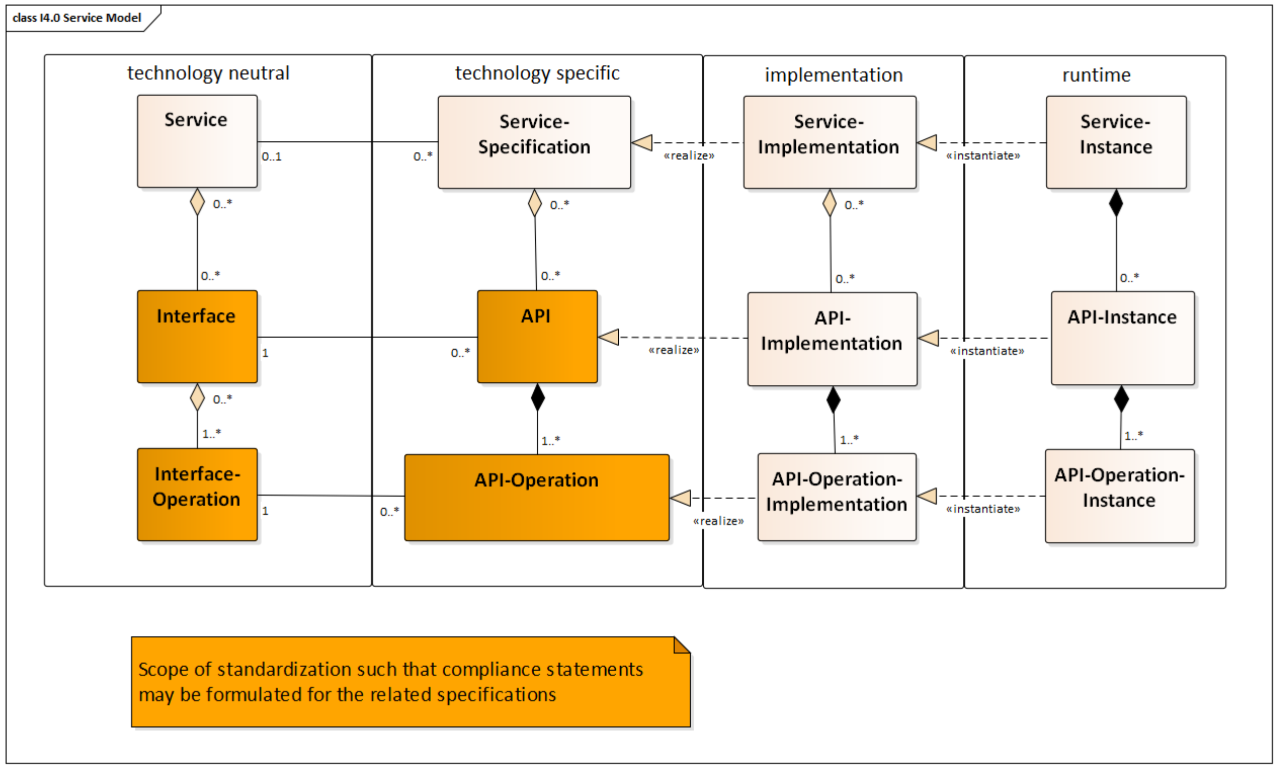
The implementation level comprises the following concepts:

- Service-Implementation: service realized in a selected implementation language following the specification in the Service Specification description on the technology-specific level.
- API-Implementation: set of operations realized in a selected implementation language following the specification in the API description on the technology-specific level.
- API-Operation-Implementation: concrete realization of an operation in a selected implementation language following the specification in the API-Operation description on the technology-specific level.

The runtime level comprises the following concepts:

- Service-Instance: instance of a Service-Implementation including its API-Instances for the communication. Additionally, it has an identifier to be identifiable within a given context.
- API-Instance: instance of an API-Implementation which has an endpoint to get the information about this instance and the related operations.
- API-Operation-Instance: instance of an API-Operation-Implementation which has an endpoint to get invoked.

**Figure 2 Services, Interfaces & APIs and Operations**



One important take-away message from the Industrie 4.0 Service Model is that it is the level of the interface (mapped to technology-specific APIs) that

- provides the unit of reusability,
- is the foundation for interoperable services, and
- provides the reference unit for compliance statements.

Therefore, in this document in Clause 3.5 the Interfaces and Operations which are needed for interaction regarding the elements of the Asset Administration Shell meta model are defined. Mappings to specific technologies are not part of this document yet but will be part in a following version.

## 3.2 Design Principles

The operations of the interfaces follow a resource-oriented approach which is close to general REST principles but not as strict in every situation. The approach consists of the three main agreements:

- Stateless
  The API is stateless. Each operation is independent. After each operation the server is always consistent.

- Resources (Nouns)
  Each resource is a clearly defined noun. This means that it has a specific name and the relation to other nouns is defined. The nouns and the relationships between them are taken from the list of referable objects of "Details of the Asset Administration Shell Part1" and their relationships. Additionally, there will be a list of resources defined in Clause 9.
- Methods (Verbs)
  A small set of standard methods which are GET, GETALL, PUT and DELETE is used to describe the semantic of the most common operations. There are only a few exceptions for methods which are high effort to do by standard methods or for situations where the standard methods do not fit (e.g. SET, REMOVE).

The standard methods are:

- GET
  A GET returns a single resource based on the resource identifier which is the identifier ([2]) for identifiables and the idShort for referables.
- GETALL
  Returns a list of resources based on optional available parameters such as filters.
- PUT
  Creates a resource if it does not exist or updates an existing one. The identifier of the resource is not created by the server, it will be part of the resource description. This is necessary because the id of identifiables is globally unique and should be the identifier for the object in every system. This leads to the point that the creation of an Identifiable is idempotent. There shall never be more than one Identifiable with the same ID in one System. If you try for example to put the same AAS object twice it will not create two AAS resources.
- DELETE
  Deletes a resource based on a given identifier.

## 3.3 Semantic References for Operations

The Operations of this document need unique identifiers to reach a common understanding and allow all involved parties to reference the same things. These identifiers need to be globally unique and understandable by the community and implementing systems. Furthermore, the identifiers need to support a versioning scheme for future updates and extensions of the metamodel. The identifiers defined in this document are reused in related resources, for instance protocol bindings of the presented operations or in self-descriptions of implementing services.

Internationalized Resource Identifiers (IRIs), Uniform Resource Identifiers (URIs) [6] in particular, and the requirements of DIN SPEC 91406, serve as the basic format. Further design decisions include 'https' as the URI scheme, and the controlled domain name 'admin-shell.io' as the chosen authority. Both decisions guarantee the interoperability of the identifiers and their durability, as URIs in general are well-known and proven and the mentioned domain is controlled and served through the Plattform Industrie 4.0. All identifiers included in the 'admin-shell.io' domain are further described in a lightweight catalogue in the form of markdown documents and continuously maintained and updated[1]. The catalogue itself is further structured in several sub-namespaces specified by the first path parameter. All URIs of this document reflect entities of the core metamodel, which are contained in the sub-namespace identified with the '/aas' path.

The thereby described identifiers appear mainly in the semanticId field of every class and operation. They are needed as the class name is not necessarily constant over time. The respective semanticIds however guarantee the

[1] https://github.com/admin-shell-io/id

unique and certain relation between a reference and the referenced class or operation. The URIs ids is as follows (compare to Clause Semantic Identifiers for Metamodel and Data Specifications in Part 1 [2]).

Note: Version information is explicitly included in each identifier.
Note: Even though the usage of the 'https' scheme might indicate URLs, all identifiers are regarded as URIs look ups and dereferencing them cannot be expected.

The following grammar is used to create valid identifiers:

```
<Identifier> ::= <Namespace>'/aas/API/'<idShortPath>'/'<Version>

<Namespace>  ::= 'https://admin-shell.io/'

<idShortPath>      ::= <idShort>('/'<idShortPath>)?

<idShort>    ::= <Character>+

<Version>    ::= <Digit>+'/'<Digit>+['/'<Character>+]

<Digit>      ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Character>  ::= an unreserved character permitted by DIN SPEC 91406


?      ::= zero or one

+      ::= one or more
```

Rule: To reference a single operation the *operationName* is added in field <idShortPath>.

Examples for valid identifiers:

— https://admin-shell.io/aas/API/GetSubmodel/1/23
— https://admin-shell.io/aas/API/GetAllSubmodelElements/1/0/RC01

Examples:

— https://admin-shell.io/aas/API/GetSubmodel/1/0/RC01

Examples for invalid identifiers:

— http://admin-shell.io/API/GetSubmodel/1/0
  The scheme is different to 'https', and the 'aas' path segment is missing
— https://admin-shell.io/aas/API/GetSubmodel
  No version information is included.
— https://admin-shell.io/aas/API/GetSubmodel/1/0#0173-%20ABC#001
  The URI includes DIN SPEC 91406-reserved (#) and not permitted (%) characters.

## 3.4 References and Keys

In part 1 ([1]) of the series Asset Administration Shell in Detail the concept of Reference is introduced.

When defining interfaces, we distinguish between relative references and absolute references.

Absolute references require a global unique id as starting point of the reference to be resolvable. In this case the type “Reference” is used.

Relative references do not start with a global unique id but assume that the context is given and unique. Then the key list only contains keys with Key/idType== IdShort and a Key/type that references a non-identifiable referable (e.g. a Property, a Range, a RelationshipElement etc.). For relative references the data type “Key[<cardinality>] is used, e.g. Key[1..*].

## 3.5 Special Parameters

Special Parameters used for consistency throughout the document are described in the following table.

| **Parameter** | **Description** |
|---|---|
| Key[] path | IdShort-Path via relative Reference/Keys to a submodel element |
| OperationHandle | The returned handle of an operation’s asynchronous invocation used to request the current state of the operation’s execution |
| OperationResult | The returned result of an operation’s invocation |
| OutputModifier | Determines the result format filtering of the response |
| SearchOptions | Determines the search options of the search space, e.g. the depth |
| SerializationFormat | Determines the format of serialization, i.e. JSON, XML, RDF, AML, etc. |
| ShellDescriptor | Object containing the Asset Administration Shell’s identification and endpoint information |
| SubmodelDescriptor | Object containing the Submodel’s identification and endpoint information |
| Key | The key name of the specific asset identifier (IdentifierKeyValuePair/key) or the predefined key “*globalAssetId*” that would refer to the *AssetInformation/globalAssetId*. |
| KeyType | Key type.<br><br>Mandatory for global asset id (corresponds to *Identifier/idType*) |
| semanticId | Identifier of the semantic definition |

# 4 Interfaces Asset Administration Shell

## 4.1 General

These interfaces allow to access the elements of administration shells or submodels.

Sometimes, these kinds of services are also classified as contextualization and classification services.

## 4.2 Asset Administration Shell Interface and Operations

### 4.2.1 Interface Asset Administration Shell

| Interface: Asset Administration Shell | |
|---|---|
| Operation Name | Description |
| GetAssetAdministrationShell | Returns the Asset Administration Shell |
| PutAssetAdministrationShell | Updates the current Asset Administration Shell |
| PatchAssetAdministrationShell | Adds or updates additional elements to the Asset Administration Shell |
| PutSubmodelReference | Creates or updates a Submodel Reference at the Asset Administration Shell |
| RemoveSubmodelReference | Removes a specific Submodel Reference from the Asset Administration Shell |

### 4.2.2 Operation GetAssetAdministrationShell

| | | |
|---|---|---|
| Operation Name | GetAssetAdministrationShell | |
| Explanation | Returns the Asset Administration Shell | |
| semanticId | https://admin-shell.io/aas/API/GetAssetAdministrationShell/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShell | Requested Asset Administration Shell |

### 4.2.3 Operation PutAssetAdministrationShell

| | | |
|---|---|---|
| Operation Name | PutAssetAdministrationShell | |
| Explanation | Updates the Asset Administration Shell | |
| semanticId | https://admin-shell.io/aas/API/PutAssetAdministrationShell/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| aas | AssetAdministrationShell | Asset Administration Shell object |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShell | Updated Asset Administration Shell |

### 4.2.4 Operation PatchAssetAdministrationShell

| | | |
|---|---|---|
| Operation Name | PatchAssetAdministrationShell | |
| Explanation | Adds or updates additional elements to the Asset Administration Shell | |
| semanticId | https://admin-shell.io/aas/API/PatchAssetAdministrationShell/1/0/RC01 | |
| | | |
| Name | Type | Description |
| Input Parameter | | |
| aas | AssetAdministrationShell | Asset Administration Shell object |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShell | Updated Asset Administration Shell |

### 4.2.5 Operation PutSubmodelReference

| | | |
|---|---|---|
| Operation Name | PutSubmodelReference | |
| Explanation | Creates or updates a Submodel Reference at the Asset Administration Shell | |
| semanticId | https://admin-shell.io/aas/API/PutSubmodelReference/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| submodelRef | Reference | Reference to the Submodel |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | Reference | Created Submodel Reference |

### 4.2.6 Operation RemoveSubmodelReference

| | | |
|---|---|---|
| Operation Name | RemoveSubmodelReference | |
| Explanation | Removes the Submodel Reference from the Asset Administration Shell | |
| semanticId | https://admin-shell.io/aas/API/RemoveSubmodelReference/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| submodelRef | Reference | Reference to the Submodel |
| Output Parameter | | |
| statusCode | StatusCode | Status code |

## 4.3 Submodel Interface and Operations

### 4.3.1 Interface Submodel

| Interface: Submodel | |
|---|---|
| Operation Name | Description |
| GetSubmodel | Returns the Submodel |
| GetAllSubmodelElements | Returns all submodel elements including their hierarchy |
| GetAllSubmodelElementsByParentPathAndSemanticId | Returns all submodel elements of the Submodel or of the parent submodel element with a specific Semantic-Id |
| GetAllSubmodelElementsBySemanticId | Returns all submodel elements from a Submodel with a specific Semantic-Id |
| GetSubmodelElementByPath | Returns a specific submodel element from the Submodel at a specified path |
| PutSubmodelElementByPath | Creates a new or updates an existing submodel element at a specified path within the submodel elements hierarchy |
| SetSubmodelElementValueByPath | Sets the value of the submodel element at a specified path according to the protocol-specific RAW-value payload |
| DeleteSubmodelElementByPath | Deletes a submodel element at a specified path within submodel elements hierarchy |
| InvokeOperationSync | Synchronously invokes an Operation at a specified path with a client timeout in ms |
| InvokeOperationAsync | Asynchronously invokes an Operation at a specified path with a client timeout in ms |
| GetOperationAsyncResult | Returns the OperationResult of an asynchronously invoked operation |

### 4.3.2 Operation GetSubmodel

| Operation Name | GetSubmodel | |
|---|---|---|
| Explanation | Returns the Submodel | |
| semanticId | https://admin-shell.io/aas/API/GetSubmodel/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | Submodel | Requested Submodel |

### 4.3.3 Operation GetAllSubmodelElements

| Operation Name | GetAllSubmodelElements | |
|---|---|---|
| Explanation | Returns all submodel elements including their hierarchy | |
| semanticId | https://admin-shell.io/aas/API/GetAllSubmodelElements/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| searchOptions | SearchOptions | Determines the search options of the search space, e.g. the depth |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | SubmodelElement[0..*] | Requested submodel elements |

### 4.3.4 Operation GetAllSubmodelElementsByParentPathAndSemanticId

| Operation Name | GetAllSubmodelElementsByParentPathAndSemanticId | |
|---|---|---|
| Explanation | Returns all submodel elements of the Submodel or of the parent submodel element with a specific Semantic-Id | |
| semanticId | https://admin-shell.io/aas/API/GetAllSubmodelElementsByParentPathAndSemanticId/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| parentPath | Key[1..*] | IdShort-Path via relative Reference/Keys to a submodel element |
| semanticId | Reference | Identifier of the semantic definition |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | SubmodelElement[0..*] | Requested Submodel Elements |

### 4.3.5 Operation GetAllSubmodelElementsBySemanticId

| Operation Name | GetAllSubmodelElementsBySemanticId | |
|---|---|---|
| Explanation | Returns all submodel elements from a Submodel with a specific Semantic-Id | |
| semanticId | https://admin-shell.io/aas/API/GetAllSubmodelElementsBySemanticId/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| semanticId | Reference | Identifier of the semantic definition |
| searchOptions | SearchOptions | Determines the search options of the search space, e.g. the depth |
| outputModifier | OutputModifier | Determines the result format filtering of the response |

| Operation Name | GetAllSubmodelElementsBySemanticId | |
|---|---|---|
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | SubmodelElement[0..*] | Requested Submodel Elements |

### 4.3.6 Operation GetSubmodelElementByPath

| Operation Name | GetSubmodelElementByPath | |
|---|---|---|
| Explanation | Returns a specific submodel element from the Submodel at a specified path | |
| semanticId | https://admin-shell.io/aas/API/GetSubmodelElementByPath/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| path | Key[1..*] | IdShort-Path via relative Reference/Keys to a submodel element |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | SubmodelElement | Requested submodel element |

### 4.3.7 Operation PutSubmodelElementByPath

| Operation Name | PutSubmodelElementByPath | |
|---|---|---|
| Explanation | Creates a new or updates an existing submodel element at a specified path within submodel elements hierarchy | |
| semanticId | https://admin-shell.io/aas/API/PutSubmodelElementByPath/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |

| Operation Name | PutSubmodelElementByPath | |
|---|---|---|
| path | Key[1..*] | IdShort-Path via relative Reference/Keys to a submodel element |
| submodelElement | SubmodelElement | Submodel element object |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | SubmodelElement | Created or updated submodel element |

### 4.3.8 Operation SetSubmodelElementValueByPath

| Operation Name | SetSubmodelElementValueByPath | |
|---|---|---|
| Explanation | Sets the value of the submodel element at a specified path according to the protocol-specific RAW-value payload | |
| semanticId | https://admin-shell.io/aas/API/SetSubmodelElementValueByPath/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| path | Key[1..*] | IdShort-Path via relative Reference/Keys to a submodel element |
| payload | anyType | The new value of the submodel element to be set |
| Output Parameter | | |
| statusCode | StatusCode | Status code |

### 4.3.9 Operation DeleteSubmodelElementByPath

| Operation Name | DeleteSubmodelElementByPath |
|---|---|
| Explanation | Deletes a submodel element at a specified path within the submodel elements hierarchy |
| semanticId | https://admin-shell.io/aas/API/DeleteSubmodelElementByPath/1/0/RC01 |

| Operation Name | DeleteSubmodelElementByPath | |
|---|---|---|
| Name | Type | Description |
| Input Parameter | | |
| path | Key[1..*] | IdShort-Path via relative Reference/Keys to a submodel element |
| Output Parameter | | |
| statusCode | StatusCode | Status code |

#### 4.3.10 Operation InvokeOperationSync

| Operation Name | InvokeOperationSync | |
|---|---|---|
| Explanation | Synchronously invokes an Operation at a specified path with a client timeout in ms | |
| semanticId | https://admin-shell.io/aas/API/InvokeOperationSync/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| path | Key[1..*] | IdShort-Path via relative Reference/Keys to a submodel element, in this case an operation |
| inputArgument | OperationVariable[0..*] | Input argument |
| inoutputArgument | OperationVariable[0..*] | Inoutput argument |
| timeout | nonNegativeInteger | Client timeout |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | OperationResult | Operation Result |

### 4.3.11 Operation InvokeOperationAsync

| Operation Name | InvokeOperationAsync | |
|---|---|---|
| Explanation | Asynchronously invokes an Operation at a specified path with a client timeout in ms | |
| semanticId | https://admin-shell.io/aas/API/InvokeOperationAsync/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| path | Key[1..*] | IdShort-Path via relative Reference/Keys to a submodel element, in this case an operation |
| inputArgument | OperationVariable[0..*] | Input argument |
| inoutputArgument | OperationVariable[0..*] | Inoutput argument |
| timeout | nonNegativeInteger | Client timeout |
| requestId | string | Client request id |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | OperationHandle | The returned handle of an operation's asynchronous invocation used to request the current state of the operation's execution |

### 4.3.12 Operation GetOperationAsyncResult

| Operation Name | GetOperationAsyncResult | |
|---|---|---|
| Explanation | Returns the OperationResult of an asynchronously invoked operation | |
| semanticId | https://admin-shell.io/aas/API/GetOperationAsnycResult/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| operationHandle | OperationHandle | The returned handle of an operation's asynchronous invocation used to request the current state of the operation's execution |

| Output Parameter | | |
|---|---|---|
| statusCode | StatusCode | Status code |
| payload | OperationResult | Operation Result |

## 4.4 Asset Administration Shell Serialization Interface and Operations

### 4.4.1 Interface Asset Administration Shell Serialization

| Interface: Asset Administration Shell Serialization | |
|---|---|
| Operation Name | Description |
| GetAASX | Returns an appropriate AASX-Package with the respective Asset Administration Shells |
| GetSerializationByIds | Returns an appropriate serialization based on the specified format (see SerializationFormat). |

### 4.4.2 Operation GetAASX

| Operation Name | GetAASX | |
|---|---|---|
| Explanation | Returns a file .aasx following the AASX-Package format containing the requested Asset Administration Shells<br><br>The package may contain more content (e.g. Asset Administration Shells, Submodels, etc.)<br><br>References to Assets, Submodels and Concept Descriptions are only resolved if the referenced content is available to the Asset Administration Shell. | |
| semanticId | https://admin-shell.io/aas/API/GetAASX/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| aasIds | Identifier[1..*] | The unique ids of the Asset Administration Shells to be contained in the AASX-Package |
| Output Parameter | | |
| statusCode | StatusCode | Status code |

| Operation Name | GetAASX | |
|---|---|---|
| payload | *.aasx package file* | .aasx file, Asset Administration Shell Exchange Package (AASX) |

### 4.4.3 Operation GetSerializationByIds

| Operation Name | GetSerializationByIds | |
|---|---|---|
| Explanation | Returns an appropriate serialization based on the specified format (see SerializationFormat).<br>References to Assets, Submodels and Concept Descriptions are only resolved if the referenced content is available to the Asset Administration Shell. | |
| semanticId | https://admin-shell.io/aas/API/GetSerializationByIds/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| aasId | Identifier[1..*] | The unique id of the Asset Administration Shells to be contained in the serialization |
| serializationFormat | SerializationFormat | Determines the format of serialization, i.e. JSON, XML, RDF, AML, etc. |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShell[0...*]+ | Serialization of requested Asset Administration Shells in specified serialization format as byte string |

# 5 Interfaces Registration and Lookup

## 5.1 General

These interfaces allow to register and unregister descriptors of administration shells or submodels. These descriptors contain the required information that is needed to access the interfaces (Interfaces described in Clause 3.5) of the corresponding element. This required information includes the endpoint in the dedicated environment.

Lookup interfaces provide access to the registered descriptors by identifiers (Asset Administration Shell and Submodel ID). These Identifiers may be discovered by Interfaces described in Clause 7.

Sometimes, these kinds of services are also classified as management services.

## 5.2 Asset Administration Shell Registry Interface and Operations

### 5.2.1 Interface Asset Administration Shell Registry

| Interface: Asset Administration Shell Registry | |
|---|---|
| Operation Name | Description |
| GetAllAssetAdministrationShellDescriptors | Returns all Asset Administration Shell Descriptors |
| GetAssetAdministrationShellDescriptorById | Returns a specific Asset Administration Shell Descriptor |
| PutAssetAdministrationShellDescriptor | Creates a new or updates an existing Asset Administration Shell Descriptor |
| DeleteAssetAdministrationShellDescriptorById | Deletes an Asset Administration Shell Descriptor |

### 5.2.2 Operation GetAllAssetAdministrationShellDescriptors

| | | |
|---|---|---|
| Operation Name | GetAllAssetAdministrationShellDescriptors | |
| Explanation | Returns all Asset Administration Shell Descriptors | |
| semanticId | https://admin-shell.io/aas/API/GetAllAssetAdministrationShellDescriptors/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShellDescriptor[0..*] | Requested Asset Administration Shell Descriptors |

### 5.2.3 Operation GetAssetAdministrationShellDescriptorById

| | | |
|---|---|---|
| Operation Name | GetAssetAdministrationShellDescriptorById | |
| Explanation | Returns a specific Asset Administration Shell Descriptor | |
| semanticId | https://admin-shell.io/aas/API/GetAssetAdministrationShellDescriptorById/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| Id | Identifier | The Asset Administration Shell’s unique id |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShellDescriptor | Requested Asset Administration Shell Descriptor |

### 5.2.4 Operation PutAssetAdministrationShellDescriptor

| | | |
|---|---|---|
| Operation Name | PutAssetAdministrationShellDescriptor | |
| Explanation | Creates a new or updates an existing Asset Administration Shell Descriptor | |
| semanticId | https://admin-shell.io/aas/API/PutAssetAdministrationShellDescriptor/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| shellDescriptor | AssetAdministrationShellDescriptor | Object containing the Asset Administration Shell’s identification and endpoint information |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShellDescriptor | Created or updated Asset Administration Shell Descriptor |

### 5.2.5 Operation DeleteAssetAdministrationShellDescriptorById

| Operation Name | DeleteAssetAdministrationShellDescriptorById | |
|---|---|---|
| Explanation | Deletes an Asset Administration Shell Descriptor | |
| semanticId | https://admin-shell.io/aas/API/DeleteAssetAdministrationShellDescriptorById/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| id | Identifer | The Asset Administration Shell’s unique id |
| Output Parameter | | |
| statusCode | StatusCode | Status code |

## 5.3 Submodel Registry Interface and Operations

### 5.3.1 Interface Submodel Registry

| Interface:Submodel Registry | |
|---|---|
| Operation Name | Description |
| GetAllSubmodelDescriptors | Returns all submodel descriptors |
| GetSubmodelDescriptorById | Returns a specific submodel descriptor |
| PutSubmodelDescriptor | Creates a new or updates an existing submodel descriptor |
| DeleteSubmodelDescriptorById | Deletes a submodel descriptor |

### 5.3.2 Operation GetAllSubmodelDescriptors

| Operation Name | GetAllSubmodelDescriptors | |
|---|---|---|
| Explanation | Returns all submodel descriptors | |
| semanticId | https://admin-shell.io/aas/API/GetAllSubmodelDescriptors/1/0/RC01 | |
| Name | Type | Description |

| Operation Name | GetAllSubmodelDescriptors | |
|---|---|---|
| Input Parameter | | |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | SubmodelDescriptor[0..*] | Requested submodel descriptors |

### 5.3.3 Operation GetSubmodelDescriptorById

| Operation Name | GetSubmodelDescriptorById | |
|---|---|---|
| Explanation | Returns a specific Submodel Descriptor | |
| semanticId | https://admin-shell.io/aas/API/GetSubmodelDescriptorById/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| Id | Identifier | The Submodel's unique id |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | SubmodelDescriptor | Requested submodel descriptor |

### 5.3.4 Operation PutSubmodelDescriptor

| Operation Name | PutSubmodelDescriptor | |
|---|---|---|
| Explanation | Creates a new or updates an existing submodel descriptor | |
| semanticId | https://admin-shell.io/aas/API/PutSubmodelDescriptor/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| submodel Descriptor | SubmodelDescriptor | Object containing the Submodel's identification and endpoint information |

| Operation Name | PutSubmodelDescriptor | |
|---|---|---|
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | SubmodelDescriptor | Created or updated submodel descriptor |

### 5.3.5 Operation DeleteSubmodelDescriptorById

| Operation Name | DeleteSubmodelDescriptorById | |
|---|---|---|
| Explanation | Deletes a Submodel Descriptor | |
| semanticId | https://admin-shell.io/aas/API/DeleteSubmodelDescriptorById/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| id | Identifier | The Submodel’s unique id |
| Output Parameter | | |
| statusCode | StatusCode | Status code |

# 6 Interfaces Repository

## 6.1 General

These interfaces allow to manage Asset Administration Shell and submodel entities and provide access to the data of these elements through interfaces described in Clause 3.5. A repository can host multiple entities. These entities can be stored in individual repositories of a decentral system. The endpoints of the entities managed by one repository shall be resolved by subsequent calls to discover (Clause 7) and lookup (Clause 5) interfaces to such decentralized systems.

Sometimes, these kinds of services are also classified as Asset Administration Shell management services.

The interfaces that provide access to the entities (administration shells, submodels) themselves are convenience interfaces that provide access in a system where the services are managed by central repositories.

## 6.2 Asset Administration Shell Repository Interface and Operations

### 6.2.1 Interface Asset Administration Shell Repository

| Interface: Asset Administration Shell Registry | |
|---|---|
| Operation Name | Description |
| GetAllAssetAdministrationShells | Returns all Asset Administration Shells |
| GetAllAssetAdministrationShellsById | Returns a specific Asset Administration Shell |
| GetAllAssetAdministrationShellsByAssetId | Returns all Asset Administration Shell that are linked to a globally unique asset identifier or to specific asset ids. |
| GetAllAssetAdministrationShellsByIdShort | Returns all Asset Administration Shells with a specific idShort |
| PutAssetAdministrationShell | Creates a new or updates an existing Asset Administration Shell |
| DeleteAssetAdministrationShellById | Deletes an Asset Administration Shell |

### 6.2.2 Operation GetAllAssetAdministrationShells

| Operation Name | GetAllAssetAdministrationShells | |
|---|---|---|
| Explanation | Returns all Asset Administration Shells | |
| semanticId | https://admin-shell.io/aas/API/GetAllAssetAdministrationShells/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| outputModifier | OutputModifier | Determines the result format filtering of the response |

| Operation Name | GetAllAssetAdministrationShells | |
|---|---|---|
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShell[0..*] | Requested Asset Administration Shells |

### 6.2.3 Operation GetAssetAdministrationShellById

| Operation Name | GetAssetAdministrationShellsById | |
|---|---|---|
| Explanation | Returns a specific Asset Administration Shell | |
| semanticId | https://admin-shell.io/aas/API/GetAssetAdministrationShellsById/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| id | Identifier | The Asset Administration Shell's unique id |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShell | Requested Asset Administration Shell |

### 6.2.4 Operation GetAllAssetAdministrationShellsByAssetId

| Operation Name | GetAllAssetAdministrationShellsByAssetId | |
|---|---|---|
| Explanation | Returns all Asset Administration Shell that are linked to a globally unique asset identifier or to specific asset ids. | |
| semanticId | https://admin-shell.io/aas/API/GetAllAssetAdministrationShellsByAssetId/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |

| Operation Name | GetAllAssetAdministrationShellsByAssetId | |
|---|---|---|
| key | string | The key name of the specific asset identifier (IdentifierKeyValuePair/key) or the predefined key "*globalAssetId*" that would refer to the *AssetInformation/globalAssetId*. |
| keyIdentifier | string | The key identifier object |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShell[0..*] | Requested Asset Administration Shells |

### 6.2.5 Operation GetAllAssetAdministrationShellsByIdShort

| Operation Name | GetAllAssetAdministrationShellsByIdShort | |
|---|---|---|
| Explanation | Returns all Asset Administration Shells with a specific *idShort* | |
| semanticId | https://admin-shell.io/aas/API/GetAllAssetAdministrationShellsByIdShort/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| idShort | string | The Asset Administration Shell's idShort |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShell[0..*] | Requested Asset Administration Shells |

### 6.2.6 Operation PutAssetAdministrationShell

| Operation Name | PutAssetAdministrationShell |
|---|---|
| Explanation | Creates a new or updates an existing Asset Administration Shell |

| Operation Name | PutAssetAdministrationShell | |
|---|---|---|
| semanticId | https://admin-shell.io/aas/API/PutAssetAdministrationShell/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| aas | AssetAdministrationShell | Asset Administration Shell object |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | AssetAdministrationShell | Created or updated Asset Administration Shell |

### 6.2.7 Operation DeleteAssetAdministrationShellById

| Operation Name | DeleteAssetAdministrationShellById | |
|---|---|---|
| Explanation | Deletes an Asset Administration Shell | |
| semanticId | https://admin-shell.io/aas/API/DeleteAssetAdministrationShellById/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| id | Identifier | The Asset Administration Shell’s unique id |
| Output Parameter | | |
| statusCode | StatusCode | Status code |

## 6.3 Submodel Repository Interface and Operations

### 6.3.1 Interface Submodel Repository

| Interface: Submodel Repository | |
|---|---|
| Operation Name | Description |
| GetAllSubmodels | Returns all Submodels |

| Interface: Submodel Repository | |
|---|---|
| GetSubmodelById | Returns a specific Submodel |
| GetAllSubmodelsBySemanticId | Returns all Submodels with a specific SemanticId |
| GetAllSubmodelsByIdShort | Returns all Submodels with a specific *idShort* |
| PutSubmodel | Creates a new or updates an existing Submodel |
| DeleteSubmodelById | Deletes a Submodel |

### 6.3.2 Operation GetAllSubmodels

| Operation Name | GetAllSubmodels | |
|---|---|---|
| Explanation | Returns all Submodels | |
| semanticId | https://admin-shell.io/aas/API/GetAllSubmodels/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | Submodel[0..*] | Requested Submodels |

### 6.3.3 Operation GetSubmodelById

| Operation Name | GetSubmodelById | |
|---|---|---|
| Explanation | Returns a specific Submodel | |
| semanticId | https://admin-shell.io/aas/API/GetSubmodelById/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |

| Operation Name | GetSubmodelById | |
|---|---|---|
| id | Identifier | The Submodel's unique id |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | Submodel | Requested Submodel |

### 6.3.4 Operation GetAllSubmodelsBySemanticId

| Operation Name | GetAllSubmodelsBySemanticId | |
|---|---|---|
| Explanation | Returns all Submodels with a specific Semantic-Id | |
| semanticId | https://admin-shell.io/aas/API/GetAllSubmodelsBySemanticId/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| semanticId | Reference | Identifier of the semantic definition |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | Submodel[0..*] | Requested Submodels |

### 6.3.5 Operation GetAllSubmodelsByIdShort

| Operation Name | GetAllSubmodelsByIdShort | |
|---|---|---|
| Explanation | Returns all Submodels with a specific *idShort* | |
| semanticId | https://admin-shell.io/aas/API/GetAllSubmodelsByIdShort/1/0/RC01 | |
| Name | Type | Description |

| Operation Name | GetAllSubmodelsByIdShort | |
|---|---|---|
| Input Parameter | | |
| idShort | string | The Submodel’s idShort |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | Submodel[0..*] | Requested Submodels |

### 6.3.6 Operation PutSubmodel

| Operation Name | PutSubmodel | |
|---|---|---|
| Explanation | Creates a new or updates an existing Submodel | |
| semanticId | https://admin-shell.io/aas/API/PutSubmodel/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| submodel | Submodel | Submodel object |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | Submodel | Created or updated Submodel |

### 6.3.7 Operation DeleteSubmodelById

| Operation Name | DeleteSubmodelById | |
|---|---|---|
| Explanation | Deletes a Submodel | |
| semanticId | https://admin-shell.io/aas/API/DeleteSubmodelById/1/0/RC01 | |
| Name | Type | Description |

| Operation Name | DeleteSubmodelById | |
|---|---|---|
| Input Parameter | | |
| id | Identifier | The Submodel’s unique id |
| Output Parameter | | |
| statusCode | StatusCode | Status code |

# 7 Interfaces Publish and Discovery

## 7.1 General

These interfaces allow to publish information about administration shells or submodels that allow a search for IDs of the corresponding elements in a subsequent discover interface call. The discover interface allows the search for identifiers with appropriate request descriptions (queries, lists of attributes, regular expressions,..).  These Identifiers can be used to lookup the descriptors of the corresponding element (by use of registration and lookup interfaces described in Clause 5) to be able to call the interfaces to access data of the element described in Clause 3.5.

Sometimes, these kinds of services are also classified as exposure and discovery services.

## 7.2 Asset Administration Shell Basic Discovery Interface and Operations

### 7.2.1 Interface Asset Administration Shell Basic Discovery

| Interface: Asset Administration Shell Basic Discovery | |
|---|---|
| Operation Name | Description |
| GetAllAssetAdministrationShellIdsByAssetId | Returns all Asset Administration Shell Ids that are linked to a globally unique asset identifier or to specific asset ids |
| PutAssetId | Creates or updates the link between the Asset Administration Shell Id and the globally unique Asset id |

### 7.2.2 Operation GetAllAssetAdministrationShellIdsByAssetId

| Operation Name | GetAllAssetAdministrationShellIdsByAssetId | |
|---|---|---|
| Explanation | Returns all Asset Administration Shell Ids that are linked to a globally unique asset identifier or to specific asset ids | |
| semanticId | https://admin-shell.io/aas/API/GetAllAssetAdministrationShellIdsByAssetId/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| key | string | The key name of the specific asset identifier (IdentifierKeyValuePair/key) or the predefined key “globalAssetId” that would refer to the AssetInformation/globalAssetId. |
| keyIdentifier | string | The key identifier object |
| Output Parameter | | |

| Operation Name | GetAllAssetAdministrationShellIdsByAssetId | |
|---|---|---|
| statusCode | StatusCode | Status code |
| payload | Identifier[0..*] | Requested Asset Administration Shell Identifiers |

### 7.2.3 Operation PutAssetId

| Operation Name | PutAssetId | |
|---|---|---|
| Explanation | Creates or updates the link between the Asset Administration Shell Id and the globally unique or a specific Asset id | |
| semanticId | https://admin-shell.io/aas/API/PutAssetId/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| id | Identifier | The Asset Administration Shell’s unique id |
| key | string | The key name of the specific asset identifier (IdentifierKeyValuePair/key) or the predefined key “*globalAssetId*” that would refer to the *AssetInformation/globalAssetId*. |
| keyType | IdentifierType[0..1] | Key type.<br>Mandatory for global asset id (corresponds to *Identifier/idType*) |
| keyIdentifier | string | The key identifier object |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | IdentifierKeyValuePair or Reference | Created or updated Asset Identifier. In case of the global asset id a Reference is returned, in case of a specific asset id an identifier key value pair is returned. |

## 7.3 Submodel Discovery Basic Interface and Operations

### 7.3.1 Interface Submodel Discovery

| Interface: Submodel Discovery | |
|---|---|
| Operation Name | Description |
| GetAllSubmodelIdsBySemanticId | Returns all Submodel Ids found based on a specific SemanticId |

### 7.3.2 Operation GetSubmodelIdsBySemanticId

| Operation Name | GetAllSubmodelIdsBySemanticId | |
|---|---|---|
| Explanation | Returns all Submodel Ids found based on a specific SemanticId | |
| semanticId | https://admin-shell.io/aas/API/GetAllSubmodelIdsBySemanticId/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| semanticId | Reference | Identifier of the semantic definition |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | Identifier[0..*] | Requested Submodel Identifiers |

# 8 Interfaces Concept Descriptions Access

## 8.1 General

In this Clause all interfaces and operations w.r.t. concept descriptions in a repository are specified.

## 8.2 Concept Description Repository Interface and Operations

### 8.2.1 Interface Concept Description Repository

| Interface: Concept Description Repository | |
|---|---|
| Operation Name | Description |
| GetAllConceptDescriptions | Returns all Concept Descriptions |
| GetConceptDescriptionById | Returns a specific Concept Description |
| GetAllConceptDescriptionsByIdShort | Returns all Concept Descriptions with a specific *idShort* |
| GetAllConceptDescriptionsByIsCaseOf | Returns all Concept Descriptions with a specific *IsCaseOf*-reference |
| GetAllConceptDescriptionsWithDataSpecificationReference | Returns all Concept Descriptions with a specific *dataSpecification* reference |
| PutConceptDescription | Creates a new or updates an existing Concept Description |
| DeleteConceptDescriptionById | Deletes a Concept Description |

### 8.2.2 Operation GetAllConceptDescriptions

| Operation Name | GetAllConceptDescriptions | |
|---|---|---|
| Explanation | Returns all Concept Descriptions | |
| semanticId | https://admin-shell.io/aas/API/GetAllConceptDescriptions/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |

| Operation Name | GetAllConceptDescriptions | |
|---|---|---|
| statusCode | StatusCode | Status code |
| payload | ConceptDescription[0..*] | Requested Concept Descriptions |

### 8.2.3 Operation GetConceptDescriptionById

| Operation Name | GetConceptDescriptionById | |
|---|---|---|
| Explanation | Returns a specific Concept Description | |
| semanticId | https://admin-shell.io/aas/API/GetConceptDescriptionById/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| id | Identifier | The Concept Description's unique id |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | ConceptDescription | Requested Concept Description |

### 8.2.4 Operation GetAllConceptDescriptionsByIdShort

| Operation Name | GetAllConceptDescriptionsByIdShort | |
|---|---|---|
| Explanation | Returns all Concept Descriptions with a specific *idShort* | |
| semanticId | https://admin-shell.io/aas/API/GetAllConceptDescriptionsByIdShort/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| idShort | string | The Concept Description's idShort |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | ConceptDescription[0..*] | Requested Concept Descriptions |

### 8.2.5 Operation GetAllConceptDescriptionsByIsCaseOf

| Operation Name | GetAllConceptDescriptionsByIsCaseOf | |
|---|---|---|
| Explanation | Returns all Concept Descriptions with a specific *IsCaseOf*-reference | |
| semanticId | https://admin-shell.io/aas/API/GetAllConceptDescriptionsByIsCaseOf/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| isCaseOf | Reference | IsCaseOf reference |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | ConceptDescription[0..*] | Requested Concept Descriptions |

### 8.2.6 Operation GetAllConceptDescriptionsWithDataSpecificationReference

| Operation Name | GetAllConceptDescriptionsWithDataSpecificationReference | |
|---|---|---|
| Explanation | Returns all Concept Descriptions with a specific *dataSpecification* reference | |
| semanticId | https://admin-shell.io/aas/API/GetAllConceptDescriptionsWithDataSpecificationReference/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| dataSpecification-Reference | Reference | *DataSpecification* reference |
| outputModifier | OutputModifier | Determines the result format filtering of the response |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | ConceptDescription[0..*] | Requested Concept Descriptions |

### 8.2.7 Operation PutConceptDescription

| Operation Name | PutConceptDescription | |
|---|---|---|
| Explanation | Creates a new or updates an existing Concept Description | |
| semanticId | https://admin-shell.io/aas/API/PutConceptDescription/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| conceptDescription | ConceptDescription | Concept Description object |
| Output Parameter | | |
| statusCode | StatusCode | Status code |
| payload | ConceptDescription | Created or updated Concept Description |

### 8.2.8 Operation DeleteConceptDescriptionById

| | | |
|---|---|---|
| Operation Name | DeleteConceptDescriptionById | |
| Explanation | Deletes a Concept Description | |
| semanticId | https://admin-shell.io/aas/API/DeleteConceptDescriptionById/1/0/RC01 | |
| Name | Type | Description |
| Input Parameter | | |
| id | Identifier | The Concept Description's unique id |
| Output Parameter | | |
| statusCode | StatusCode | Status code |

# 9 Data Types for Payload

## 9.1 General

For metamodel elements like AssetAdministrationShell, Submodel, Identifier etc. that are specified in [1], please refer to [1]. In this clause, additional classes needed for interface payloads are specified.

## 9.2 Metamodel Specification Details: Designators

### 9.2.1 AssetAdministrationShellDescriptor

| Class Name | AssetAdministrationShellDescriptor | |
|---|---|---|
| Explanation | Descriptor of an Asset Administration Shell | |
| Inherits from | - | |
| semanticId | https://admin-shell.io/aas/API/AssetAdministrationShellDescriptor/1/0/RC01 | |
| Attribute<br>(* = mandatory) | Type | Description |
| administration | AdministrativeInformation[0..1] | Administrative information of the Asset Administration Shell. |
| description | LangStringSet[0..1] | Description or comments on the Asset Administration Shell. |
| globalAssetId | Reference[0..1] | Reference to either an Asset object or a global reference to the asset the AAS is representing. |
| specificAssetId | IdentifierKeyValuePair[0..*] | Specific asset identifier. |
| endpoint | Endpoint[0..*] | Endpoint of the Asset Administration Shell. |
| idShort | String[0..1] | Short name of the Asset Administration Shell. |
| identification* | Identifier | Globally unique identification of the Asset Administration Shell. |
| submodelDescriptor | SubmodelDescriptor[0..*] | Descriptor of a submodel of the Asset Administration Shell. |

### 9.2.2 SubmodelDescriptor

| Class Name | SubmodelDescriptor | |
|---|---|---|
| Explanation | A descriptor of a submodel | |
| Inherits from | - | |
| semanticId | https://admin-shell.io/aas/API/SubmodelDescriptor/1/0/RC01 | |
| Attribute (* = mandatory) | Type | Description |
| administration | AdministrativeInformation[0..1] | Administrative information of the Submodel. |
| description | LangStringSet[0..1] | Description or comments on the Asset Administration Shell. |
| endpoint | Endpoint[0..*] | Endpoint of the Submodel |
| idShort | String[0..1] | Short name of the Submodel. |
| identification* | Identifier | Globally unique identification of the Submodel. |
| semanticId | Reference[0..1] | Identifier of the semantic definition of the Submodel. |

### 9.2.3 Endpoint

| Class Name | Endpoint | |
|---|---|---|
| Explanation | An endpoint | |
| Inherits from | - | |
| semanticId | https://admin-shell.io/aas/API/Endpoint/1/0/RC01 | |
| Attribute (* = mandatory) | Type | Description |
| address | string | Address |
| type | string | Type of the endpoint. |

### 9.2.4 Status Code, Error Handling & Result Messages

In this clause it will be dealt with the error and result handling of an operation's execution in a technology-independent manner.

The first section covers generic status codes that are returned on each and every request independent of the operation's success or failure. The subsequent section describes the result object that is returned in case of failure.

#### 9.2.4.1 Generic Status Codes

Successful operations return one of the success status codes and their respective payload. Unsuccessful operations return one of the failure status codes and a result object as defined in Clause 9.2.4.2.

Table 1 shows generic status codes being returned to the requester. Additionally, the table indicates whether a specific status code comes with a result object in the returned payload.

| Generic Status Code | Meaning | Has Result Object |
|---|---|---|
| Success | Success | No |
| SuccessCreated | Creation of a new resource successful | No |
| SuccessNoContent | Success with explicitly no content in the payload | No |
| ClientForbidden | Request is unauthorized | Yes |
| ClientErrorBadRequest | Bad or malformed request | Yes |
| ClientMethodNotAllowed | Operation request is not allowed | Yes |
| ClientErrorResourceNotFound | Resource not found | Yes |
| ServerInternalError | Unexpected error | Yes |
| ServerErrorBadGateway | Bad Gateway | Yes |

#### 9.2.4.2 General Result Object

In case of a failed operation execution a result object shall be returned containing more information about the reasons why the operation failed to execute.

| Class Name | Result | |
|---|---|---|
| Explanation | The result object | |
| Attribute (* = mandatory) | Type | Description |

| Class Name | Result | |
|---|---|---|
| success* | boolean | Indicated whether the operation execution is seen as successful |
| message | Message[0..*] | Additional message containing information for the requester |

| Class Name | Message | |
|---|---|---|
| Explanation | A message containing more information for the requester about a certain happening in the backend. | |
| Attribute (* = mandatory) | Type | Description |
| messageType* | MessageTypeEnum | The message type |
| text* | string | The message text |
| code | String[0..1] | Technology-dependent status or error code |
| timestamp | dateTime[0..1] | Timestamp of the message |

| Enumeration | MessageTypeEnum |
|---|---|
| Explanation | The message type |
| Literal | Explanation |
| Info | Used to inform the user about a certain fact |
| Warning | Used for warnings. Warnings may lead to errors in the subsequent execution |
| Error | Used for handled errors |
| Exception | Used if it is an internal and/or unhandled exception that occurred |

# 10 Base Operation Parameters

## 10.1 General

In this clause the parameters used in the operation specifications are specified.

## 10.2 OutputModifier in Operations

**Definition**

The OutputModifier indicates the requester's expected or desired format of the response content of a requested operation. The OutputModifier comprises three orthogonal choices/fields etc. These enumerations combined form the response content of the requested operation.

1. **Enumeration: Level**

The first enumeration *Level* indicates the depth of the response content's structure.

| **Value** | **Explanation** |
|---|---|
| Deep (Default) | All elements of requested hierarchy level and all children on all sublevels are returned |
| Core | Only elements of a requested hierarchy level as well as direct children are being returned |

2. **Enumeration: Content**

The second enumeration *Content* indicates the kind of the response content's serialization.

| **Value** | **Explanation** |
|---|---|
| Normal (Default) | The standard serialization of the model element is applied. |
| Value | Only the raw value of the model element is being returned according to Clause 10.3.1 |
| Reference | Only applicable to Referables. The reference to found elements is being returned. |
| Path | Returns a list of *idShort* paths to found elements within a SubmodelElement hierarchy |

3. **Enumeration: Extent**

The third enumeration *Extent* indicates to which extent the result content is being serialized.

| **4. Value** | **5. Explanation** |
|---|---|
| WithoutBLOBValue (Default) | Only applicable to BLOB-elements. The BLOB content is not being returned. |
| WithBLOBValue | Only applicable to BLOB-elements. The BLOB content is being returned as *base64* encoded string |

## 10.3 Serialization in Specified Formats (Output Modifier)

### 10.3.1 General

For the output modifier „Content = Value“ it depends on the serialization output format how it is realized.

Up to now only the serialization for JSON is specified .For other serialization formats (e.g. XML, RDF etc.) this has to be defined in a similar way but is not yet part of this document version.

### 10.3.2 Serialization in JSON Values Format

This clause explains how a return value is serialized in JSON if the output modifier „Content = Value“ is set.

In many cases, applications using the data from Asset Administration Shells already know the submodel regarding its structure, attributes and semantics. Consequently, there is not always a need to receive the entire model information in each and every request since they are stable most of the time. Instead, applications are mostly interested only in the raw values of the modelled data. Furthermore, having limited processing power or limited bandwidth, the use case of this output modifier is to transfer data as efficient as possible.

Values are only available for

- All subtypes of abstract type *DataElement*,
- SubmodelElementCollection,
- ReferenceElement,
- RelationshipElement + AnnotatedRelationshipElement,
- Entity

Operations and Events and Capabilities are considered to be excluded from the scope of that output modifier since only the output for elements containing data is relevant. In the serialization they are omitted.

The following rules shall be adhered when serializing a submodel with the output modifier *Value*:

- A submodel is serialized as an unnamed JSON object.
- A submodel element is considered a leaf submodel element if it does not contain other submodel elements. A leaf submodel element follows the rules as described in the following for the different submodel elements considered in the serialization. Otherwise, i.e. if not a leaf element, it means transitively following the serialization rules until the value is a leaf submodel element.
- For each submodel element:
    - *Property* is serialized as `${Property/idShort}: ${Property/value}` where `${Property/value}` is the JSON serialization of the respective property’s value.
    - *MultiLanguageProperty* is serialized as named JSON object with `${MultiLanguageProperty/idShort}` as the name of the containing JSON property. The JSON object contains JSON properties for each language of the *MultiLanguageProperty* with the language as name and the corresponding localized string as value. The language name is defined as two chars according to ISO 639-1.

- *Range* is serialized as named JSON object with `${Range/idShort}` as the name of the containing JSON property. The JSON object contains two JSON properties. The first is named "min". The second is named "max". Their corresponding values are `${Range/min}` resp. `${Range/max}`.

- *File* and *Blob* are serialized as named JSON objects with `${File/idShort}` or `${Blob/idShort}` as the name of the containing JSON property. The JSON object contains two JSON properties. The first refers to the mime type named "mimeType" `${File/mimeType}` resp. `${Blob/mimeType}`. The seconds refers to the value named "value" `${File/value}` resp. `${Blob/value}`.

- *SubmodelElementCollection*s are serialized depending on the attribute *allowDuplicates*.
  - allowDuplicates = true
    - The collection is assumed to be a list, set or bag and hence serialized as named JSON array with `${Collection/idShort}` as the name of the containing JSON property. The elements contained within the collection are serialized as unnamed JSON objects.
  - allowDuplicates = false
    - The collection is assumed to be an entity with distinct elements and hence serialized as named JSON object with `${Collection/idShort}` as the name of the containing JSON property.
  - Nested case: The *SubmodelElementCollection's* contained submodel elements `${Collection/value}` are serialized according to the rules mentioned in this section.

- *ReferenceElement* is serialized as `${ReferenceElement/idShort}`: `${ReferenceElement/value}` where `${ReferenceElement/value}` is the standardized string representation of a reference according to Clause *Serialization of Values of Type "Reference"* in [2].

- *RelationshipElement* is serialized as named JSON object with `${ReleationshipElement/idShort}` as the name of the containing JSON property. The JSON object contains two JSON properties. The first is named "first". The second is named "second". Their corresponding values are `${RelationshipElement/first}` resp. `${Relationship/second}`. The values are serialized according to the serialization of a *ReferenceElement* see above.

- *AnnotatedRelationshipElement* is serialized according to the serialization of a *ReleationshipElement* see above. Additionally, a third named JSON object is introduced with "annotation" as the name of the containing JSON property. The value is `${AnnotatedRelationshipElement/annotation}`. The value is serialized depending on the type of the annotation data element.

- *Entity* is serialized as named JSON object with `${Entity/idShort}` as the name of the containing JSON property. The JSON object contains three JSON properties. The first is named "statements" `${Entity/statements}` and contains the serialized submodel elements according to their respective serialization mentioned in this clause. The second is named either “globalAssetId” or “specificAssetId” and contains either a *Reference* (see above) or an *IdentifierKeyValuePair*. The third property is named “entityType” and contains a string representation of `${Entity/entityType}`.

  - *IdentifierKeyValuePair* is serialized as named JSON object with three JSON properties names as the attributes of *IdentifierKeyValuePair*.

- Submodel elements defined in the submodel other than the ones mentioned above are not subject to serialization of that output modifier.

Examples conformant to [3]:

Standard-Output for a single submodel element (here: *Property*) in the payload:

```
{
  "value": "5000",
  "semanticId": {
    "keys": [
      {
        "type": "ConceptDescription",
        "value": "0173-1#02-BAA120#008",
        "index": 0,
        "idType": "IRDI"
      }
    ]
  },
  "constraints": [],
  "idShort": "MaxRotationSpeed",
  "category": "PARAMETER",
  "modelType": {
    "name": "Property"
  },
  "valueType": {
    "dataObjectType": {
      "name": "integer"
    }
  },
  "kind": "Instance"
}
```

With output modifier *Value* the payload is minimized to the following

```
          "MaxRotationSpeed": "5000"
```

For a *SubmodelElementCollection* (allowDuplicates = false) meaning that submodel elements within the collection can only have distinct *semanticId*s (in this example: for each the mother's name, the father's name and the son's name), the collection is serialized as objects denoted by curly brackets:

```
          "NamesOfFamilyMembers": {
               "NameOfMother": "Mary ExampleFamily",
               "NameOfFather": "Jonathan ExampleFamily",
               "NameOfSon": "Clark ExampleFamily"
           }
```

For a *SubmodelElementCollection* (allowDuplicates = true) meaning that submodel elements within the collection have the same *semanticId* (in this example: all family members have a name that in turn has one *semanticId* for the name) are allowed, the collection is serialized as array denoted by square brackets:

```
   "NamesOfFamilyMembers": [
               "Mary ExampleFamily", "Jonathan ExampleFamily", "Clark ExampleFamily"
           ]
```

For *SubmodelElementCollection* (allowDuplicates = true) named "Families" containing other *SubmodelElementCollection*s named "NamesOfFamilyMembers" (allowDuplicates = false) the payload is serialized as follows:

```
   "Families": [
{
          "NamesOfFamilyMembers": {
               "NameOfMother": "Mary ExampleFamily",
               "NameOfFather": "Jonathan ExampleFamily",
               "NameOfSon": "Clark ExampleFamily"
                       }
               },
{
          "NamesOfFamilyMembers": {
               "NameOfMother": "Mary OtherFamily",
               "NameOfFather": "Thomas OtherFamily",
               "NameOfSon": "Harry OtherFamily"
                       }
               }
    ]
```

For *SubmodelElementCollection* (allowDuplicates = true) named "Families" containing other *SubmodelElementCollection*s named "NamesOfFamilyMembers" (allowDuplicates = true) the payload is serialized as follows:

```
 "Families": [
{
    "NamesOfFamilyMembers":
     [ "Mary ExampleFamily", "Jonathan ExampleFamily", "Clark ExampleFamily" ]
               },
{
   "NamesOfFamilyMembers":
      [ "Mary ExampleFamily", "Thomas ExampleFamily", "Bruce ExampleFamily" ]
}
           ]
```

For a *MultiLanguageProperty* named “Label” the payload is minimized to the following:

```
   "Label": {
               "de": "Das ist ein deutscher Bezeichner",
               "en": "That’s an English label"
          }
```

In case a preferred language is defined, say English, than the payload is reduced to:

```
   "Label": “That’s an English label"
```

For a *Range* named “TorqueRange” the payload is minimized to the following:

```
          "TorqueRange": {
               "Min": "50",
               "Max": "5000"
           }
```

For a *ReferenceElement* named “MaxRotationSpeedReference” the payload is minimized to the following:

```
       "MaxRotationSpeedReference":
"(Submodel)[IRI]http://customer.com/demo/aas/1/1/1234859590,(Property)[IdShort]MaxRotationSpeed"
```

For a *File* named “Document” the payload is minimized to the following:

```
          "Document": {
               "mimeType": "application/pdf",
               "value": "SafetyInstructions.pdf"
           }
```

For a *Blob* named “Library” the payload is minimized to the following if the output modifier *Extent* is set to ***WithoutBLOBValue***

```
          "Libary": {
               "mimeType": "application/pdf"
           }
```

If the output modifier Extent is set to ***WithBlobValue***, there is an additional attribute containing the base64 encoded value:

```
    "Libary": {
        "mimeType": "application/pdf",
        "value": "VGhpcyBpcyBteSBibG9i"
     }
```

For a *RelationshipElement* named “CurrentFlowsFrom” the payload is minimized to the following:

```
"CurrentFlowsFrom": {
       "first":  "(Submodel)[IRI]http://customer.com/demo/aas/1/1/1234859590,
(Property)[IdShort]PlusPole"
       "second": "(Submodel)[IRI]http://customer.com/demo/aas/1/0/1234859123490,
(Property)[IdShort]MinusPole"
}
```

For an *Entity* named “MySubAssetEntity” the payload is minimized to the following:

```
"MySubAssetEntity": {
       "statements":  [ { "MaxRotationSpeed": "5000" } ],
       "entityType": "SelfManagedEntity",
       "globalAssetId": "(Asset)[IRI]http://customer.com/demo/asset/1/1/MySubAsset"
}
```

## 10.4 SearchOptions Used in Operations

**Definition**

The *SearchOption*s determine the scope of the search space and thus define the subject of searches.

The search space can be restricted by the level of deepness defined by the enumeration *Depth*.

SearchOption **Depth** with Enumeration: **DepthEnum**

The enumeration *DepthEnum* indicates the depth of the search space.

| **Value** | **Explanation** |
|---|---|
| Recursive (Default) | The search space is unrestricted. All elements of all hierarchy levels are subject to searches. |
| Non-Recursive | The search space is restricted to the current requested element and its directly listed children. Only these elements are subject to searches. |

In the following some informal and simplified examples are sketched:

Submodel: MySubmodel

- ⇨ Property: MyTopLevelProperty (SemanticId: TopLevelSemanticId)
- ⇨ SMC: MySubmodelElementCollection (SemanticId: SubSMCSemanticId)
    - o Property: MySubProperty1 (SemanticId: SubProperty1SemancticId)
    - o Property: MySubProperty2 (SemanticId: SubProperty2SemancticId)
    - o SMC: MySubSubmodelElementCollection (SemanticId: SubSubSMCSemanticId)
        - ▪ Property: MySubSubProperty1 (SemanticId: SubSubProperty1SemancticId)
        - ▪ Property: MySubSubProperty2 (SemanticId: SubSubProperty2SemancticId)

Request (GetSubmodelElementByPathAndSemanticId):

**GET** SubmodelElement **FROM** MySubmodel/MySubmodelElementCollection

**WHERE** *SemanticId* = SubSubProperty1SemancticId **AND**

SEARCH_OPTIONS (DEPTH = RECURSIVE)

Response:

Status-Code: **SUCCESS**

Content: Property: MySubSubProperty1

Request (GetSubmodelElementByPathAndSemanticId):

**GET** SubmodelElement **FROM** MySubmodel/MySubmodelElementCollection

**WHERE** *SemanticId* = SubSubProperty1SemancticId **AND**

SEARCH_OPTIONS *(*DEPTH = NON-RECURSIVE)

Response:

Status-Code: **NOTFOUND**

# 11 Summary and Outlook

This document specifies the interfaces for a single Asset Administration Shell and its submodels as well as for a repository of Asset Administration Shells. Additionally, infrastructural interfaces like Registry and Lookup and Discovery of a set of Asset Administration Shells are specified. All specifications are provided in a technology neutral way.

In subsequent versions of this specification APIs specified for a specific technology will be added. The first technology to be considered is http/REST. Other technologies that are planned to be supported in the future are OPC UA and MQTT.

Additionally, also some more interfaces, basic services or service profiles may be defined. Querying will be a topic.

Another very important topic that will be looked at in next versions of the specification is the very important topic of access control to the information an Asset Administration Shell provides and the trustworthiness of the information.

# Annex

# Annex A. Templates Used for Specification

In this Annex the table templates used for documentation of interfaces, operations, data types etc. are explained.

**Table 1 Interface Description**

| Interface: <Interface Name> | |
|---|---|
| Operation Name | Description |
| Oper1 | Human understandable description of the operation of the interface. Only major input and output information shall be described, no individual request and result parameters. Note: All words in the service operation name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case |
| … | |
| operN (optional) | Human understandable description of the operation n of the interface. Optional operations are to be marked by suffix (optional) after the operation name. |

**Table 2 Operation Description**

| | |
|---|---|
| Operation Name: | Name of the Operation: All individual words in the operation name are capitalized |
| Explanation: | Human understandable description of the functionality.<br><br>The operation provides its functionality through the following input and output parameters:<br><br>• Input Parameter 1: human understandable description of the purpose of the input parameter 1<br><br>• …<br><br>• Input Parameter N: human understandable description of the purpose of input parameter N<br><br>• Output Parameter 1: human understandable description of the purpose of output parameter 1: human understandable description of the purpose of the input parameter 1<br><br>• …<br><br>• Output Parameter N: human understandable description of the purpose of output parameter N:<br><br>If ***payload*** is mentioned as output parameter, only the returned payload in case of a successful operation (status code: Success, SuccessCreated) is denoted in column *Type*. In case of failure see Clause 9.2.4. |

| Operation Name: | Name of the Operation: All individual words in the operation name are capitalized | |
|---|---|---|
| | If <u>no</u> ***payload*** is mentioned as output parameter, the status code shall be SuccessNoContent in case of success, otherwise see Clause 9.2.4.<br><br>Convention: All words in the interface name are written together in italics without a blank in between. The first letter of the first word and all other words are written in upper case letters. | |
| semanticId | The unique identifier of this operation. | |
| Name | Type | Description |
| Input Parameter | | |
| inputParameter1 | Type of the input parameter 1 | Human understandable description of the input parameter 1 of the operation. Note: All words in the parameter name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case. |
| … | | |
| inputParameterN | Type of the input parameter N | Human understandable description of the input parameter N of the operation. Note: All words in the parameter name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case. |
| Output Parameter | | |
| outputParameter1 | Type of the output parameter 1 | Human understandable description of the output parameter 1 of the operation. Note: All words in the parameter name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case. |
| … | | |
| outputParameterN | Type of the output parameter N | Human understandable description of the output parameter N of the operation. Note: All words in the parameter name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case. |

**Table 3 Data Types for Payload Description**

| Class Name: | Name of the Class: All individual words in the clas name are capitalized |
|---|---|
| Explanation: | Human understandable description of the class.<br><br>The Class has following attributes:<br><br>• Attribute 1: human understandable description of the purpose of the attribute 1<br><br>• … |

| | | |
|---|---|---|
| Class Name: | Name of the Class: All individual words in the clas name are capitalized | |
| | • Attribute N: human understandable description of the purpose of the attribute N<br><br>Convention: All words in the class name are written together in italics without a blank in between. The first letter of the first word and all other words are written in upper case letters. | |
| Inherits from: | Name of the class this class inherits from | |
| semanticId | The unique identifier of this class. | |
| Attribute (*=mandatory) | Type | Description |
| attribute1 | Type of the attribute 1 | Human understandable description of the attribute 1 of the class. Note: All words in the attribute name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case. |
| … | | |
| attributeN | Type of the attribute N | Human understandable description of the attribute N of the class. Note: All words in the attribute name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case. |

**Table 4 Enumeration Description**

| | |
|---|---|
| Enumeration Name: | Name of the Enumeration: All individual words in the enumeration name are capitalized |
| Explanation: | Human understandable description of the enumeration.<br>The Enumeration has following literals:<br>• Literal 1: human understandable description of the purpose of the literal 1<br>• …<br>• Literal N: human understandable description of the purpose of the literal N<br><br>Convention: All words in the enumeration name are written together in italics without a blank in between. The first letter of the first word and all other words are written in upper case letters. |
| semanticId | The unique identifier of this enumeration. |
| Literal | Description |

| | |
|---|---|
| Literal1 | Human understandable description of the literal 1 of the enumeration. Note: All words in the literal name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case |
| … | |
| LiteralN | Human understandable description of the literal N of the enumeration. Note: All words in the literal name are written together in italics without a blank in between. The first letter of the first word is lower case, all other words upper case |

**<datatype>+** means that the references are resolved. For instance, AssetAdminstrationShell+ means that the submodels are also returned although only referenced from the Asset Administration Shell.

# Annex B. Legend for UML Modelling

## i. OMG UML General

In the following the used UML elements used in this specification are explained. For more information please refer to the comprehensive literature available for UML. The formal specification can be found in [5].

Figure 3 shows a class with name “Class1” and an attribute with name “attr” of type *Class2*. Attributes are owned by the class. Some of these attributes may represents the end of binary associations, see also Figure 10. In this case the instance of *Class2* is navigable via the instance of the owning class *Class1*.[2]

**Figure 3 Class**

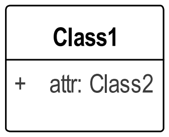


Figure 4 shows that *Class4* is inheriting all member elements from *Class3*. Or in other word, *Class3* is a generalization of *Class4*, *Class4* is a specialization of *Class3*. This means that each instance of *Class4* is also an instance of *Class3*. An instance of the *Class4* has the attributes *attr1* and *attr2* whereas instances of *Class3* only have the attribute *attr1*.

**Figure 4 Inheritance/Generalization**

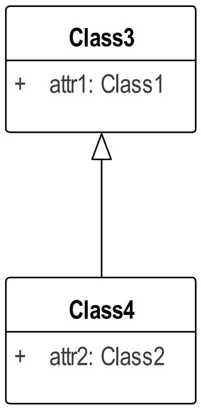


Figure 5 defines the required and allowed multiplicity/cardinality within an association between instances of *Class1* and *Class2*. In this example an instance of *Class2* is always related to exactly one instance of *Class1*. An instance of

[2] „Navigability notation was often used in the past according to an informal convention, whereby non-navigable ends were assumed to be owned by the Association whereas navigable ends were assumed to be owned by the Classifier at the opposite end. This convention is now deprecated. Aggregation type, navigability, and end ownership are separate concepts, each with their own explicit notation. Association ends owned by classes are always navigable, while those owned by associations may be navigable or not. [5]”

*Class1* is either related to none, one or more (unlimited, i.e. no constraint on the upper bound) instances of *Class2*. The relationship can change over time.

Multiplicity constraints can also be added to attributes and aggregations.

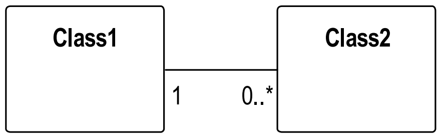
The notation of multiplicity is as follows:

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

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

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

**Figure 5 Multiplicity**



A multiplicity element represents a collection of values. The default is a set, i.e. it is not ordered and the elements within the collection are unique, i.e. contain no duplicates. In Figure 6 an ordered collection is shown: the instances of *Class2* related to an instance of *Class1* are ordered. The stereotype <<ordered>> is used to denote that the relationship is ordered.

**Figure 6 Ordered Multiplicity**

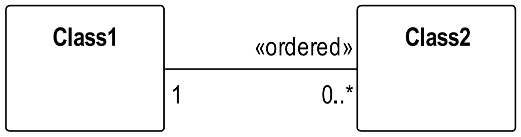


Figure 7 shows that the member ends of an association can be named as well. I.e. an instance of *Class1* can be in relationship "relation" to an instance of *Class2*. Vice versa the instance of *Class2* is in relationship "reverseRelation" to the instance of *Class1*.

**Figure 7 Association**

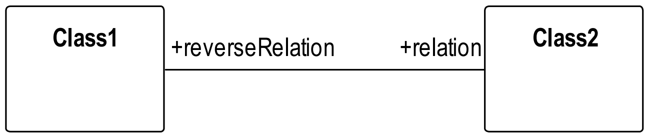


Figure 8 shows a composition, also called a composite aggregation. A composition is a binary association. It groups a set of instances. The individuals in the set are typed as specified by *Class2*. The multiplicity of instances of *Class2* to *Class1* is always 1 (i.e. upper-bound and lower-bound have value "1"). One instance of *Class2* belongs to exactly one instance of *Class1*. There is no instance of *Class2* without a relationship to an instance of *Class1*. In Figure 9 the composition is shown using an association relationship with a filled diamond as composition adornment.

**Figure 8 Composition (composite aggregation)**

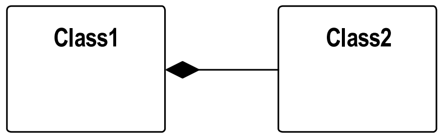


Figure 9 show an aggregation. An aggregation is a binary association. In contrast to a composition an instance of *Class2* can be shared by several instances of *Class1*. In Figure 9 the shared aggregation is shown using an association relationship with a hallow diamond as aggregation adornment.

**Figure 9 Aggregation**

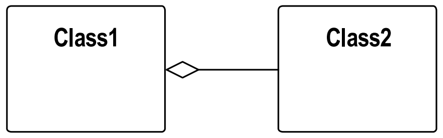


Figure 10 shows that the attribute notation can be used for an association end owned by a class. In this example the attribute name is “attr” and the elements of this attribute are typed with *Class2.* The multiplicity, here “0..*”, is added in square brackets. If the aggregation is ordered then this is added in curly brackets like in this example.

**Figure 10  Navigable Attribute Notation for Associations**

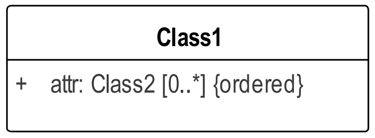


Figure 11 shows that there is a dependency relationship between *Class1* and *Class2*. In this case the dependency means that *Class1* depends on *Class2.* Why is this: because the type of attribute *attr* depends on the specification of class *Class2.* A dependency is shown as dashed arrow between two model elements.

**Figure 11 Dependency**

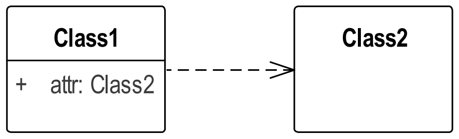


Figure 12 shows an abstract class. It uses the stereotype <<abstract>>. There are no instances of abstract classes. They are typically used to specific member elements that are then inherited by non-abstract classes.

**Figure 12 Abstract Class**

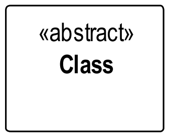


Figure 13 shows a package with name “Package2”. A package is a namespace for its members. In this example the member belonging to *Package2* is class *Class2*.

**Figure 13 Package**

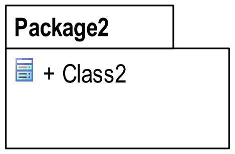
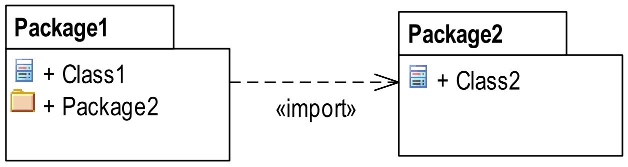


Figure 14 shows that all elements in *Package2* are imported into the namespace defined by *Package1*. This is a special dependency relationship between the two packages with stereotype <<import>>.

**Figure 14 Imported Package**



An enumeration is a data type whose values are enumerated as literals. Figure 15 shows an enumeration with name “Enumeration1”. It contains two literal values, “a” and “b”. It is a class with stereotype <<enumeration>>. The literals owned by the enumeration are ordered.

**Figure 15 Enumeration**

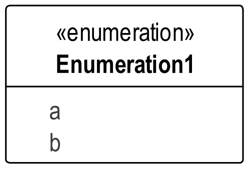


Figure 16 show the definition of the data type with name “DataType1”. A data type is a type whose instances are identified only by their value. It is a class with stereotype <<dataType>>.

**Figure 16 Data Type**

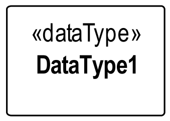


Figure 17 shows a primitive data type with name “int”. Primitive data types are predefined data types, without any substructure. The primitive data types are defined outside UML.

**Figure 17 Primitive Data Type**

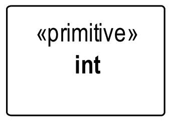


Figure 18 shows how a note can be attached to an element, in this example to class “Class1”.

**Figure 18 Note**

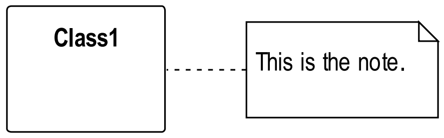
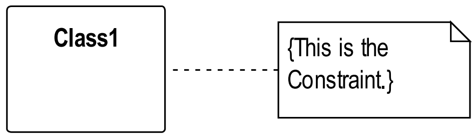


Figure 19 shows how a constraint is attached to an element, in this example to class “Class1”.

**Figure 19 Constraint**



## ii. Notes to Graphical Representation

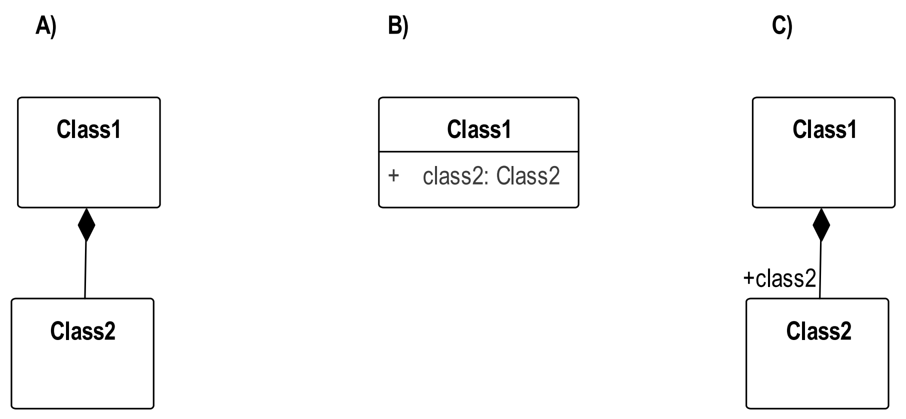
In the following specific graphical modelling rules used in this specification are explained that are not included in this form in [5].

Figure 20 shows two different graphical representations of a composition (composite aggregation). In Variant A) a relationship with a filled aggregation diamond is used. In Variant B) an attribute with the same semantics is defined. And in Variant C) the implicitly assumed default name of the attribute in Variant A) is explicitly stated as such.

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

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

**Figure 20 Graphical Representations of Composite Aggregation/Composition**



In Figure 21 different representations of a shared aggregation are shown. In a shared aggregation a *Class2* instance can exist independent of the existence of an an *Class1* instance. It is just referencing the instances of *Class2*. In Variant B) an attribute with the same semantics is defined. The reference is denoted by a star added after the type of the attribute.

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

A speciality in Figure 21 is that the aggregated instances are referables in the sense of the Asset Administration Shell metamodel (i.e. they inherit from the predefined abstract class “Referable”). This is why Variant C) is also identical to Variant A) and B). This would not be the case for non-referable elements in the metamodel. The structure of a reference to a model element of the Asset Administration Shell is explicitly defined. A reference consists of an ordered list of keys. The last key in the key chain shall reference an instance of type *Class2* (i.e. Reference/type equal to “Class2”).

**Figure 21 Graphical Representation of Shared Aggregation**

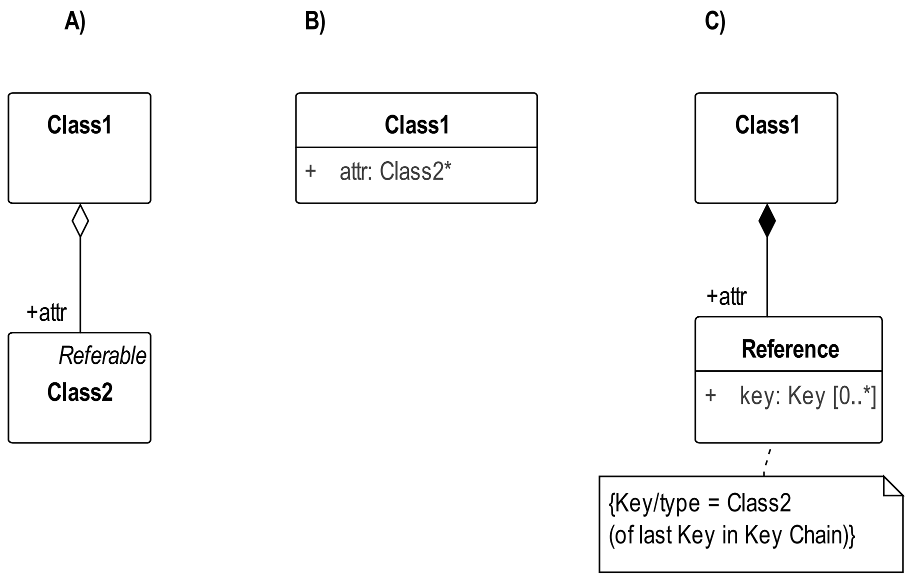
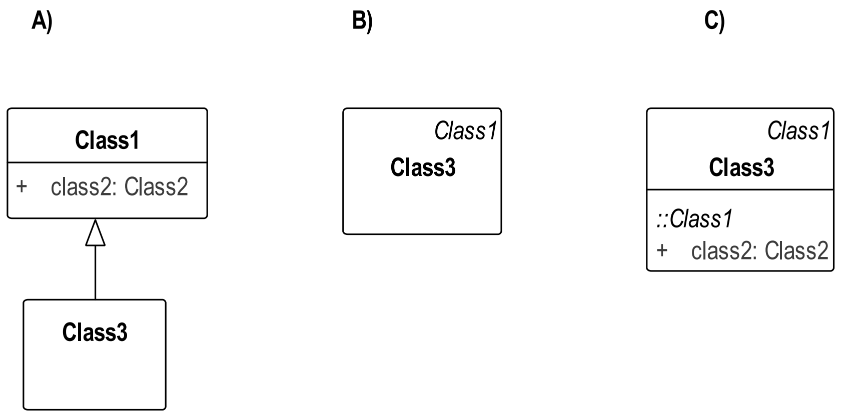


Figure 22 show different graphical representations of generalization. Variant A) is the classical graphical representation as defined in [5]. Variant B) is a short form if *Class1* is not on the same diagram. To see from which class *Class3* is inheriting the name of the class is depicted in the upper right corner.

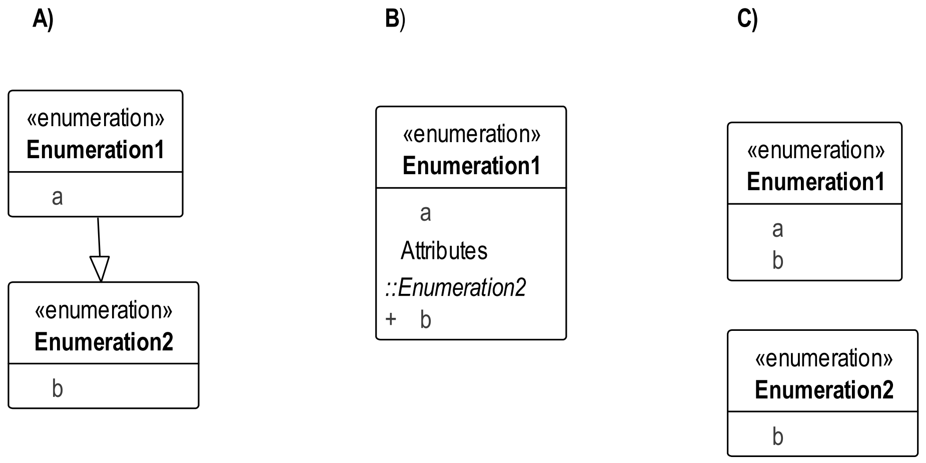
Variant C) is not only showing from which class Class3 instances are inheriting but also what they are inheriting. This is depicted by the class name it is inheriting from followed by "::" and then the list of all inherited elements – here attribute *class2*. Typically, the inherited elements are not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

**Figure 22 Graphical Representation of Generalization/Inheritance**



In Figure 23 different graphical notations for enumerations  in combination with inheritance are shown. In Variant A) enumeration "Enumeration1" additionally contains the literals as defined by "Enumeration2". Note: the direction of inheritance is opposite to the one for class inheritance. This can be seen in Variant C) that defined the same enumerations but without inheritance. In Variant B) another graphical notation is shown that makes it visible which literals are inherited by which enumeration. The literals within an enumeration are ordered so the order of classes it is inheriting from is important.

**Figure 23 Graphical Representation for Enumeration with Inheritance**



# Annex C. Bibliography

[1] Details of the Asset Administration Shell. Document Series. Federal Ministry for Economic Affairs and Energy (BMWi). Online. Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Standardartikel/specification-administrationshell.html

[2] Details of the Asset Administration Shell. Part 1 - The exchange of information between partners in the value chain of Industrie 4.0", Federal Ministry for Economic Affairs and Energy (BMWi). Online. Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details-of-the-Asset-Administration-Shell-Part1.html

[3] “Details of the Asset Administration Shell. Part 1 - The exchange of information between partners in the value chain of Industrie 4.0", Version 3.0RC01. Federal Ministry for Economic Affairs and Energy (BMWi), November 2020. Online. Available: https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/Details-of-the-Asset-Administration-Shell-Part1/3/0.html

[4] Tom Preston-Werner. Semantic Versioning. Version 2.0.0. Online. Available: https://semver.org/spec/v2.0.0.html

[5] OMG Unified Modeling Language (OMG UML). Formal/2017-12-05. Version 2.5.1. December 2018. [Online] Available: https/www.omg.org/spec/UML/[7] DIN SPEC 91406: “Automatic identification of physical objects and information on physical objects in IT systems, particularly IoT systems”. December 2019. https://www.beuth.de/de/technische-regel/din-spec-91406/314564057

[6] RFC 8820: URI Design and Ownership. Internet Engineering Task Force (IETF), 2020. Online. Available: https://tools.ietf.org/html/rfc8820

## AUTHORS

Sebastian Bader, Fraunhofer IAIS

Bernd Berres, MPDV Mikrolab GmbH

Dr. Birgit Boss, Robert Bosch GmbH

Dr. Andreas Graf Gatterburg, Hilscher Gesellschaft für Systemautomation mbH

Dr. Michael Hoffmeister, Festo AG & Co. KG

Yevgen Kogan, KUKA Deutschland GmbH

Alexander Köpke, Microsoft Deutschland GmbH

Matthias Lieske, Hitachi Europe GmbH

Torben Miny, Lehrstuhl für Prozessleittechnik, RWTH Aachen University

Dr. Jörg Neidig, Siemens AG

Andreas Orzelski, Phoenix Contact GmbH & Co. KG

Stefan Pollmeier, ESR Pollmeier GmbH Servo-Antriebstechnik

Manuel Sauer, SAP SE

Daniel Schel, Fraunhofer IPA

Tizian Schröder, OVGU Magdeburg

Mario Thron, Institut f. Automation und Kommunikation e.V. (ifak)

Dr. Thomas Usländer, Fraunhofer IOSB

Jens Vialkowitsch, Robert Bosch GmbH

Friedrich Vollmar

Jörg Wende, IBM Deutschland GmbH

Constantin Ziesche, Robert Bosch GmbH

This document has been elaborated in the Working Group “Asset Administration Shell” and the Working Group “Infrastructure of the Asset Administration Shell”, both subgroups of the Working Group on “Reference Architectures, Standards and Norms “ of Plattform Industrie 4.0.