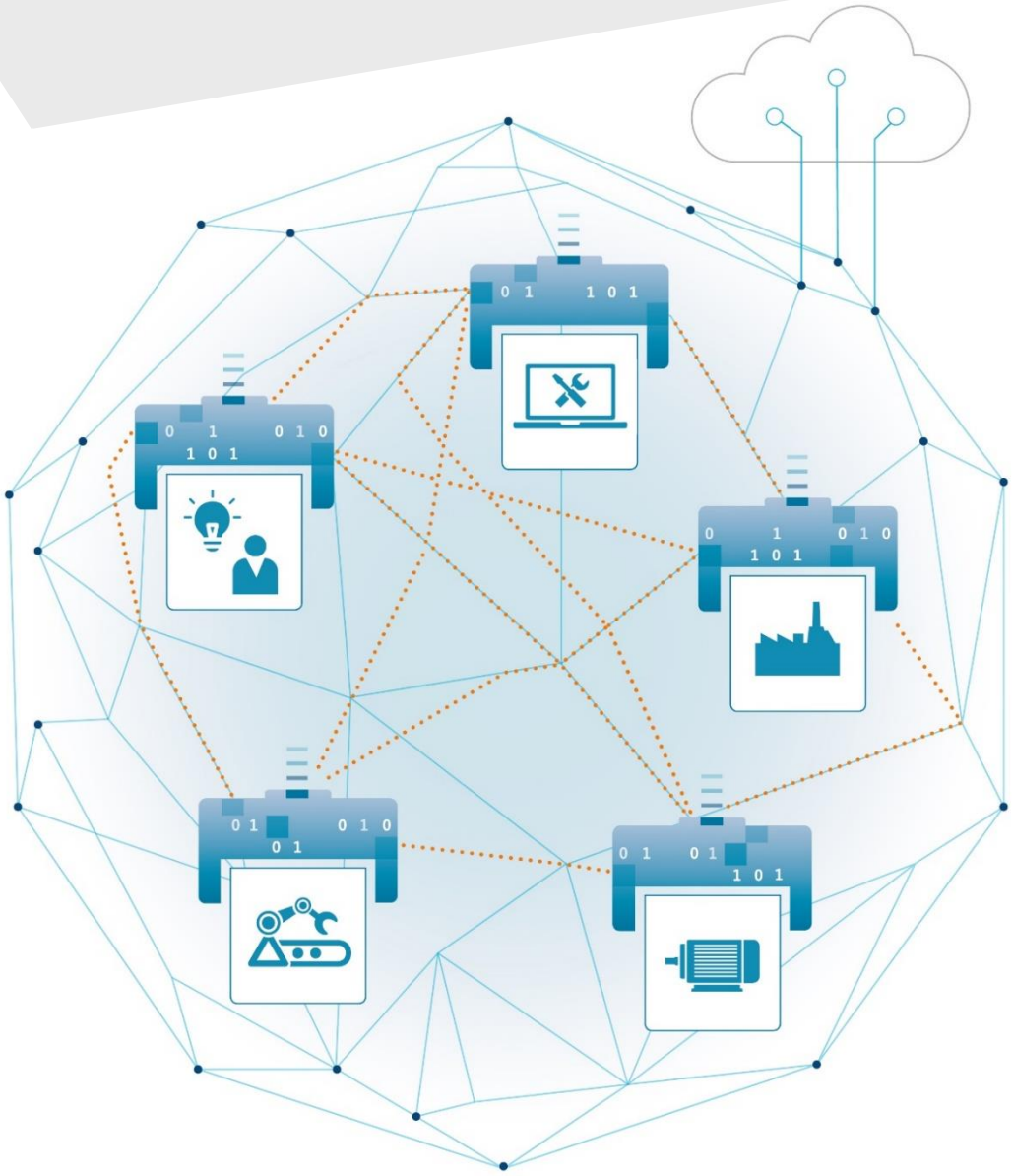
PLATTFORM INDUSTRIE4.0 in cooperation with ZVEI: Die Elektroindustrie

SPECIFICATION

Details of the Asset Administration Shell



Part 2 – Interoperability at Runtime – Exchanging Information via Application Programming Interfaces (Version 1.0RC02)

Imprint

Publisher

Federal Ministry for Economic Affairs
and Energy (BMWi)
Public Relations
10119 Berlin
www.bmwi.de

Text and editing

Plattform Industrie 4.0
Bülowstraße 78
10783 Berlin

Design and production

The Plattform Industrie 4.0 secretariat, Berlin

Status

November 2021 For Review

Illustrations

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

Contents

1 Preamble 9
1.1 Editorial notes 10
1.2 Scope of this Document 10
1.3 Structure of the Document 10
1.4 Terms & Definitions 10
1.5 Abbreviations 13
2 Introduction 15
3 General 17
3.1 Services, Interfaces and Interface Operations 18
3.2 Design Principles 20
3.3 Semantic References for Operations 22
3.4 References and Keys 24
3.5 Special Parameters 24
3.6 Relation of interfaces 24
4 Interfaces Asset Administration Shell 29
4.1 General 30
4.2 Asset Administration Shell Interface and Operations 30
4.2.1 Interface Asset Administration Shell 30
4.2.2 Operation GetAssetAdministrationShell 30
4.2.3 Operation PutAssetAdministrationShell 31
4.2.4 Operation GetAllSubmodelReferences 32
4.2.5 Operation PostSubmodelReference 32
4.2.6 Operation DeleteSubmodelReference 33
4.2.7 Operation GetAssetInformation 34
4.2.8 Operation PutAssetInformation 34
4.3 Submodel Interface and Operations 35
4.3.1 Interface Submodel 35

4.3.2 Operation GetSubmodel 36
4.3.3 Operation GetAllSubmodelElements 37
4.3.4 Operation GetSubmodelElementByPath 37
4.3.5 Operation PutSubmodel 38
4.3.6 Operation PostSubmodelElement 39
4.3.7 Operation PostSubmodelElementByPath 39
4.3.8 Operation PutSubmodelElementByPath 40
4.3.9 Operation SetSubmodelElementValueByPath 41
4.3.10 Operation DeleteSubmodelElementByPath 41
4.3.11 Operation InvokeOperationSync 42
4.3.12 Operation InvokeOperationAsync 43
4.3.13 Operation GetOperationAsyncResult 44
4.4 Asset Administration Shell Serialization Interface and Operations 44
4.4.1 Interface Asset Administration Shell Serialization 44
4.4.2 Operation GenerateSerializationByIds 45
4.5 AASX File Server Interface and Operations 45
4.5.1 Interface AASX File Server 46
4.5.2 Operation GetAllAASXPackageIds 46
4.5.3 Operation GetAASXByPackageId 47
4.5.4 Operation PostAASXPackage 47
4.5.5 Operation PutAASXPackageById 48
4.5.6 Operation DeleteAASXPackageById 49
5 Interfaces Registration and Lookup 50
5.1 General 51
5.2 Asset Administration Shell Registry Interface and Operations 51
5.2.1 Interface Asset Administration Shell Registry 51
5.2.2 Operation GetAllAssetAdministrationShellDescriptors 51
5.2.3 Operation GetAssetAdministrationShellDescriptorById 52
5.2.4 Operation PostAssetAdministrationShellDescriptor 53

5.2.5 Operation PutAssetAdministrationShellDescriptorById 53
5.2.6 Operation DeleteAssetAdministrationShellDescriptorById 54
5.3 Submodel Registry Interface and Operations 55
5.3.1 Interface Submodel Registry 55
5.3.2 Operation GetAllSubmodelDescriptors 55
5.3.3 Operation GetSubmodelDescriptorById 56
5.3.4 Operation PostSubmodelDescriptor 57
5.3.5 Operation PutSubmodelDescriptorById 57
5.3.6 Operation DeleteSubmodelDescriptorById 58
6 Interfaces Repository 59
6.1 General 60
6.2 Asset Administration Shell Repository Interface and Operations 60
6.2.1 Interface Asset Administration Shell Repository 60
6.2.2 Operation GetAllAssetAdministrationShells 61
6.2.3 Operation GetAssetAdministrationShellById 61
6.2.4 Operation GetAllAssetAdministrationShellsByAssetId 62
6.2.5 Operation GetAllAssetAdministrationShellsByIdShort 63
6.2.6 Operation PostAssetAdministrationShell 63
6.2.7 Operation PutAssetAdministrationShellById 64
6.2.8 Operation DeleteAssetAdministrationShellById 65
6.3 Submodel Repository Interface and Operations 65
6.3.1 Interface Submodel Repository 65
6.3.2 Operation GetAllSubmodels 66
6.3.3 Operation GetSubmodelById 67
6.3.4 Operation GetAllSubmodelsBySemanticId 67
6.3.5 Operation GetAllSubmodelsByIdShort 68
6.3.6 Operation PostSubmodel 69
6.3.7 Operation PutSubmodelById 69
6.3.8 Operation DeleteSubmodelById 70

6.4 Concept Description Repository Interface and Operations 71
6.4.1 Interface Concept Description Repository 71
6.4.2 Operation GetAllConceptDescriptions 71
6.4.3 Operation GetConceptDescriptionById 72
6.4.4 Operation GetAllConceptDescriptionsByIdShort 73
6.4.5 Operation GetAllConceptDescriptionsByIsCaseOf 73
6.4.6 Operation GetAllConceptDescriptionsByDataSpecificationReference 74
6.4.7 Operation PostConceptDescription 75
6.4.8 Operation PutConceptDescriptionById 75
6.4.9 Operation DeleteConceptDescriptionById 76
7 Interfaces Publish and Discovery 77
7.1 General 78
7.2 Asset Administration Shell Basic Discovery Interface and Operations 78
7.2.1 Interface Asset Administration Shell Basic Discovery 78
7.2.2 Operation GetAllAssetAdministrationShellIdsByAssetLink 78
7.2.3 Operation GetAllAssetLinksById 79
7.2.4 Operation PostAllAssetLinksById 80
7.2.5 Operation DeleteAllAssetLinksById 81
8 Data Types for Payload 82
8.1 General 83
8.2 Metamodel Specification Details: Designators 83
8.2.1 Descriptor 83
8.2.2 AssetAdministrationShellDescriptor 83
8.2.3 SubmodelDescriptor 84
8.2.4 Endpoint 85
8.2.5 ProtocolInformation 87
8.2.6 Status Code, Error Handling & Result Messages 88
8.2.7 Generic Status Codes 88
9 Basic Operation Parameters 94

9.1 General ... 95
9.2 Output Modifiers in Operations ... 95
9.3 Applicability of the Output Modifiers ... 96
9.4 Serialization in Specified Formats (Output Modifier *Content*) ... 97
9.4.1 General ... 97
9.4.2 ValueOnly-Serialization in JSON ... 97
9.4.3 JSON-Schema for the ValueOnly-Serialization ... 108
9.4.4 IdShortPath serialization ... 116
10 HTTP/REST API ... 117
10.1 General ... 118
10.2 Design Decisions ... 118
10.3 Addressing Resources ... 120
10.4 Trimmed Objects ... 122
10.5 Payload ... 124
10.6 Modifiers ... 124
10.7 Mapping of Operations ... 127
10.8 Mapping of Status Codes ... 132
10.9 Additional Data Types for Payload specific for HTTP/REST ... 133
10.9.1 AssetAdministrationShellEnvironment ... 133
10.9.2 PackageDescription ... 133
10.1 Interactions ... 134
10.2 Security ... 136
11 Summary and Outlook ... 138
Annex A. Templates Used for Specification ... 141
Annex B. Legend for UML Modelling ... 146
i. OMG UML General ... 146
ii. Notes to Graphical Representation ... 151
Annex C. ValueOnly-Serialization Example ... 155
Annex D. Bibliography ... 158

Annex E. Change Notes 159
1. General 159
2. Interface Changes w.r.t. V1.0RC01 159
3. Operation Changes w.r.t. V1.0RC01 161

1 Preamble

1.1 Editorial notes

This document was developed from November 2020 to November 2021 by the joint 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“.

Version 1.0RC01 of 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: Security elements and Views are not yet considered in this version of the interfaces and API specification of the Asset Administration Shell.

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 9. General topics are discussed in the Clause before, in Clause 3.

In Clause 10 the API specification for HTTP/REST is defined. Annex C gives an example for the ValueOnly serialization of the payload.

Clause 11 gives a summary and outlook.

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 thisdocument. 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 Industrie 4.0 component in an Industrie 4.0 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
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

Abbreviation	Description
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
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.

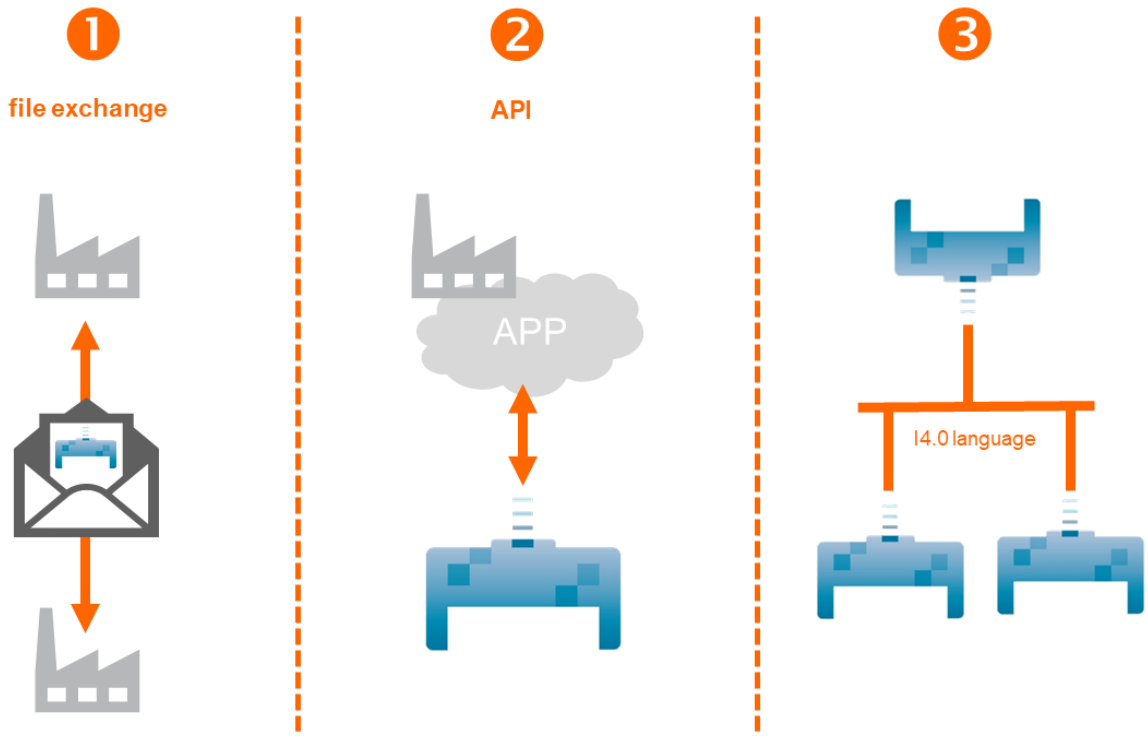
2 Introduction

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.

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: Interface operations define interaction patterns via the specified interface

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.

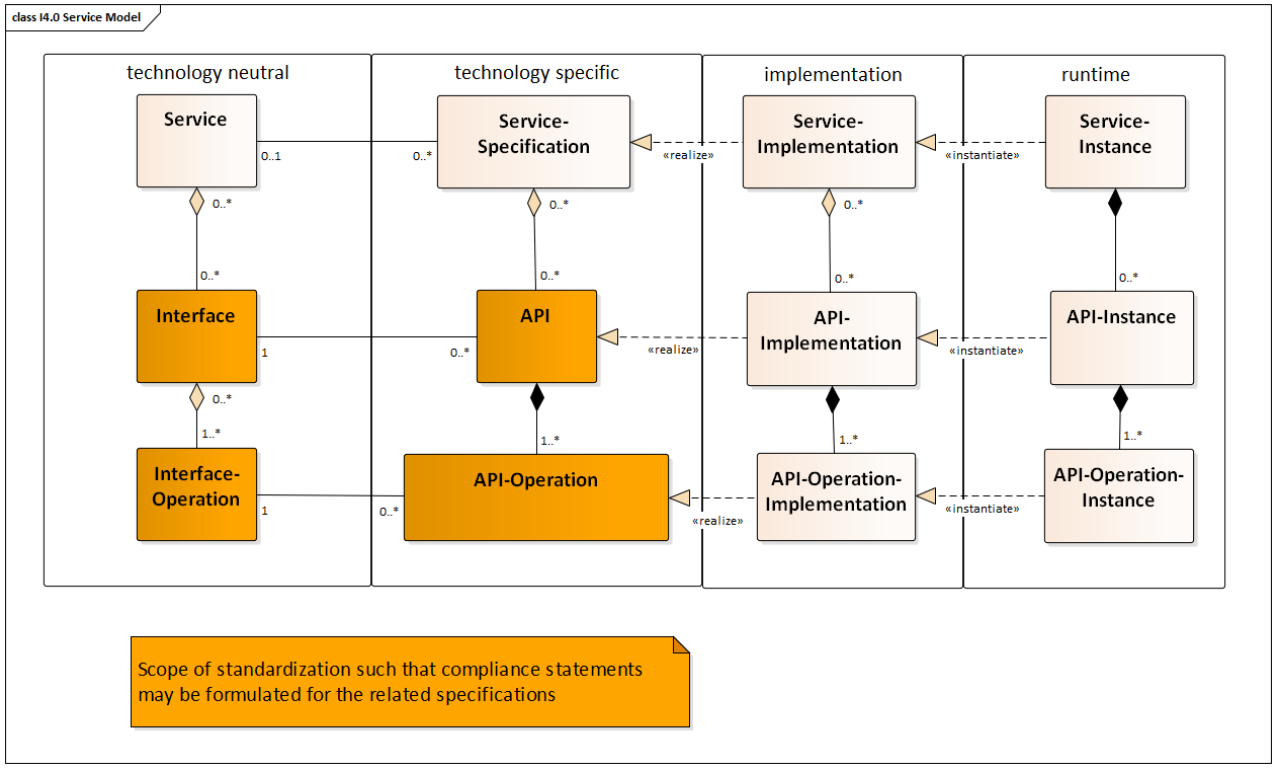
The implementation level comprises the following concepts:

- 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 metamodel 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 10.8.
- Methods (Verbs)
  A small set of standard REST methods which are GET, POST, PUT and DELETE is used to describe the semantic of the most common operations. There are only a few exceptions for methods for situations where the standard methods do not fit (e.g. GETALL, SET, INVOKE).

The methods are:

- GET
  A GET returns a single resource based on the resource identifier which is the identifier ([2]) for identifiables and the idShortPath for referables.
- GETALL
  Returns a list of resources based on optional available parameters such as filters.
- POST
  Creates a new resource. The identifier of the resource is 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 post the same AAS object twice it will not create two AAS resources.
- PUT
  Updates an existing resource.
- DELETE
  Deletes a resource based on a given identifier.
- SET
  Sets the value of an object, e.g. the value of a Property
- INVOKE
  Invokes an operation at a specified path

Naming rules for operations:

For the operation names in Asset Administration Shell Interface, Submodel Interface, Shell Repository Interface, Submodel Repository Interface, Concept Description Repository Interface the following rules shall apply:

<Interface Operation> ::= <Method Verb><Model Element Name>[<Modifier>]
[By <By-Qualifier>]

```
<Method Verb> ::= Get | GetAll | Put | Post | Delete | Set | Invoke

<Model Element Name> ::= AssetAdministrationShell[s] |
SubmodelReference[s] | AssetInformation | Submodel[s] |
SubmodelElement[s] | ConceptDescription[s]

<Modifier> ::= Value | IdShortPath | Reference

<By-Qualifier> ::= Id  | SemanticId | ParentPathAndSemanticId | Path |
AssetId | IdShort | IsCaseOf | DataSpecificationReference
```

Examples:

GetSubmodel has method verb “Get” and Element Name “Submodel”.

GetAllSubmodelElementsBySemanticId has method verb “GetAll” and Element Name “SubmodelElements” plus a By-Qualifier “SemanticId”.

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) [7] 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.

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

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 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/'<OperationName>'/'<Version>

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

<OperationName>   ::= <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 *interfaceName* and the *operationName* are 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/RC02

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/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
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*.
SemanticId	Identifier of the semantic definition

3.6 Relation of interfaces

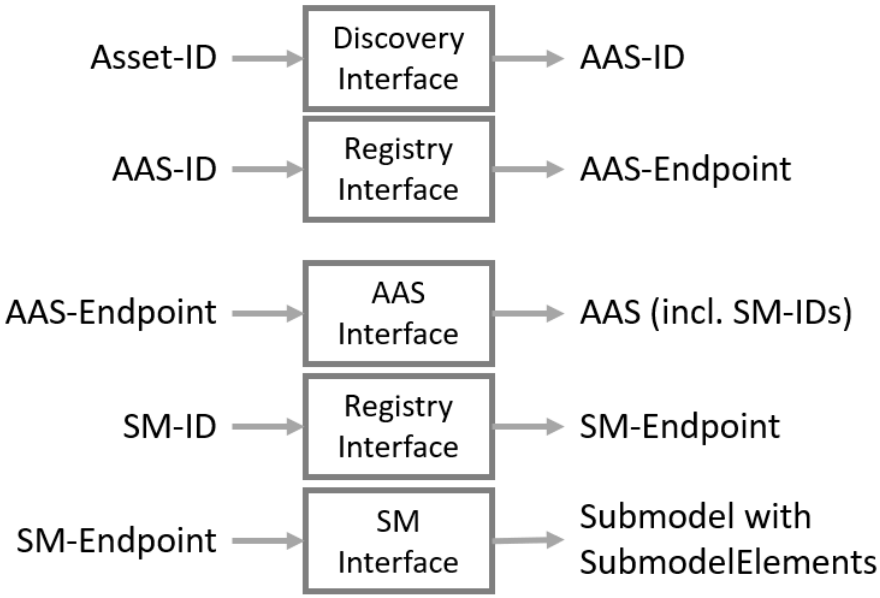
In the following chapters several interfaces are defined, which work together as a system, and which support different deployment scenarios.

There are 3 major components of the overall system:

1. Repositories store the data of AAS, submodels and concept descriptions,
2. Registries are “directories” which store AAS-IDs and Submodel-IDs together with the related endpoints (typically an URL-path into a repository or to a single AAS/Submodel),
3. Discovery (servers) support a fast search and only store copies of essential information, i.e. key value pairs to find IDs by other IDs.

Figure 3 shows a typical sequence. Discovery finds the AAS-ID for a given Asset-ID. A Registry provides the endpoint for a given AAS-ID. By such endpoint for an AAS and the related Submodel-IDs the submodels with their submodelElements can be accessed.

Figure 3 Retrieval of asset related information by AAS and Submodels



The Asset Administration Shell model is an asset oriented model.

An Asset-ID may be retrieved e.g. by a QRCODE on the asset, by an RFID for the asset, from the firmware of the asset or from an asset database. IEC 61406 (formerly DIN SPEC 91406) defines the format of such Asset-IDs.

With an Asset-ID the “Administration Shell Basic Discovery Interface” may be used to get the related AAS-IDs (“GetAllAssetAdministrationShellIdsByAssetLink”).

With an AAS-ID the “Asset Administration Shell Registry Interface” may be used to retrieve the related descriptor for an AAS (“GetAssetAdministrationShellDescriptorById”). The retrieved AAS Descriptor includes the endpoint for the “Asset Administration Shell Interface”.

With the “Asset Administration Shell Interface” the information about the AAS itself and the references to the related submodels are available.

The related submodels of an AAS are retrieved by “GetAllSubmodelReferences”. Such reference includes the SM-ID of a related submodel.

Similarly to the AAS above, the “Submodel Registry Interface” may be used to retrieve the related descriptor for a submodel (“GetSubmodelDescriptorById”) with a specific SM-ID. The retrieved Submodel Descriptor includes the endpoint for the “Submodel Interface”.

With the “Submodel Interface” the information about the submodel itself and about all its included submodel elements is available.

Asset Administration Shells and submodels may be deployed on different endpoints in different ways.

One deployment example is the deployment of an AAS on a device. In such case the AAS might be fixed and might not be changed or deleted. In a cloud scenario a single AAS may also be deployed as a single container (e.g. docker container) similarly.

Another deployment example is the deployment of many AAS in an AAS repository. In such case the “Asset Administration Shell Repository Interface” may allow to create and manage multiple AAS in the repository.

The separate interfaces of the HTTP/REST API allow many different ways to support such different deployments. A later version of this specification will define related profiles.

For an AAS repository the combination “Asset Administration Shell Repository Interface”, “Submodel Repository Interface”, “Concept Description Repository Interface”, “Asset Administration Shell Interface”, “Submodel Interface” and “Asset Administration Shell Serialization Interface” is proposed.

This will result in the following HTTP/REST paths as described in the related swagger (AssetAdministrationShell-Repository | Final-Draft | Plattform_i40 | SwaggerHub):

/shells

/shells/{aas-identifier}

/shells/{aas-identifier}/aas

/shells/{aas-identifier}/aas/asset-information

/shells/{aas-identifier}/aas/submodels

/shells/{aas-identifier}/aas/submodels/{submodel-identifier}

/shells/{aas-identifier}/aas/submodels/{submodel-identifier}/submodel

/shells/{aas-identifier}/aas/submodels/{submodel-identifier}/submodel/submodel-elements

/shells/{aas-identifier}/aas/submodels/{submodel-identifier}/submodel/submodel-elements/{idShortPath}

/shells/{aas-identifier}/aas/submodels/{submodel-identifier}/submodel/submodel-elements/{idShortPath}/invoke

/shells/{aas-identifier}/aas/submodels/{submodel-identifier}/submodel/submodel-elements/{idShortPath}/operation-results/{handleId}

/submodels

/submodels/{submodel-identifier}/submodel

/submodels/{submodel-identifier}/submodel/submodel-elements

/submodels/{submodel-identifier}/submodel/submodel-elements/{idShortPath}

/submodels/{submodel-identifier}/submodel/submodel-elements/{idShortPath}/invoke

/submodels/{submodel-identifier}/submodel/submodel-elements/{idShortPath}/operation-results/{handleId}

/concept-descriptions

/concept-descriptions/{cd-identifier}

/serialization

If the repository also supports AASX Packages it shall be extended by the “AASX File Server Interface”.

The example of a device or container containing 1 AAS with its related submodels will result in the following HTTP/REST paths as described in the related swagger (AssetAdministrationShell-Standalone | Final-Draft | Plattform_i40 | SwaggerHub):

/aas

/aas/asset-information

/aas/submodels

/aas/submodels/{submodel-identifier}

/aas/submodels/{submodel-identifier}/submodel

/aas/submodels/{submodel-identifier}/submodel/submodel-elements

/aas/submodels/{submodel-identifier}/submodel/submodel-elements/{idShortPath}

/aas/submodels/{submodel-identifier}/submodel/submodel-elements/{idShortPath}/invoke

/aas/submodels/{submodel-identifier}/submodel/submodel-elements/{idShortPath}/operation-results/{handleId}

/serialization

4 Interfaces Asset Administration Shell

4.1 General

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

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
GetAllSubmodelReferences	Returns all Submodel References
PostSubmodelReference	Creates a Submodel Reference at the Asset Administration Shell
DeleteSubmodelReference	Deletes a specific Submodel Reference from the Asset Administration Shell
GetAssetInformation	Returns the Asset Information
PutAssetInformation	Updates the Asset Information

4.2.2 Operation GetAssetAdministrationShell

Operation Name	GetAssetAdministrationShell	
Explanation	Returns the Asset Administration Shell	
semanticId	https://admin-shell.io/aas/API/GetAssetAdministrationShell/1/0/RC02	
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/RC02	
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 GetAllSubmodelReferences

Operation Name	GetAllSubmodelReferences	
Explanation	Returns all Submodel References	
semanticId	https://admin-shell.io/aas/API/GetAllSubmodelReferences/1/0/RC02	
Name	Type	Description
Input Parameter		
outputModifier	OutputModifier	Determines the result format filtering of the response
Output Parameter		
statusCode	StatusCode	Status code
payload	References	Requested Submodel References

4.2.5 Operation PostSubmodelReference

Operation Name	PostSubmodelReference	
Explanation	Creates a Submodel Reference at the Asset Administration Shell	
semanticId	https://admin-shell.io/aas/API/PostSubmodelReference/1/0/RC02	
Name	Type	Description
Input Parameter		
submodelRef	Reference	Reference to the Submodel

Operation Name	PostSubmodelReference	
Output Parameter		
statusCode	StatusCode	Status code
payload	Reference	Created Submodel Reference

4.2.6 Operation DeleteSubmodelReference

Operation Name	DeleteSubmodelReference	
Explanation	Deletes the Submodel Reference from the Asset Administration Shell	
semanticId	https://admin-shell.io/aas/API/DeleteSubmodelReference/1/0/RC02	
Name	Type	Description
Input Parameter		
submodelId	Identifier	The unique id of the Submodel for the reference to be deleted
Output Parameter		
statusCode	StatusCode	Status code

4.2.7 Operation GetAssetInformation

Operation Name	GetAssetInformation	
Explanation	Returns the Asset Information	
semanticId	https://admin-shell.io/aas/API/GetAssetInformation/1/0/RC02	
Name	Type	Description
Input Parameter		
Output Parameter		
statusCode	StatusCode	Status code
payload	AssetInformation	Requested Asset Information

4.2.8 Operation PutAssetInformation

Operation Name	PutAssetInformation	
Explanation	Updates the Asset Information	
semanticId	https://admin-shell.io/aas/API/PutAssetInformation/1/0/RC02	
Name	Type	Description
Input Parameter		
assetInfo	AssetInformation	Asset Information object
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
GetSubmodelElementByPath	Returns a specific submodel element from the Submodel at a specified path
PutSubmodel	Updates the Submodel
PostSubmodelElement	Creates a new submodel element as a child of the submodel. The idShort of the the new submodel element must be set in the payload. Note: The creation of the idshort is out of scope and has to be done with an external (company-specific) service.
PostSubmodelElementByPath	Creates a new submodel element at a specified path within the submodel elements hierarchy. The idShort of the the new submodel element must be set in the payload. Note: The creation of the idShort is out of scope and has to be done with an external (company-specific) service.
PutSubmodelElementByPath	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

Interface: Submodel	
Operation Name	Description
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/RC02	
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/RC02	
Name	Type	Description
Input Parameter		
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 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/RC02	
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

4.3.5 Operation PutSubmodel

Operation Name	PutSubmodel	
Explanation	Updates the Submodel	
semanticId	https://admin-shell.io/aas/API/PutSubmodel/1/0/RC02	
Name	Type	Description
Input Parameter		
submodel	Submodel	Submodel object
Output Parameter		
statusCode	StatusCode	Status code
payload	Submodel	Updated submodel

4.3.6 Operation PostSubmodelElement

Operation Name	PostSubmodelElement	
Explanation	Creates a new submodel element as a child of the submodel. The idShort of the the new submodel element must be set in the payload. Note: The creation of the idShort is out of scope and has to be done with an external (company-specific) service.	
semanticId	https://admin-shell.io/aas/API/PostSubmodelElement/1/0/RC02	
Name	Type	Description
Input Parameter		
submodelElement	SubmodelElement	Submodel element object
Output Parameter		
statusCode	StatusCode	Status code
payload	SubmodelElement	Created submodel element

4.3.7 Operation PostSubmodelElementByPath

Operation Name	PostSubmodelElementByPath	
Explanation	Creates a new submodel element at a specified path within the submodel elements hierarchy. The idShort of the the new submodel element must be set in the payload. Note: The creation of the idShort is out of scope and has to be done with an external (company-specific) service.	
semanticId	https://admin-shell.io/aas/API/PostSubmodelElementByPath/1/0/RC02	
Name	Type	Description
Input Parameter		

Operation Name	PostSubmodelElementByPath	
path	Key[0..*]	IdShort-Path via relative Reference/Keys to a submodel element.
submodelElement	SubmodelElement	Submodel element object
Output Parameter		
statusCode	StatusCode	Status code
payload	SubmodelElement	Created submodel element

4.3.8 Operation PutSubmodelElementByPath

Operation Name	PutSubmodelElementByPath	
Explanation	Updates an existing submodel element at a specified path within the submodel elements hierarchy	
semanticId	https://admin-shell.io/aas/API/PutSubmodelElementByPath/1/0/RC02	
Name	Type	Description
Input Parameter		
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	Updated submodel element

4.3.9 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/RC02	
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

4.3.10 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/RC02	
Name	**Type**	**Description**
Input Parameter		

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

4.3.11 Operation InvokeOperationSync

Operation Name	InvokeOperationSync	
Explanation	Synchronously invokes an Operation at a specified path	
semanticId	https://admin-shell.io/aas/API/InvokeOperationSync/1/0/RC02	
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
timestamp	DateTime (UTC)[2]	Timestamp until when the client expects the server to have finished execution of the invoked operation
requestId	string	Client request id
Output Parameter		

[2] see RFC 3339 (https://datatracker.ietf.org/doc/html/rfc3339)

statusCode	StatusCode	Status code
payload	OperationResult	Operation Result

4.3.12 Operation InvokeOperationAsync

Operation Name	InvokeOperationAsync	
Explanation	Asynchronously invokes an Operation at a specified path	
semanticId	https://admin-shell.io/aas/API/InvokeOperationAsync/1/0/RC02	
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
timestamp	DateTime (UTC)[3]	Timestamp until when the client expects the server to have finished execution of the invoked operation
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

[3] see RFC 3339 (https://datatracker.ietf.org/doc/html/rfc3339)

4.3.13 Operation GetOperationAsyncResult

Operation Name	GetOperationAsyncResult	
Explanation	Returns the OperationResult of an asynchronously invoked operation	
semanticId	https://admin-shell.io/aas/API/GetOperationAsnycResult/1/0/RC02	
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
GenerateSerializationByIds	Returns an appropriate serialization based on the specified format (see SerializationFormat).

4.4.2 Operation GenerateSerializationByIds

Operation Name	GenerateSerializationByIds	
Explanation	Returns an appropriate serialization based on the specified format (see SerializationFormat).	
semanticId	https://admin-shell.io/aas/API/GenerateSerializationByIds/1/0/RC02	
Name	**Type**	**Description**
Input Parameter		
aasIds	Identifier[0..*]	The unique ids of the Asset Administration Shells to be contained in the serialization
submodelIds	Identifier[0..*]	The unique ids of the Submodels to be contained in the serialization
includeConceptDescriptions	boolean	Include concept descriptions
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

4.5 AASX File Server Interface and Operations

4.5.1 Interface AASX File Server

Interface: AASX File Server	
Operation Name	Description
GetAllAASXPackageIds	Returns a list of available AASX packages at the server
GetAASXByPackageId	Returns a specific AASX package from the server
PostAASXPackage	Creates an AASX package at the server
PutAASXByPackageId	Updates the AASX package at the server
DeleteAASXByPackageId	Deletes a specific AASX package

4.5.2 Operation GetAllAASXPackageIds

Operation Name	GetAllAASXPackageIds	
Explanation	Returns a list of available AASX packages at the server	
semanticId	https://admin-shell.io/aas/API/GetAllAASXPackageIds/1/0/RC02	
Name	Type	Description
Input Parameter		
aasId	Identifier[0..1]	List of AAS Ids which all must be in each matching AASX package
Output Parameter		
statusCode	StatusCode	Status code
payload	PackageDescription[0...*]	Matching package list

4.5.3 Operation GetAASXByPackageId

Operation Name	GetAASXByPackageId	
Explanation	Returns a specific AASX package from the server	
semanticId	https://admin-shell.io/aas/API/GetAASXByPackageId /1/0/RC02	
Name	Type	Description
Input Parameter		
packageId	string	Requested package ID from the package list
Output Parameter		
statusCode	StatusCode	Status code
filename	String	Filename of the AASX package
payload	AASX package	Requested AASX package

4.5.4 Operation PostAASXPackage

Operation Name	PostAASXPackage	
Explanation	Creates an AASX package at the server	
semanticId	https://admin-shell.io/aas/API/PostAASXPackage/1/0/RC02	
Name	Type	Description
Input Parameter		
aasIds	Identifier[0..*]	Included AAS Ids
file	AASX package	New AASX package

Operation Name	PostAASXPackage	
filename	String	Filename of the AASX package
Output Parameter		
statusCode	StatusCode	Status code
packageId	String	New Package ID

4.5.5 Operation PutAASXPackageById

Operation Name	PutAASXPackageById	
Explanation	Updates the AASX package at the server	
semanticId	https://admin-shell.io/aas/API/PutAASXPackageById/1/0/RC02	
Name	Type	Description
Input Parameter		
packageId	String	Package ID from the package list
aasIds	Identifier[0..*]	Included AAS Ids
file	AASX package	New AASX package
filename	String	Filename of the AASX package
Output Parameter		
statusCode	StatusCode	Status code

4.5.6 Operation DeleteAASXPackageById

Operation Name	DeleteAASXPackageById	
Explanation	Deletes a specific AASX package from the server	
semanticId	https://admin-shell.io/aas/API/DeleteAASXPackageById/1/0/RC02	
Name	Type	Description
Input Parameter		
packageId	String	Package ID from the package list
Output Parameter		
statusCode	StatusCode	Status code

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.

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
PostAssetAdministrationShellDescriptor	Creates a new Asset Administration Shell Descriptor, i.e. registers an AAS.
PutAssetAdministrationShellDescriptorById	Updates an existing Asset Administration Shell Descriptor, i.e. updates registration information.
DeleteAssetAdministrationShellDescriptorById	Deletes an Asset Administration Shell Descriptor, i.e. de-registers an AAS

5.2.2 Operation GetAllAssetAdministrationShellDescriptors

Operation Name	GetAllAssetAdministrationShellDescriptors
Explanation	Returns all Asset Administration Shell Descriptors

Operation Name	GetAllAssetAdministrationShellDescriptors	
semanticId	https://admin-shell.io/aas/API/GetAllAssetAdministrationShellDescriptors/1/0/RC02	
Name	Type	Description
Input Parameter		
Output Parameter		
statusCode	StatusCode	Status code
payload	AssetAdministrationShellDescriptor[0..*]	List of 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/RC02	
Name	Type	Description
Input Parameter		
aasIdentifier	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 PostAssetAdministrationShellDescriptor

Operation Name	PostAssetAdministrationShellDescriptor	
Explanation	Creates a new Asset Administration Shell Descriptor, i.e. registers an AAS	
semanticId	https://admin-shell.io/aas/API/PostAssetAdministrationShellDescriptor/1/0/RC02	
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 Asset Administration Shell Descriptor

5.2.5 Operation PutAssetAdministrationShellDescriptorById

Operation Name	PutAssetAdministrationShellDescriptorById	
Explanation	Updates an existing Asset Administration Shell Descriptor, i.e. updates registration information.	
semanticId	https://admin-shell.io/aas/API/PutAssetAdministrationShellDescriptorById/1/0/RC02	
Name	**Type**	**Description**
Input Parameter		
shellDescriptor	AssetAdministrationShellDescriptor	Object containing the Asset Administration Shell's identification and endpoint information

Operation Name	PutAssetAdministrationShellDescriptorById	
Output Parameter		
statusCode	StatusCode	Status code
payload	AssetAdministrationShellDescriptor	Updated Asset Administration Shell Descriptor

5.2.6 Operation DeleteAssetAdministrationShellDescriptorById

Operation Name	DeleteAssetAdministrationShellDescriptorById	
Explanation	Deletes an Asset Administration Shell Descriptor, i.e. de-registers an AAS	
semanticId	https://admin-shell.io/aas/API/DeleteAssetAdministrationShellDescriptorById/1/0/RC02	
Name	Type	Description
Input Parameter		
aasIdentifier	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
PostSubmodelDescriptor	Creates a new submodel descriptor, i.e. registers a submodel
PutSubmodelDescriptorById	Updates an existing submodel descriptor, i.e. updates registration information
DeleteSubmodelDescriptorById	Deletes a submodel descriptor, i.e. de-registers a submodel

5.3.2 Operation GetAllSubmodelDescriptors

Operation Name	GetAllSubmodelDescriptors	
Explanation	Returns all submodel descriptors	
semanticId	https://admin-shell.io/aas/API/GetAllSubmodelDescriptors/1/0/RC02	
Name	Type	Description
Input Parameter		
Output Parameter		
statusCode	StatusCode	Status code
payload	SubmodelDescriptor[0..*]	List of 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/RC02	
Name	Type	Description
Input Parameter		
submodelIdentifier	Identifier	The Submodel's unique id
Output Parameter		
statusCode	StatusCode	Status code
payload	SubmodelDescriptor	Requested submodel descriptor

5.3.4 Operation PostSubmodelDescriptor

Operation Name	PostSubmodelDescriptor	
Explanation	Creates a new submodel descriptor, i.e. registers a submodel	
semanticId	https://admin-shell.io/aas/API/PostSubmodelDescriptor/1/0/RC02	
Name	Type	Description
Input Parameter		
submodel Descriptor	SubmodelDescriptor	Object containing the Submodel's identification and endpoint information
Output Parameter		
statusCode	StatusCode	Status code
payload	SubmodelDescriptor	Created submodel descriptor

5.3.5 Operation PutSubmodelDescriptorById

Operation Name	PutSubmodelDescriptorById	
Explanation	Updates an existing submodel descriptor, i.e. updates registration information	
semanticId	https://admin-shell.io/aas/API/PutSubmodelDescriptorById/1/0/RC02	
Name	Type	Description
Input Parameter		
submodel Descriptor	SubmodelDescriptor	Object containing the Submodel's identification and endpoint information

Operation Name	PutSubmodelDescriptorById	
Output Parameter		
statusCode	StatusCode	Status code
payload	SubmodelDescriptor	Updated submodel descriptor

5.3.6 Operation DeleteSubmodelDescriptorById

Operation Name	DeleteSubmodelDescriptorById	
Explanation	Deletes a Submodel Descriptor, i.e. de-registers a submodel	
semanticId	https://admin-shell.io/aas/API/DeleteSubmodelDescriptorById/1/0/RC02	
Name	Type	Description
Input Parameter		
submodelIdentifier	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 Shells, submodels and and concept descriptions 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 (asset administration shells, submodels, concept descriptions) 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
GetAssetAdministrationShellById	Returns a specific Asset Administration Shell
GetAllAssetAdministrationShellsByAssetId	Returns all Asset Administration Shells that are linked to a globally unique asset identifier or to specific asset ids.
GetAllAssetAdministrationShellsByIdShort	Returns all Asset Administration Shells with a specific idShort
PostAssetAdministrationShell	Creates a new Asset Administration Shell. The id of the the new Asset Administration shell must be set in the payload. Note: The creation of the id is out of scope and has to be done with an external (company-specific) service.
PutAssetAdministrationShellById	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/RC02	
Name	Type	Description
Input Parameter		
outputModifier	OutputModifier	Determines the result format filtering of the response
Output Parameter		
statusCode	StatusCode	Status code
payload	AssetAdministrationShell[0..*]	List of Asset Administration Shells

6.2.3 Operation GetAssetAdministrationShellById

Operation Name	GetAssetAdministrationShellById	
Explanation	Returns a specific Asset Administration Shell	
semanticId	https://admin-shell.io/aas/API/GetAssetAdministrationShellById/1/0/RC02	
Name	Type	Description
Input Parameter		
id	Identifier	The Asset Administration Shell’s unique id

Operation Name	GetAssetAdministrationShellById	
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 Shells that are linked to a globally unique asset identifier or to specific asset ids.	
semanticId	https://admin-shell.io/aas/API/GetAllAssetAdministrationShellsByAssetId/1/0/RC02	
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
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/RC02	
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 PostAssetAdministrationShell

Operation Name	PostAssetAdministrationShell	
Explanation	Creates a new Asset Administration Shell. The id of the the new Asset Administration shell must be set in the payload. Note: The creation of the id is out of scope and has to be done with an external (company-specific) service.	
semanticId	https://admin-shell.io/aas/API/PostAssetAdministrationShell/1/0/RC02	
Name	Type	Description
Input Parameter		

Operation Name	PostAssetAdministrationShell	
aas	AssetAdministrationShell	Asset Administration Shell object
Output Parameter		
statusCode	StatusCode	Status code
payload	AssetAdministrationShell	Created Asset Administration Shell

6.2.7 Operation PutAssetAdministrationShellById

Operation Name	PutAssetAdministrationShellById	
Explanation	Updates an existing Asset Administration Shell	
semanticId	https://admin-shell.io/aas/API/PutAssetAdministrationShellById/1/0/RC02	
Name	Type	Description
Input Parameter		
aas	AssetAdministrationShell	Asset Administration Shell object
Output Parameter		
statusCode	StatusCode	Status code
payload	AssetAdministrationShell	Updated Asset Administration Shell

6.2.8 Operation DeleteAssetAdministrationShellById

Operation Name	DeleteAssetAdministrationShellById	
Explanation	Deletes an Asset Administration Shell	
semanticId	https://admin-shell.io/aas/API/DeleteAssetAdministrationShellById/1/0/RC02	
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
GetSubmodelById	Returns a specific Submodel
GetAllSubmodelsBySemanticId	Returns all Submodels with a specific SemanticId
GetAllSubmodelsByIdShort	Returns all Submodels with a specific *idShort*
PostSubmodel	Creates a new Submodel. The id of the the new submodel must be set in the payload.

Interface: Submodel Repository	
	Note: The creation of the id is out of scope and has to be done with an external (company-specific) service.
PutSubmodelById	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/RC02	
Name	Type	Description
Input Parameter		
outputModifier	OutputModifier	Determines the result format filtering of the response
Output Parameter		
statusCode	StatusCode	Status code
payload	Submodel[0..*]	List of Submodels

6.3.3 Operation GetSubmodelById

Operation Name	GetSubmodelById	
Explanation	Returns a specific Submodel	
semanticId	https://admin-shell.io/aas/API/GetSubmodelById/1/0/RC02	
Name	Type	Description
Input Parameter		
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/RC02	
Name	Type	Description
Input Parameter		
semanticId	Reference	Identifier of the semantic definition

Operation Name	GetAllSubmodelsBySemanticId	
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/RC02	
Name	Type	Description
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 PostSubmodel

Operation Name	PostSubmodel	
Explanation	Creates a new Submodel. The id of the the new submodel must be set in the payload. Note: The creation of the id is out of scope and has to be done with an external (company-specific) service.	
semanticId	https://admin-shell.io/aas/API/PostSubmodel/1/0/RC02	
Name	Type	Description
Input Parameter		
submodel	Submodel	Submodel object
Output Parameter		
statusCode	StatusCode	Status code
payload	Submodel	Created Submodel

6.3.7 Operation PutSubmodelById

Operation Name	PutSubmodelById	
Explanation	Updates an existing Submodel	
semanticId	https://admin-shell.io/aas/API/PutSubmodelById/1/0/RC02	
Name	Type	Description
Input Parameter		
submodel	Submodel	Submodel object

Output Parameter		
statusCode	StatusCode	Status code
payload	Submodel	Updated Submodel

6.3.8 Operation DeleteSubmodelById

Operation Name	DeleteSubmodelById	
Explanation	Deletes a Submodel	
semanticId	https://admin-shell.io/aas/API/DeleteSubmodelById/1/0/RC02	
Name	Type	Description
Input Parameter		
id	Identifier	The Submodel's unique id
Output Parameter		
statusCode	StatusCode	Status code

6.4 Concept Description Repository Interface and Operations

6.4.1 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
GetAllConceptDescriptionsByDataSpecificationReference	Returns all Concept Descriptions with a specific *dataSpecification* reference
PostConceptDescription	Creates a new Concept Description. The id of the the new Concept Description must be set in the payload. Note: The creation of the id is out of scope and has to be done with an external (company-specific) service.
PutConceptDescriptionById	Updates an existing Concept Description
DeleteConceptDescriptionById	Deletes a Concept Description

6.4.2 Operation GetAllConceptDescriptions

Operation Name	GetAllConceptDescriptions
Explanation	Returns all Concept Descriptions
semanticId	https://admin-shell.io/aas/API/GetAllConceptDescriptions/1/0/RC02

Operation Name	GetAllConceptDescriptions	
Name	Type	Description
Input Parameter		
outputModifier	OutputModifier	Determines the result format filtering of the response
Output Parameter		
statusCode	StatusCode	Status code
payload	ConceptDescription[0..*]	List of Concept Descriptions

6.4.3 Operation GetConceptDescriptionById

Operation Name	GetConceptDescriptionById	
Explanation	Returns a specific Concept Description	
semanticId	https://admin-shell.io/aas/API/GetConceptDescriptionById/1/0/RC02	
Name	Type	Description
Input Parameter		
cdIdentifier	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

6.4.4 Operation GetAllConceptDescriptionsByIdShort

Operation Name	GetAllConceptDescriptionsByIdShort	
Explanation	Returns all Concept Descriptions with a specific *idShort*	
semanticId	https://admin-shell.io/aas/API/GetAllConceptDescriptionsByIdShort/1/0/RC02	
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

6.4.5 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/RC02	
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

6.4.6 Operation GetAllConceptDescriptionsByDataSpecificationReference

Operation Name	GetAllConceptDescriptionsByDataSpecificationReference	
Explanation	Returns all Concept Descriptions with a specific *dataSpecification* reference	
semanticId	https://admin-shell.io/aas/API/GetAllConceptDescriptionsByDataSpecificationReference/1/0/RC02	
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

6.4.7 Operation PostConceptDescription

Operation Name	PostConceptDescription	
Explanation	Creates a new Concept Description. The id of the the new Concept Description must be set in the payload. Note: The creation of the id is out of scope and has to be done with an external (company-specific)	
semanticId	https://admin-shell.io/aas/API/PostConceptDescription/1/0/RC02	
Name	Type	Description
Input Parameter		
conceptDescription	ConceptDescription	Concept Description object
Output Parameter		
statusCode	StatusCode	Status code
payload	ConceptDescription	Created Concept Description

6.4.8 Operation PutConceptDescriptionById

Operation Name	PutConceptDescriptionById	
Explanation	Updates an existing Concept Description	
semanticId	https://admin-shell.io/aas/API/PutConceptDescriptionById/1/0/RC02	
Name	Type	Description
Input Parameter		
conceptDescription	ConceptDescription	Concept Description object

Operation Name	PutConceptDescriptionById	
Output Parameter		
statusCode	StatusCode	Status code
payload	ConceptDescription	Updated Concept Description

6.4.9 Operation DeleteConceptDescriptionById

Operation Name	DeleteConceptDescriptionById	
Explanation	Deletes a Concept Description	
semanticId	https://admin-shell.io/aas/API/DeleteConceptDescriptionById/1/0/RC02	
Name	Type	Description
Input Parameter		
cdIdentifier	Identifier	The Concept Description’s unique id
Output Parameter		
statusCode	StatusCode	Status code

7 Interfaces Publish and Discovery

7.1 General

These interfaces allow to publish information about asset administration shells that allow a search for asset IDs of the corresponding asset administration shells in a subsequent discovery interface call.

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**
GetAllAssetAdministrationShellIdsByAssetLink	Returns a list of Asset Administration Shell ids based on Asset identifier key-value-pairs
GetAllAssetLinksById	Returns a list of Asset identifier key-value-pairs based on an given Asset Administration Shell id
PostAllAssetLinksById	Creates or updates all Asset identifier key-value-pairs linked to an Asset Administration Shell to edit discoverable content
DeleteAllAssetLinksById	Deletes all Asset identifier key-value-pair linked to an Asset Administration Shell

7.2.2 Operation GetAllAssetAdministrationShellIdsByAssetLink

Operation Name	GetAllAssetAdministrationShellIdsByAssetLink	
Explanation	Returns a list of Asset Administration Shell ids based on Asset identifier key-value-pair	
semanticId	https://admin-shell.io/aas/API/GetAllAssetAdministrationShellIdsByAssetLink/1/0/RC02	
Name	**Type**	**Description**
Input Parameter		

Operation Name	GetAllAssetAdministrationShellIdsByAssetLink	
assetIdentifierPairs	IdentifierKeyValuePair[1..*]	The key-value-pair of an Asset identifier, which could be the globalAssetId or specificAssetIds. Note: The key of the Asset identifier key-value-pair for the globalAssetId is defined in chapter 3.5. It is the predefined key "*globalAssetId*" that would refer to the *AssetInformation/globalAssetId*.
Output Parameter		
statusCode	StatusCode	Status code
payload	Identifier[0..*]	Requested Asset Administration Shell Identifiers

7.2.3 Operation GetAllAssetLinksById

Operation Name	GetAllAssetLinksById	
Explanation	Returns a list of Asset identifier key-value-pairs based on an Asset Administration Shell id to edit discoverable content	
semanticId	https://admin-shell.io/aas/API/GetAllAssetLinksById/1/0/RC02	
Name	Type	Description
Input Parameter		
aasIdentifier	string	The Asset Administration Shell's unique id
Output Parameter		
statusCode	StatusCode	Status code
payload	IdentifierKeyValuePair	Requested Asset identifier key-value-pairs, which could be the globalAssetId or specificAssetIds. Note: The key of the key value pair for the globalAssetId is defined in chapter 3.5. It is the predefined key "*globalAssetId*" that would refer to the *AssetInformation/globalAssetId*.

7.2.4 Operation PostAllAssetLinksById

Operation Name	PostAllAssetLinksById	
Explanation	Creates new Asset identifier key-value-pairs linked to an Asset Administration Shell for discoverable content. It may be needed to delete the existing content first.	
semanticId	https://admin-shell.io/aas/API/PostAllAssetLinksById/1/0/RC02	
Name	Type	Description
Input Parameter		
aasIdentifier	string	The Asset Administration Shell's unique id
assetLinks	IdentifierKeyValuePair	Asset identifier key-value-pairs, which could be the globalAssetId or specificAssetIds. Note: The key of the key value pair for the globalAssetId is defined in chapter 3.5. It is the predefined key "*globalAssetId*" that would refer to the *AssetInformation/globalAssetId*.
Output Parameter		
statusCode	StatusCode	Status code
payload	IdentifierKeyValuePair	Asset identifier key-value-pairs created successfully

7.2.5 Operation DeleteAllAssetLinksById

Operation Name	DeleteAllAssetLinksById	
Explanation	Deletes all Asset identifier key-value-pair linked to an Asset Administration Shell to edit discoverable content	
semanticId	https://admin-shell.io/aas/API/DeleteAllAssetLinksById/1/0/RC02	
Name	Type	Description
Input Parameter		
aasIdentifier	string	The Asset Administration Shell’s unique id
Output Parameter		
statusCode	StatusCode	Status code

8 Data Types for Payload

8.1 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.

8.2 Metamodel Specification Details: Designators

The following type definitions are used to describe specific metamodel elements like Asset Administration Shells and Submodels with regard to their network / deployment configuration. In doing so, they use certain attributes copied from the model element itself to describe it – hence called *Descriptor*.

8.2.1 Descriptor

Class Name	Descriptor			
Explanation	The self-describing information of a network resource. This class is not part of the metamodel.			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/Descriptor/1/0/RC02			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
endpoint*	Endpoint of the network resource	Endpoint	attr	1..*

8.2.2 AssetAdministrationShellDescriptor

Class Name	AssetAdministrationShellDescriptor
Explanation	Descriptor of an Asset Administration Shell
Inherits from	Descriptor
semanticId	https://admin-shell.io/aas/API/DataTypes/AssetAdministrationShellDescriptor/1/0/RC02

Attribute (* = mandatory)	Explanation	Type	Kind	Card.
administration	Administrative information of the Asset Administration Shell.	AdministrativeInformation	attr	0..1
description	Description or comments on the Asset Administration Shell.	LangStringSet	attr	0..1
globalAssetId	Global reference to the asset the AAS is representing.	Reference	attr	0..1
specificAssetId	Specific asset identifier.	IdentifierKeyValuePair	attr	0..*
idShort	Short name of the Asset Administration Shell.	String	attr	0..1
identification*	Globally unique identification of the Asset Administration Shell.	Identifier	attr	1
submodelDescriptor	Descriptor of a submodel of the Asset Administration Shell.	SubmodelDescriptor	attr	0..*

8.2.3 SubmodelDescriptor

Class Name	SubmodelDescriptor
Explanation	A descriptor of a submodel
Inherits from	Descriptor
semanticId	https://admin-shell.io/aas/API/DataTypes/SubmodelDescriptor/1/0/RC02

Attribute (* = mandatory)	Explanation	Type	Kind	Card.

administration	Administrative information of the Submodel.	AdministrativeInformation	attr	0..1
description	Description or comments on the Submodel.	LangStringSet	attr	0..1
idShort	Short name of the Submodel.	String	attr	0..1
identification*	Globally unique identification of the Submodel.	Identifier	attr	1
semanticId	Identifier of the semantic definition of the Submodel.	Reference	attr	0..1

8.2.4 Endpoint

Class Name	Endpoint			
Explanation	The endpoint description of a network resource. This class is not part of the metamodel.			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/Endpoint/1/0/RC02			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
protocolInformation*	Protocol information of the network resource endpoint	ProtocolInformation	attr	1
interface*	Name of the offered interface at the endpoint	string	attr	1

The following names will be used for the interfaces:

Interface	**interface-shortName**
Asset Administration Shell Interface	AAS
Submodel Interface	SUBMODEL
Asset Administration Shell Serialization Interface	AAS-SERIALIZE
AASX File Server Interface	AASX-FILE
Asset Administration Registry Interface	AAS-REGISTRY
Submodel Registry Interface	SUBMODEL-REGISTRY
Asset Administration Shell Repository Interface	AAS-REPOSITORY
Submodel Repository Interface	SUBMODEL-REPOSITORY
Concept Description Repository Interface	CD-REPOSITORY
Asset Administration Shell Basic Discovery Interface	AAS-DISCOVERY

The value for the interface attribute is “{interface-shortName}-{interface-version}”.

The interface-version of this specification is “1.0”, e.g. the entry for the Asset Administration Shell Interface is “AAS-1.0”.

An example for a descriptor with several endpoints is shown in the following:

```
{
    "endpoints": [{
        "protocolInformation": {
            "endpointAddress": "https://localhost:1234",
            "endpointProtocolVersion: "1.1"
        },
        "interface": "AAS-1.0"
    },
    {
        "protocolInformation": {
            "endpointAddress": "opc.tcp://localhost:4840"
        },
        "interface": "AAS-1.0"
    },
    {
        "protocolInformation": {
            "endpointAddress": "https://localhost:5678",
            "endpointProtocolVersion: "1.1",
            "subprotocol": "OPC UA Basic SOAP",
            "subprotocolBody": "ns=2;s=MyAAS",
            "subprotocolBodyEncoding": "plain"
        },
        "interface": "AAS-1.0"
    }]
}
```

8.2.5 ProtocolInformation

Class Name	ProtocolInformation
Explanation	The protocol information of a network resource endpoint will be defined in DIN SPEC 16593-2. After the release of DIN SPEC 16593-2 the missing explanations will be copied into this document. This class is not part of the metamodel.
Inherits from	--
semanticId	https://admin-shell.io/aas/API/DataTypes/ProtocolInformation/1/0/RC02

Attribute (* = mandatory)	Explanation	Type	Kind	Card.
endpointAddress*	Will be defined in DIN SPEC 16593-2	string	attr	1
endpointProtocol	Will be defined in DIN SPEC 16593-2	string	attr	0..1
endpointProtocolVersion	Will be defined in DIN SPEC 16593-2	string	attr	0..1
subprotocol	Will be defined in DIN SPEC 16593-2	string	attr	0..1
subprotocolBody	Will be defined in DIN SPEC 16593-2	string	attr	0..1
subprotocolBodyEncoding	Will be defined in DIN SPEC 16593-2	string	attr	0..1
securityAttributes	Will be defined in DIN SPEC 16593-2	securityAttribute	attr	1..*

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

8.2.7 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 8.2.7.1.

Table 1 shows generic status codes 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

8.2.7.1 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			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/Result/1/0/RC02			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.

Class Name	Result			
Explanation	The result object			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/Result/1/0/RC02			
success*	Indicated whether the operation execution is seen as successful	Boolean	attr	1
message	Additional message containing information for the requester	Message	attr	0..*

Class Name	Message			
Explanation	A message containing more information for the requester about a certain happening in the backend.			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/Message/1/0/RC02			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
messageType*	The message type	MessageTypeEnum	attr	1
text*	The message text	string	attr	1
code	Technology-dependent status or error code	String	attr	0..1
timestamp	Timestamp of the message	dateTime	attr	0..1

Enumeration	MessageTypeEnum
Explanation	The message type
semanticId	https://admin-shell.io/aas/API/DataTypes/MessageTypeEnum/1/0/RC02
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 handling errors
Exception	Used if it is an internal and/or unhandled exception that occurred

8.2.7.2 Operation Objects

The following type definitions are used to call and handle the requests and responses while performing synchronous or asynchronous operation invocation.

8.2.7.2.1 OPERATIONREQUEST

Class Name	OperationRequest			
Explanation	The operation request object			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/OperationRequest/1/0/RC02			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
requestId*	Client request id	string	attr	1
inputArguments	Input argument	OperationVariable	attr	0..*

inoutputArguments	InOutput argument	OperationVariable	attr	0..*
timestamp	Timestamp until when the client expects the server to have finished execution of the invoked operation	DateTime (UTC)	attr	0..1

8.2.7.2.2 OPERATIONRESULT

Class Name	OperationResult			
Explanation	The operation’s invocation result object			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/OperationResult/1/0/RC02			
Attribute (* = mandatory)	**Explanation**	**Type**	**Kind**	**Card.**
requestId*	Client request id	String	attr	1
outputArguments	Output argument	OperationVariable	attr	0..*
inoutputArguments	InOutput argument	OperationVariable	attr	0..*
executionResult*	Execution result object	Result	attr	1
executionState*	Execution state	ExecutionState	attr	1

8.2.7.2.3 ENUMERATION EXECUTIONSTATE

Enumeration	ExecutionState
Explanation	The operation's invocation result state
semanticId	https://admin-shell.io/aas/API/DataTypes/ExecutionState/1/0/RC02
Literal	**Explanation**
Initiated	The operation is ready to be executed (initial state)
Running	The operation is running
Completed	The operation is completed
Canceled	The operation was cancelled externally
Failed	The operation failed
Timeout	The operation has timed out due to given client timeout

8.2.7.2.4 OPERATIONHANDLE

Class Name	OperationHandle			
Explanation	The returned handle of an operation's asynchronous invocation used to request the current state of the operation's execution.			
Inherits from				
semanticId	https://admin-shell.io/aas/API/DataTypes/OperationHandle/1/0/RC02			
Attribute (* = mandatory)	**Explanation**	**Type**	**Kind**	**Card.**
requestId*	Client request id	string	attr	1
handleId*	Handle id	string	attr	1

9 Basic Operation Parameters

9.1 General

In this clause the parameters for API operations are specified.

9.2 Output Modifiers in Operations

Definition

An OutputModifier indicates the requester's expected or desired format of the response content of a requested operation. The OutputModifier comprises out of three orthogonal enumerations. These enumerations combined influence 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 a 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.

For Content equal to Value see Clause 9.4.2 for details.

Value	Explanation
Normal (Default)	The standard serialization of the model element or child elements is applied.
Value	Only the raw value of the model element or child elements is returned. Commonly referred to as *ValueOnly*-serialization.
Reference	Only applicable to Referables. The reference to found element is returned.
Path	Returns the idShort of the requested element and a list of *idShort* paths to child elements if the requested element is a Submodel, a SubmodelElementStruct, a SubmodelElementList, a AnnotatedRelationshipElement or an Entity.

3. Enumeration: Extent

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

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

9.3 Applicability of the Output Modifiers

The defined OutputModifiers are only valid for specific operations. In general, OutputModifiers are only applicable to GET-operations. Also, the applicability depends on the kind of the requested resource. The following list defines the applicability of the modifiers to the resources.

Ressource Name	Level Modifier	Content Modifier	Extent Modifier
Asset Administration Shell	No	Normal/Reference	No
Submodel Reference	No	No	No
Submodel	Deep/Core	Normal/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
SubmodelElements			
SubmodelElementStruct	Deep/Core	Normal/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
SubmodelElementList	Deep/Core	Normal/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
Entity	Deep/Core	Normal/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
BasicEvent	No	Normal/Reference	No

Ressource Name	Level Modifier	Content Modifier	Extent Modifier
Capability	No	Normal/Reference	No
Operation	No	Normal/Reference	No
DataElements			
Property	No	Normal/Value/Reference	No
MultilanguageProperty	No	Normal/Value/Reference	No
Range	No	Normal/Value/Reference	No
RelationshipElement	No	Normal/Value/Reference	No
AnnotatedRelationshipElement	No	Normal/Value/Reference	No
Blob	No	Normal/Value/Reference	WithoutBLOBValue/ WithBLOBValue
File	No	Normal/Value/Reference	No

9.4 Serialization in Specified Formats (Output Modifier *Content*)

9.4.1 General

If the output modifier *Content* is set to **Value**, the returned payload depends on the selected serialization format.

Up to now only the serialization in JSON is specified. Other serialization formats (e.g. XML, RDF, etc.) are to be defined in future versions of this document.

9.4.2 ValueOnly-Serialization in JSON

This clause explains how to return only the submodel element's value if the output modifier *Content* is set to *Value*.

In many cases, applications using 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 constant most of the time. Instead, applications are most likely interested in the values of the modelled data only. Furthermore, having limited processing power or limited bandwidth, one use case of this output modifier is to transfer data as efficient as possible. In that regard, one might split semantics and data into two separate architecture building blocks. For example, a database would suit the needs for querying semantics and a device would only provide the data at runtime. With two separate requests one can build up a user interface (UI) and show new upcoming values highly efficiently.

Values are only available for

- All subtypes of abstract type *DataElement*,
- SubmodelElementList and SubmodelElementStruct resp. for their included SubmodelElements,
- ReferenceElement,
- RelationshipElement + AnnotatedRelationshipElement,
- Entity

Operations and Capabilities are excluded from the output modifier's scope since only data containing elements are in the centre of focus. Consequently, 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 in accordance with the data type to value mapping (see table after this section).

- *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}` and `${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}`.

- *SubmodelElementStruct* is serialized as named JSON object with `${SubmodelElementStruct/idShort}` as the name of the containing JSON property. The elements contained within the struct are serialized according to their respective type with `${SubmodelElement/idShort}` as the name of the containing JSON property.

- *SubmodelElementList* is serialized as named JSON array with `${SubmodelElementList/idShort}` as the name of the containing JSON property. The elements contained within the list are serialized according to their respective type.

- *ReferenceElement* is serialized as `${ReferenceElement/idShort}`: `${ReferenceElement/value}` where $`{ReferenceElement/value}` is the serialization of the *Reference* class depending on its subtype:
  - *GlobalReference:* The value is serialized as an array of (*Identifier*-)strings
  - *ModelReference:* The value is serialized as an array of keys.

- *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}`.

- *BasicEvent* is serialized as named JSON object with `${BasicEvent/idShort}` as the name of the containing JSON property. The JSON object contains one JSON property named “observed” with the corresponding value of `${BasicEvent/observed}` as the standard serialization of the *Reference* class.

- *IdentifierKeyValuePair* is serialized as named JSON object with three JSON properties named 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.

Data type to value mapping[4]

The serialization of submodel element values is described in the following table. The left column “Data Type” shows the data types which can be used for submodel element values. The data types are defined according to the W3C XML Schema (https://www.w3.org/TR/xmlschema-2/#built-in-datatypes and https://www.w3.org/TR/xmlschema-2/#built-in-derived). “Value Range” further explains the possible range of data values for this data type. In the right column are related examples of the serialization of submodel element values.

	Data Type	Value Range	Sample Values
Core Types	xsd:string	Character string	"Hello world", "Καλημέρα κόσμε", "コンニチハ"
	xsd:boolean	true, false	true, false
	xsd:decimal	Arbitrary-precision decimal numbers	-1.23, 12678967237489273942 4.543233, +100000.00, 210
	xsd:integer	Arbitrary-size integer numbers	-1, 0, 126789675432332938792837429837429837429, +100000
IEEE-floating-point numbers	xsd:double	64-bit floating point numbers incl. ±Inf, ±0, NaN	-1.0, +0.0, -0.0, 234.567e8, -INF, NaN
	xsd:float	32-bit floating point numbers incl. ±Inf, ±0, NaN	-1.0, +0.0, -0.0, 234.567e8, -INF, NaN
Time and data	xsd:date	Dates (yyyy-mm-dd) with or without timezone	"2000-01-01","2000-01-01Z", "2000-01-01+12:05"
	xsd:time	Times (hh:mm:ss.sss…) with or without timezone	"14:23:00", "14:23:00.527634Z", "14:23:00+03:00"
	xsd:dateTime	Date and time with or without timezone	"2000-01-01T14:23:00", "2000-01-01T14:23:00.66372+14:00"
	xsd:dateTimeStamp	Date and time with required timezone	"2000-01-01T14:23:00.66372+14:00"
Recurring and partial dates	xsd:gYear	Gregorian calendar year	"2000", "2000+03:00"
	xsd:gMonth	Gregorian calendar month	"--04", "--04+03:00"
	xsd:gDay	Gregorian calendar day of the month	"---04", "---04+03:00"
	xsd:gYearMonth	Gregorian calendar year and month	"2000-01", "2000-01+03:00"
	xsd:gMonthDay	Gregorian calendar month and day	"--01-01", "--01-01+03:00"
	xsd:duration	Duration of time	"P30D", "-P1Y2M3DT1H", "PT1H5M0S"
	xsd:yearMonthDuration	Duration of time (months and years only)	"P10M", 'P5Y2M"

[4] cf. https://openmanufacturingplatform.github.io/sds-bamm-aspect-meta-model/bamm-specification/v1.0.0/datatypes.html

	xsd:dayTimeDuration	Duration of time (days, hours, minutes, seconds only)	"P30D", 'P1DT5H", 'PT1H5M0S"
Limited-range integer numbers	xsd:byte	-128…+127 (8 bit)	-1, 0, 127
	xsd:short	-32768…+32767 (16 bit)	-1, 0, 32767
	xsd:int	2147483648…+2147483647 (32 bit)	-1, 0, 2147483647
	xsd:long	-9223372036854775808…+9223372036854775807 (64 bit)	-1, 0, 9223372036854775807
	xsd:unsignedByte	0…255 (8 bit)	0, 1, 255
	xsd:unsignedShort	0…65535 (16 bit)	0, 1, 65535
	xsd:unsignedInt	0…4294967295 (32 bit)	0, 1, 4294967295
	xsd:unsignedLong	0…18446744073709551615 (64 bit)	0, 1, 18446744073709551615
	xsd:positiveInteger	Integer numbers >0	1, 734568374657836485736847563874 5
	xsd:nonNegativeInteger	Integer numbers ≥0	0, 1, 734568374657836485736847563874 5
	xsd:negativeInteger	Integer numbers <0	-1, -2348726384762837648273648726384 7
	xsd:nonPositiveInteger	Integer numbers ≤0	-1, 0, -9384583749857398749879898739 4
Encoded binary data	xsd:hexBinary	Hex-encoded binary data	"6b756d6f77617368657265"
	xsd:base64Binary	Base64-encoded binary data	"a3Vtb3dhc2hlcmU="
Miscellaneous types	xsd:anyURI	Absolute or relative URIs and IRIs	"http://customer.com/demo/aas/1/1/1234859590", "urn:example:company:1.0.0"
	rdf:langString	Strings with language tags	"Hello"@en, "Hallo"@de. Note that this is written in RDF/Turtle syntax, and that only "Hello" and "Hallo" are the actual values.

The following types defined by the XSD and RDF specifications are explicitly omitted for serialization:

xsd:language, xsd:normalizedString, xsd:token, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:QName, xsd:ENTITY, xsd:ID, xsd:IDREF, xsd:NOTATION, xsd:IDREFS, xsd:ENTITIES, xsd:NMTOKENS, rdf:HTML and rdf:XMLLiteral.

Note: Due to the limits in the representation of numbers in JSON, the maximum integer number that can be used without losing precision is 2^{53}-1 (defined as Number.MAX_SAFE_INTEGER). This means that even if the used data type would allow higher or lower values, if they cannot be represented in JSON, they cannot be used. Affected data types are unbounded numeric types `xsd:decimal`, `xsd:integer`, `xsd:positiveInteger`, `xsd:nonNegativeInteger`, `xsd:negativeInteger`, `xsd:nonPositiveInteger` and the bounded type `xsd:unsignedLong`. Other numeric types are not affected.[5]

Examples conformant to [3]:

Full serialization of single submodel element *Property*:

```
{
    "idShort": "MaxRotationSpeed",
    "category": "PARAMETER",
    "kind": "Instance",
    "semanticId": {
        "keys": [{
            "type": "ConceptDescription",
            "value": "0173-1#02-BAA120#008",
        }]
    },
    "modelType": {
        "name": "Property"
    },
    "valueType": "int",
    "value": 5000
}
```

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

```
{
    "MaxRotationSpeed" : 5000
}
```

[5] cf. https://openmanufacturingplatform.github.io/sds-bamm-aspect-meta-model/bamm-specification/v1.0.0/payloads.html#data-type-mappings

For a *SubmodelElementStruct* the struct is serialized as objects denoted by curly brackets:

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

For a *SubmodelElementList* the struct is serialized as array denoted by square brackets:

```
{
    "NamesOfFamilyMembers": [
        "Martha ExampleFamily",
        "Jonathan ExampleFamily",
        "Clark 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"
    }
}
```

Note: In accordance with IETF RFC 5646, the language names match the following regular expression:

```
^[a-z]{2,4}(-[A-Z][a-z]{3})?(-([A-Z]{2}|[0-9]{3}))?$
```

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

```
{
    "TorqueRange": {
        "min": 3,
        "max": 15
    }
}
```

For a *ReferenceElement* named “MaxRotationSpeedReference” the payload is minimized to the following in case the *Reference* is of subtype *ModelReference*:

```
{
    "MaxRotationSpeedReference": [
        {
            "type": "Submodel",
            "value": "http://customer.com/demo/aas/1/1/1234859590"
        },
        {
            "type": "Property",
            "value": "MaxRotationSpeed"
        }
    ]
}
```

For the same *ReferenceElement* the payload is minimized to the following in case the *Reference* is of subtype *GlobalReference*:

```
{
    "MaxRotationSpeedReference": [ "0173-1#02-BAA120#008" ]
}
```

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***

```
{
    "Library": {
        "mimeType": "application/octet-stream"
    }
}
```

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


```
{
    "Library": {
        "mimeType": "application/octet-stream",
        "value": "VGhpcyBpcyBteSBibG9i"
    }
}
```


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


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

For a *AnnotatedRelationshipElement* named “CurrentFlowFrom” with an annotated *Property*-DataElement “AppliedRule” the payload is minimized to the following:

```
{
    "CurrentFlowsFrom": {
        "first": [
            {
                "type": "Submodel",
                "value": "http://customer.com/demo/aas/1/1/1234859590"
            },
            {
                "type": "Property",
                "value": "PlusPole"
            }
        ],
        "second": [
            {
                "type": "Submodel",
                "value": "http://customer.com/demo/aas/1/0/1234859123490"
            },
            {
                "type": "Property",
                "value": "MinusPole"
            }
        ],
        "annotation": [
            {
                "AppliedRule": "TechnicalCurrentFlowDirection"
            }
        ]
    }
}
```

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

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

For a BasicEvent named “MyBasicEvent” the payload is minimized to the following:

```
{
    "MyBasicEvent": {
        "observed": [
            {
                "type": "Submodel",
                "value": "http://customer.com/demo/aas/1/1/1234859590"
            },
            {
                "type": "Property",
                "value": "CurrentValue"
            }
        ]
    }
}
```

9.4.3 JSON-Schema for the ValueOnly-Serialization

The following JSON-Schema represents the validation schema for the ValueOnly-serialization of submodel elements. This holds true for all submodel elements mentioned in the previous chapter except for *SubmodelElementStruct*s and *SubmodelElementList**s***. Since *SubmodelElementStruct*s and *SubmodelElementList*s are treated as objects containing submodel elements of any kind, the integration into the same validation schema would result in a circular reference or ambiguous results ignoring the actual validation of other submodel elements that are no *SubmodelElementStruct*s or *SubmodelElementList*s. Hence, for each *SubmodelElementStruct* and *SubmodelElementList* within a submodel element hierarchy the same validation schema must be applied. In this case, it may be necessary to create a specific JSON-schema for the individual use-case resp. submodel referencing this validation schema from the respective *SubmodelElementStruct* or *SubmodelElementList*.
See Annex C for an example that validates against this schema.

```
{
  "$schema": "https://json-schema.org/draft/2019-09/schema",
  "title": "ValueOnlySerializationSchema",
  "$id": "http://www.admin-shell.io/schema/valueonly/json/V3.0RC02",
  "definitions": {
    "PropertyValue": {
      "oneOf": [
        {
          "$ref": "#/definitions/StringValue"
        },
        {
          "$ref": "#/definitions/NumberValue"
```

```
      },
      {
        "$ref": "#/definitions/BooleanValue"
      }
    ]
  },
  "MultiLanguagePropertyValue": {
    "type": "object",
    "patternProperties": {
      "^[a-z]{2,4}(-[A-Z][a-z]{3})?(-([A-Z]{2}|[0-9]{3}))?$": {
        "type": "string"
      }
    },
    "additionalProperties": false
  },
  "RangeValue": {
    "type": "object",
    "properties": {
      "min": {
        "type": "number"
      },
      "max": {
        "type": "number"
      }
    },
    "required": [
      "min",
      "max"
    ],
    "additionalProperties": false
  },
  "FileBlobValue": {
    "type": "object",
    "properties": {
```

```
        "mimeType": {
          "type": "string"
        },
        "value": {
          "type": "string"
        }
      },
      "required": [
        "mimeType",
        "value"
      ],
      "additionalProperties": false
    },
      "ReferenceElementValue": {
        "$ref": "#/definitions/ReferenceValue"
      },
    "ReferenceValue": {
      "oneOf":[
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Key"
          },
        },
        {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Identifier"
          }
        }
      ]
    },
      "Identifier": {
        "type": "string"
```

```
    },
  "BasicEventValue": {
    "type": "object",
    "properties": {
      "observed": {
          "$ref": "#/definitions/ReferenceValue"
      }
    },
    "required": ["observed"],
    "additionalProperties": false
  },
  "EntityValue": {
    "type": "object",
    "properties": {
      "statements": {
        "$ref": "#/definitions/ValueOnly"
      },
      "entityType": {
        "enum": [
          "SelfManagedEntity",
          "CoManagedEntity"
        ]
      },
      "globalAssetId": {
          "$ref": "#/definitions/ReferenceValue"
      },
      "specificAssetIds": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/IdentifierKeyValuePairValue"
          }
      }
    },
    "required": [
```

```
        "statements",
        "entityType"
      ],
      "additionalProperties": false
    },
      "IdentifierKeyValuePairValue": {
        "type": "object",
      "patternProperties": {
        "(.*?)": {
          "type": "string"
        }
      }
      },
    "RelationshipElementValue": {
      "type": "object",
      "properties": {
        "first": {
            "$ref": "#/definitions/ReferenceValue"
        },
        "second": {
            "$ref": "#/definitions/ReferenceValue"
        },
      },
      "required": [
        "first",
        "second"
      ],
      "additionalProperties": false
    },
    "AnnotatedRelationshipElementValue": {
      "type": "object",
      "properties": {
        "first": {
            "$ref": "#/definitions/ReferenceValue"
```

```
      },
      "second": {
          "$ref": "#/definitions/ReferenceValue"
      },
      "annotation": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/ValueOnly"
        }
      }
    },
    "required": [
      "first",
      "second",
      "annotation"
    ],
    "additionalProperties": false
  },
  "Key": {
    "type": "object",
    "properties": {
      "type": {
        "type": "string"
      },
      "value": {
        "type": "string"
      }
    },
    "required": [
      "type",
      "value"
    ],
    "additionalProperties": false
  },
```

```
    "StringValue": {
      "type": "string",
      "additionalProperties": false
    },
    "NumberValue": {
      "type": "number",
      "additionalProperties": false
    },
    "BooleanValue": {
      "type": "boolean",
      "additionalProperties": false
    },
    "SubmodelElementValue": {
      "oneOf": [
        {
          "$ref": "#/definitions/BasicEventValue"
        },
        {
          "$ref": "#/definitions/RangeValue"
        },
        {
          "$ref": "#/definitions/MultiLanguagePropertyValue"
        },
        {
          "$ref": "#/definitions/FileBlobValue"
        },
        {
          "$ref": "#/definitions/ReferenceElementValue"
        },
        {
          "$ref": "#/definitions/RelationshipElementValue"
        },
        {
          "$ref": "#/definitions/AnnotatedRelationshipElementValue"
```

```
        },
        {
          "$ref": "#/definitions/EntityValue"
        },
        {
          "$ref": "#/definitions/PropertyValue"
        }
      ]
    },
    "ValueOnly": {
      "propertyNames": {
        "pattern": "^[A-Za-z_][A-Za-z0-9_-]*$"
      },
      "patternProperties": {
        "^[A-Za-z_][A-Za-z0-9_-]*$": {
          "$ref": "#/definitions/SubmodelElementValue"
        }
      },
      "additionalProperties": false
    }
  }
}
```

9.4.4 IdShortPath serialization

To get only the idShort paths of a submodel element hierarchy, the serialization format is specified in terms of an idShortPath notation to be returned in an unnamed JSON-array. The notation differs whether a SubmodelElementStruct or a SubmodelElementList is used. In the first case, the submodel element's idShort is separated via "." (dot) going from top level down to child level. In the second case, after the idShort of the containing SubmodelElementList square brackets with an index are appended "[<<index>>]".

Given the following example, a request for idShort paths starting at *MySubmodelElementStruct* with OutputModifier level = deep, the list of idShort paths are returned as follows:

Submodel: MySubmodel

- ⇨ Property: MyTopLevelProperty
- ⇨ SMS: MySubmodelElementStruct
  - Property: MySubProperty1
  - Property: MySubProperty2
  - SMS: MySubSubmodelElementStruct
    - Property: MySubSubProperty1
    - Property: MySubSubProperty2
  - SML: MySubSubmodelElementList
    - Property<string>: "MySubTestValue1",
    - Property<string>: "MySubTestValue2",

```
[
    "MySubmodelElementStruct",
    "MySubmodelElementStruct.MySubProperty1",
    "MySubmodelElementStruct.MySubProperty2",
    "MySubmodelElementStruct.MySubSubmodelElementStruct",
    "MySubmodelElementStruct.MySubSubmodelElementStruct.MySubSubProperty1",
    "MySubmodelElementStruct.MySubSubmodelElementStruct.MySubSubProperty2",
    "MySubmodelElementStruct.MySubSubmodelElementList[0]",
    "MySubmodelElementStruct.MySubSubmodelElementList[1]"
]
```

10 HTTP/REST API

10.1 General

In this clause the technology mapping to HTTP/REST APIs is described.

The OpenAPI specification of the HTTP/REST APIs can be found at SwaggerHub.

Schema for all APIs:
https://app.swaggerhub.com/domains/Plattform_i40/Shared-Domain-Models/Final-Draft

All single APIs:
https://app.swaggerhub.com/apis/Plattform_i40/AssetAdministrationShell-REST-API/Final-Draft

AAS API with SM and Serialization APIs included:
https://app.swaggerhub.com/apis/Plattform_i40/AssetAdministrationShell-Standalone/Final-Draft

AAS Repository with AAS, SM, SM Repository, CD Repository and Serialization APIs included:
https://app.swaggerhub.com/apis/Plattform_i40/AssetAdministrationShell-Repository/Final-Draft

AAS Registry with AAS Discovery API included:
https://app.swaggerhub.com/apis/Plattform_i40/Registry-and-Discovery/Final-Draft

This clause gives an overview of the HTTP/REST API and describes general design decisions.

10.2 Design Decisions

The following design decisions and constraints hold for the HTTP/REST API:

- It has been decided to use OpenAPI and Swaggerhub for specification. This leads to the constraint that one operation can only provide one type of a resulting payload.
- This document assumes version 1.1 of HTTP.
- Generic output parameters changing the type of payload have been mapped to corresponding query parameters, e.g.”?level=” or “?content=”.
- Query parameters are also used when the type of a resulting payload is a list of objects and the type remains the same, but the query parameter filters the content of the list, e.g. GetAllSubmodels with optional query parameters “?semanticId=” or “?idShort=”.
- By standard complete objects are provided as requested payload, e.g. a complete submodel. This corresponds to the generic output parameter content=”normal”. Reduced objects can be requested by query parameter “?content=trimmed”. In these trimmed objects selected elements are left off in the payload. Please see clause 10.4.
- FILE content is not part of the meta model defined in Part 1 [2], V3.0RC02. For this reason, the resolution of the FILE content must be enabled separately by the server. Typically, the value of a FILE element may contain an URL, where to retrieve the file data by a GET operation.
- By default, blobs are not part of the payload. Using ?extent=WithBLOBValue includes blobs for submodel elements of kind BLOB.

- Submodels define a hierarchical structure. Certain operations use an idShort-path to access deeper parts in the hierarchy. To easily support this in the REST API, “.” or “[index]” is used as a delimiter in the idShort-paths. Please see clause 10.3. Since, an idShort-path could include square brackets like “[index]”, the idShort-path must be URL-encoded.
- Identifiers of Identifiables are BASE64-URL-encoded to be passed to the HTTP/REST API (see https://www.base64url.com/). These may be identifiers for Asset Administration Shells, Submodels or Concept Descriptions
  Identifiers may also be passed as BASE64-URL-encoded query parameters, e.g. also for semanticId or assetId. Such query parameters are typically used when a list of objects may be retrieved in the resulting payload. A list of BASE64-URL-encoded ids is simply passed as comma separated query parameters.
- When retrieving AssetAdministrationShells (/shells, /lookup/shells) a query parameter “?assetids=” can be specified. Such assetId may be a globalAssetId or specificAssetId. The corresponding key-value-pair is first serialized to JSON and then BASE64-URL-encoded. The resulting encoded string is the value of “?assetids=”.
- In some operations references are part of the query parameters e.g. “?semanticId=”. The corresponding reference is first serialized to JSON and then BASE64-URL-encoded. The resulting encoded string is the value of “?semanticId=”.
- This encoding (serialize to JSON + BASE64-URL) is also used for IdentifierKeyValuePairs, i.e. for GetAllAssetAdministrationShellIdsByAssetLink (i.e. /lookup/shells). For the example
  “[{"key": "globalAssetId","value": "http://example.company/myAsset"},{"key": "myOwnInternalAssetId","value": "12345ABC"}]”
  the resulting BASE64-URL encoded value of the query parameter is
  “?assetIds=W3sia2V5IjogImdsb2JhbEFzc2V0SWQiLCJ2YWx1ZSI6ICJodHRwOi8vZXhhbXBsZS5jb21wYW55L215QXNzZXQifSx7ImtleSI6ICJteU93bkludGVybmFsQXNzZXRJZCIsInZhbHVlIjogIjEyMzQ1QUJDIn1d”.
  If several key-value-pairs are included, all must be part of the key-value-pairs on the server.
- Comparisons of idShort are made case-sensitive in the HTTP/REST API to avoid repeating toupper()/tolower() conversions. Note: This is conformant to the change made in Part 1 [2], V3.0RC02.
- GetAll.. will retrieve a list of objects as the resulting payload, e.g. GetAllSubmodelElements.
- In general, only GET, POST, PUT and DELETE are used. POST is used to create new objects and to invoke operations.
- Some interfaces may be combined in a so called “superpath”, e.g. the Shell Repository Interface may be combined with the AAS Interface and the Submodel Interface. This results in a complete path like: “/shells/{aas-identifier}/aas/submodels/{submodel-identifier}/submodel/*”. This is especially useful when all data is hosted in the same

repository. The support of such superpath is currently recommended but not mandatory. In a future version the /descriptor interface will provide profiles, which will express if superpath is supported by a server or not. Without superpath a client has to follow the mandatory standard interaction pattern to always retrieve endpoints of e.g. submodels from a registry.

- Each interface includes a “/descriptor” operation for self discovery to provide detailed information about the interface. A server supporting the HTTP/REST API may also provide a server global “/descriptor” to provide the information about all available interfaces on that server.
- Some parts of the metamodel of Part 1 [3] are currently not included in the HTTP/REST API V1. These are security metamodel elements and views (views are deprecated in Part 1 [2], V3.0RC02).

10.3 Addressing Resources

The API allows to address each referable element, either by its global identifier or by its idShort-path depending on the object type.

If the referable element is an identifiable, addressing is only possible by the global identifier of the object.
All other referable elements are addressable by the idShort-path.
The idShort-path is a chain of idShorts or SubmodelElementList-indexes which points to an element within a hierarchy of elements. The root of the idShort-path is always a submodel and the first element in an idShort-path is always an idShort of a first level SubmodelElement within a Submodel. Technically the idShort path is a string and the idShorts are separated by a dot while the SubmodelElementList-indexes are written in brackets.

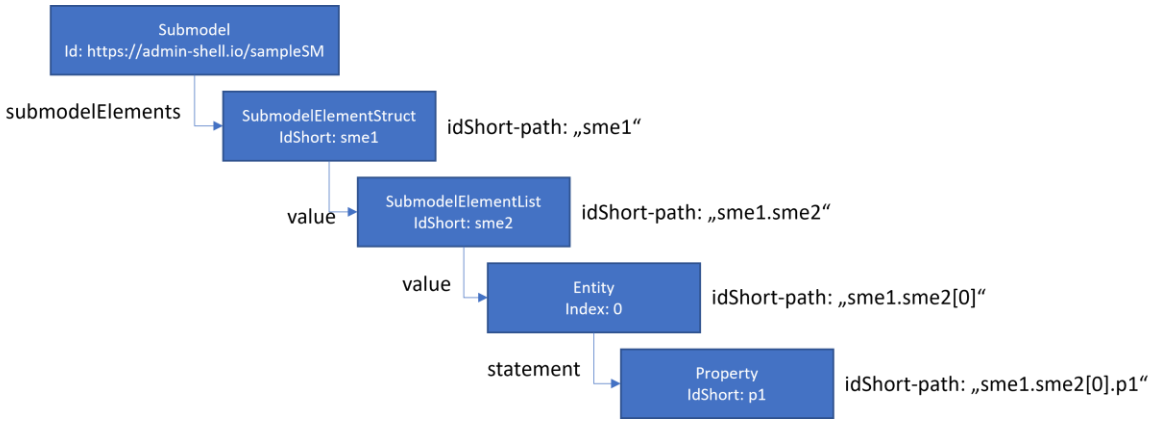


Figure 4 example hierarchy

The example hierarchy shows a Submodel with a hierarchical structure of SubmodelElements. The Submodel can be addressed by its global identifier “https://admin-shell.io/sampleSM”. The

other elements in the picture do not have a global identifier but are uniquely identifiable and addressable by the submodel identifier and the idShort-path. The idShort-path in this example pointing to the Property p1 is “sme1.sme2[0].p1”. The hierarchy is built on parent-child relations between the elements. There are four elements which are able to aggregate submodelElements and by this can create deeper hierarchal structures. The elements are Submodel, SubmodelElementStruct, SubmodelList and Entity. The fields which are used to navigate to a deeper level of the hierarchy can be seen in the following table.

Element Name	Child aggregation field name
Submodel	SubmodelElement
SubmodelElementStruct	value
SubmodelElementList	value
AnnotatedRelationshipElement	annotations
Entity	statements

Example requests:

GET /submodels/aHR0cHM6Ly9hZG1pbi1zaGVsbC5pby9zYW1wbGVTTQ/submodel/submodelElements/ sme1.sme2%5B0%5D.p1

Add a new Property to the Entity statements:

POST /submodels/aHR0cHM6Ly9hZG1pbi1zaGVsbC5pby9zYW1wbGVTTQ/submodel/submodelElements/ sme1.sme2%5B0%5D

To avoid problems with IRIs in URLs the identifiers shall be BASE64-URL-encoded before using them as parameters in the HTTP-APIs. IdshortPaths are URL-encoded to handle including square brackets.

In the example above “aHR0cHM6Ly9hZG1pbi1zaGVsbC5pby9zYW1wbGVTTQ” is the BASE64-URL-encoding of “https://admin-shell.io/sampleSM”, “sme1.sme2%5B0%5D.p1” is the URL-encoding of “sme1.sme2[0].p1” and “sme1.sme2%5B0%5D” is the URL-encoding of “sme1.sme2[0]”.

10.4 Trimmed Objects

Trimmed objects are defined for scenarios where a client only wants to access the metadata of an object but not the value. **Trimmed objects are only part of HTTP/REST and do not change the metamodel.** Trimmed objects are used to reduce the payload response to a minimum and to avoid the recursive traversing through the data model when not needed. In many cases a client is not interested in each child element or value of a resource but only in the resource itself.

A trimmed object does not contain any additional fields in relation to its full object representation, only some fields are left off, i.e. not available (“trimmed”). The left off fields are fields which could be requested by an own API call and may consist of a recursive or potentially large substructure. The serialization of a trimmed object is the same as for the original full object, but without the left off fields.

Class Name	Fields not available in trimmed representation
Identifiables	
AssetAdministrationShell	security, views, assetInformation, submodels
Submodel	submodelElements
SubmodelElements	
SubmodelElementStruct	value
SubmodelElementList	value
Entity	statements, globalAssetId, specificAssetId
BasicEvent	observed
Capability	--
Operation	--
DataElements	
Property	value, valueId
MultilanguageProperty	value, valueId
Range	min, max

RelationshipElement	first, second
AnnotatedRelationshipElement	first, second, annotations
Blob	value, mimeType
File	value, mimeType

Example

The example shows an JSON serialization of an AssetAdministrationShell object in its full representation and how it looks like in a trimmed representation.

For editorial reasons some fields which are the same for both representations are omitted.

Table 1 AssetAdministrationShell JSON serialization example

```
{
    "idShort": "TestAssetAdministrationShell",
    "description": [...],
    "Identification": {...},

     ...

    "derivedFrom": {...}
    "assetInformation": {...},
    "submodels": [...],
    "security": {...},
    "views": [...]
}
```

Table 2 AssetAdministrationShell trimmed JSON serialization example

```
{
    "idShort": "TestAssetAdministrationShell",
    "description": […],
    "Identification": {…}

    ...

    "derivedFrom": {…}
    }
}
```

10.5 Payload

The payload is generated from the technology neutral specification as described in Part 1 of the Asset Administration Shell Series for JSON [2].

The serialization of JSON values is described in clause 9.4.2.

Additional classes needed for payload of the HTTP/REST API specification are found in clause 10.8.

10.6 Modifiers

To use trimmed objects as described in section 10.4, the content modifier has been extended by the “Trimmed” option:

Value	Explanation
Normal (Default)	The standard serialization of the model element or child elements is applied.
Trimmed	Only metadata of an element or child elements but not the value is returned (see section 10.4)
Value	Only the raw value of the model element or child elements is returned. Commonly referred to as *ValueOnly*-serialization.
Reference	Only applicable to Referables. The reference to found element is returned.
Path	Returns the idShort of the requested element and a list of *idShort* paths to child elements if the requested element is a Submodel, a SubmodelElementStruct, a SubmodelElementList, a AnnotatedRelationshipElement or a Entity.

In combination with the level modifier the following rules apply:

- If Level=Core and Content=Trimmed, then only the requested object without the children will be returned in trimmed serialization.
- If Level=Deep and Content=Trimmed, then the requested object and all children on all sublevels are returned in trimmed serialization.

Consequently, the list defining the assignment of the modifiers to the resources contains:

Ressource Name	Level Modifier	Content Modifier	Extent Modifier
Asset Administration Shell	No	Normal/Trimmed/Reference	No
Submodel Reference	No	No	No
Submodel	Deep/Core	Normal/Trimmed/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
SubmodelElements			
SubmodelElementStruct	Deep/Core	Normal/Trimmed/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
SubmodelElementList	Deep/Core	Normal/Trimmed/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
Entity	Deep/Core	Normal/Trimmed/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
BasicEvent	No	Normal/Trimmed/Value/Reference	No
Capability	No	Normal/Reference	No
Operation	No	Normal/Reference	No

Ressource Name	Level Modifier	Content Modifier	Extent Modifier
DataElements			
Property	No	Normal/Trimmed/Value/Reference	No
MultilanguageProperty	No	Normal/Trimmed/Value/Reference	No
Range	No	Normal/Trimmed/Value/Reference	No
RelationshipElement	No	Normal/Trimmed/Value/Reference	No
AnnotatedRelationshipElement	No	Normal/Trimmed/Value/Reference	No
Blob	No	Normal/Trimmed/Value/Reference	WithoutBLOBValue/ WithBLOBValue
File	No	Normal/Trimmed/Value/Reference	No

In addition, the modifiers can also be used for PUT operations. They define how the request content is delivered and have the same semantics as for the related GET operation. Only Content=Reference and Content=Path are not possible for PUT.

In general, the combination of Level=Deep and Content=Reference is not allowed.

10.7 Mapping of Operations

The following table shows the mapping of the generic operations to the HTTP/REST API.

The black entries correspond to the corresponding generic operations.

The blue entries are operations which only exist in the HTTP/REST API.

Operation Name	HTTP Verb	REST-Path	Comment (e.g. optional query parameters)
Asset Administration Shell Interface			
GetAssetAdministrationShell	GET	/aas	?content=normal/trimmed/reference
PutAssetAdministrationShell	PUT	/aas	?content=normal/trimmed
GetAllSubmodelReferences	GET	/aas/submodels	
PostSubmodelReference	POST	/aas/submodels	use BASE64-URL-encoded identifier
DeleteSubmodelReference	DELETE	/aas/submodels/{submodelIdentifier}	use BASE64-URL-encoded identifier
GetAssetInformation	GET	/aas/asset-information	
PutAssetInformation	PUT	/aas/asset-information	
	*	/aas/submodels/{submodel-identifier}/submodel/*	recommended: Submodel Interface for SuperPath
Submodel Interface			
GetSubmodel	GET	/submodel	?level=deep/core ?content=normal/trimmed/value/reference/path ?extent=WithoutBLOBValue/WithBLOBValue
PutSubmodel	PUT	/submodel	?level=deep/core ?content=normal/trimmed/value ?extent=WithoutBLOBValue/WithBLOBValue
GetAllSubmodelElements	GET	/submodel/submodel-elements	?level=deep/core ?content= normal/trimmed/value/reference/path ?extent=WithoutBLOBValue/WithBLOBValue
GetSubmodelElementByPath	GET	/submodel/ submodel-elements/{ idShortPath}	use seperated idshort path of this element ?level=deep/core ?content= normal/trimmed/value/reference/path ?extent=WithoutBLOBValue/WithBLOBValue

			URL-encoded IdShortPath
PostSubmodelElement	POST	/submodel/submodel-elements	?level=deep/core ?content=normal/trimmed/value ?extent=WithoutBLOBValue/WithBLOBValue
PostSubmodelElementByPath	POST	/submodel/submodel-elements/{idShortPath}	use seperated idshort path of the parent element ?level=deep/core ?content=normal/trimmed/value ?extent=WithoutBLOBValue/WithBLOBValue URL-encoded IdShortPath
PutSubmodelElementByPath	PUT	/submodel/ submodel-elements/{idShortPath}	use seperated idshort path of this element ?level=deep/core ?content=normal/trimmed/value ?extent=WithoutBLOBValue/WithBLOBValue URL-encoded IdShortPath
SetSubmodelElementValueByPath	PUT	/submodel/ submodel-elements/{ idShortPath}	use seperated idshort path of this element; see clause 10.3.1 for values ?content=value ?extent=WithoutBLOBValue/WithBLOBValue URL-encoded IdShortPath
DeleteSubmodelElementByPath	DELETE	/submodel/ submodel-elements/{ idShortPath}	use seperated idshort path of this element URL-encoded IdShortPath
InvokeOperationSync	POST	/submodel/ submodel-elements/{ idShortPath}/invoke	?content=normal/value URL-encoded IdShortPath
InvokeOperationAsync	POST	/submodel/ submodel-elements/{ idShortPath}/invoke	get operationHandle ?async=true ?content= normal/value URL-encoded IdShortPath
GetOperationAsyncResult	GET	/submodel/ submodel-elements/{ idShortPath}/operation-results/{handleId}	handleId=operationHandle ?content= normal/value URL-encoded IdShortPath
Shell Repository Interface			
GetAllAssetAdministrationShells	GET	/shells	
GetAllAssetAdministrationShellsByAssetId	GET	/shells	BASE64-URL-encoded JSON-serialized key-value-pairs ?assetids=…

GetAllAssetAdministrationShellsByIdShort	GET	/shells	
GetAssetAdministrationShellById	GET	/shells/{aasIdentifier}	BASE64-URL-encoded identifier
PostAssetAdministrationShell	POST	/shells	
PutAssetAdministrationShellById	PUT	/shells/{aasIdentifier}	BASE64-URL-encoded identifier
DeleteAssetAdministrationShellById	DELETE	/shells/{aasIdentifier}	BASE64-URL-encoded identifier
AasInterface	*	/shells/{aasIdentifier}/aas/*	recommended AAS Interface for SuperPath
Submodel Repository Interface			
GetAllSubmodels	GET	/submodels	
GetAllSubmodelsBySemanticId	GET	/submodels	BASE64-URL-encoded identifier
GetAllSubmodelsByIdShort	GET	/submodels	
GetSubmodelById	GET	/submodels/{submodelIdentifier}	BASE64-URL-encoded identifier
PostSubmodel	POST	/submodels	
PutSubmodelById	PUT	/submodels/{submodelIdentifier}	BASE64-URL-encoded identifier
DeleteSubmodelById	DELETE	/submodels/{submodel>Identifier}	BASE64-URL-encoded identifier
SubmodelInterface	*	/submodels/{submodelIdentifier}/submodel/*	recommended Submodel Interface for SuperPath
Concept Description Repository Interface			
GetAllConceptDescriptions	GET	/concept-descriptions	
GetConceptDescriptionById	GET	/concept-descriptions/{cdIdentifier}	BASE64-URL-encoded identifier
GetAllConceptDescriptionsByIdShort	GET	/concept-descriptions	
GetAllConceptDescriptionsByIsCaseOf	GET	/concept-descriptions	BASE64-URL-encoded identifier
GetAllConceptDescriptionsByDataSpecificationReference	GET	/concept-descriptions	BASE64-URL-encoded identifier
PostConceptDescription	POST	/concept-descriptions/	
PutConceptDescriptionById	PUT	/concept-descriptions/{cdIdentifier}	BASE64-URL-encoded identifier

DeleteConceptDescriptionById	DELETE	/concept-descriptions/{cdIdentifier}	BASE64-URL-encoded identifier
AASX File Server Interface			
GetAllAASXPackageIds	GET	/packages	BASE64-URL-encoded identifier
PostAASXPackage	POST	/packages	
GetAASXByPackageId	GET	/packages/{packageId}	BASE64-URL-encoded identifier
PutAASXByPackageId	PUT	/packages/{packageId}	BASE64-URL-encoded identifier
DeleteAASXByPackageId	DELETE	/packages/{packageId}	BASE64-URL-encoded identifier
AAS Serialization Interface			
GenerateSerializationByIds	GET	/serialization	BASE64-URL-encoded identifier; AcceptHeader: application/aasx+xml oder application/json oder application/xml
AAS Basic Discovery Interface			
GetAllAssetAdministrationShellIdsByAssetLink	GET	/lookup/shells	BASE64-URL-encoded JSON-serialized key-value-pairs ?assetids=…
GetAllAssetLinksById	GET	/lookup/shells/{aasIdentifier}	BASE64-URL-encoded identifier
PostAllAssetLinksById	POST	/lookup/shells/{aasIdentifier}	BASE64-URL-encoded identifier
DeleteAllAssetLinksById	DELETE	/lookup/shells/{aasIdentifier}	BASE64-URL-encoded identifier
AAS Registry Interface			
GetAllAssetAdministrationShellDescriptors	GET	/registry/shell-descriptors	
GetAssetAdministrationShellDescriptorById	GET	/registry/shell-descriptors/{aasIdentifier}	BASE64-URL-encoded identifier
PostAssetAdministrationShellDescriptorById	POST	/registry/shell-descriptors/{aasIdentifier}	BASE64-URL-encoded identifier

PutAssetAdministrationShellDescriptorById	PUT	/registry/shell-descriptors/{aasIdentifier}	BASE64-URL-encoded identifier
DeleteAssetAdministrationShellDescriptorById	DELETE	/registry/shell-descriptors/{aasIdentifier}	BASE64-URL-encoded identifier
Submodel Registry Interface	*	/registry/shell-descriptors/{aasIdentifier}/submodelDescriptors/*	recommended: Submodel Registry Interface for SuperPath
Submodel Registry Interface			
GetAllSubmodelDescriptors	GET	/registry/submodel-descriptors	
GetSubmodelDescriptorById	GET	/registry/submodel-descriptors/{submodelIdentifier}	BASE64-URL-encoded identifier
PostSubmodelDescriptor	POST	/registry/submodel-descriptors/{submodelIdentifier}	BASE64-URL-encoded identifier
PutSubmodelDescriptorById	PUT	/registry/submodel-descriptors/{submodelIdentifier}	BASE64-URL-encoded identifier
DeleteSubmodelDescriptorById	DELETE	/registry/submodel-descriptors/{submodelIdentifier}	BASE64-URL-encoded identifier
Descriptor Interface			
GetDescriptor	GET	/descriptor	Provide additional information on interface endpoint; may also be used at a server endpoint to list all interfaces available on that server

10.8 Mapping of Status Codes

The following table shows the mapping of the generic status codes to HTTP status codes according to IETF RFC 7231.

Generic Status Code	Meaning	HTTP status code	Explanation
Success	Success	200	Standard response for successful requests
SuccessCreated	Creation of a new resource successful	201	Successful request resulting in the creation of a new resource, e.g. SubmodelElement
SuccessNoContent	Success with explicitly no content in the payload	204	Successful request with no content in return, e.g. used for updating existing resources
ClientForbidden	Request is unauthorized	403	The request content is basically valid and understood by the server, but the server refuses the action due to certain restrictions, e.g. profiles.
ClientErrorBadRequest	Bad or malformed request	400	The server does not / cannot process the request due to a general client error, e.g. malformed request
ClientMethodNotAllowed	Operation request is not allowed	405	The method is not supported on a requested operation, e.g. using POST to create a new SubmodelElement
ClientErrorResourceNotFound	Resource not found	404	The requested resource was not found
ServerInternalError	Unexpected error	500	General server internal error due to an unexpected condition
ServerErrorBadGateway	Bad Gateway	502	The primarily addressed server that was acting as gateway or proxy received an invalid response from subsequent systems/servers.

10.9 Additional Data Types for Payload specific for HTTP/REST

In addition to the data types used in the technology neutral specification the HTTP/REST API uses the data types as defined in this clause.

10.9.1 AssetAdministrationShellEnvironment

Class Name	AssetAdministrationShellEnvironment			
Explanation	The top-level aggregation of all identifiables. Identifiables are AssetAdministrationShell, Submodel and ConceptDescription. This class is not part of the metamodel.			
Inherits from	--			
Attribute (* = mandatory)	Explanation	Type	Kind	Card.
assetAdministrationShell	Asset Administration Shell in environment	AssetAdministrationShell	aggr	0..*
submodel	Submodel in environment	Submodel	aggr	0..*
conceptDescription	Concept Description in environment	ConceptDescription	aggr	0..*

10.9.2 PackageDescription

Class Name	PackageDescription			
Explanation	The package description consists of a system wide unique packageId and their corresponding Asset Administration Shell identifiers. The packageId is used to identify the AASX package at the AASX file server. The package description is used to list the Asset Administration Shells in a given AASX package. This class is not part of the metamodel.			
Inherits from				
Attribute (* = mandatory)	Explanation	Type	Kind	Card.

Class Name	PackageDescription			
packageId*	File server specific package id	string	attr	1
aasId	Asset Administration Shell unique identifier	Identifier	attr	0..*

10.1 Interactions

Interactions describe the sequence of calls of operations by a client application to achieve a defined goal in a use cases. Future versions of the document will describe interactions for further usecases.

Currently only the key usecase “Access a submodel in a distributed system” with focus on a completely decentralized Indsutrie 4.0 system is described.

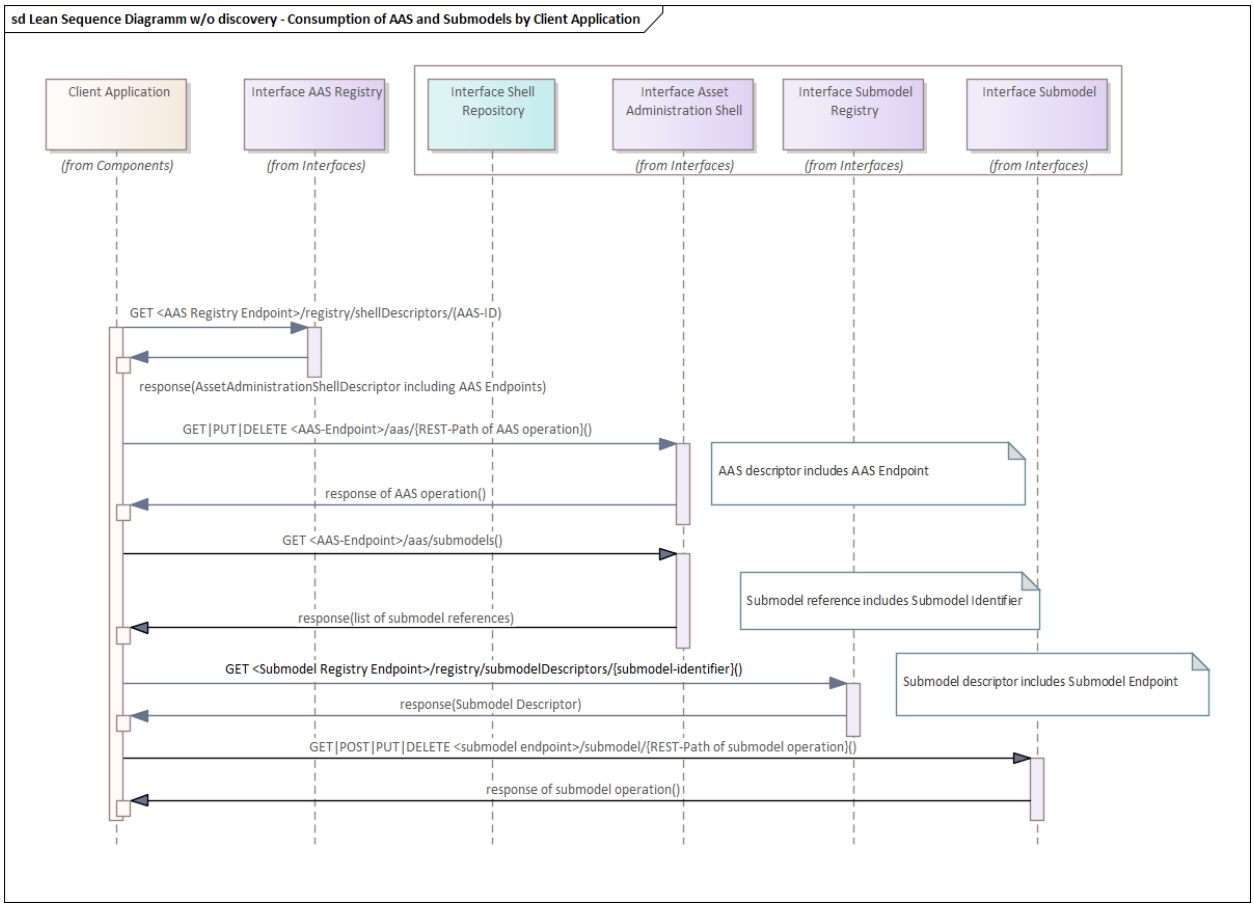
As the interaction diagram in the current version just describes a first subset of interactions, some constraints and assumptions are made according the configuration and qualities of the system. In future versions of the document further interactions will be described (mentioned below as “to be created”), improving the degree if automation of the configuration and quality (flexibility, security,…) of the system itself.

Constraints and assumptions for calling an AAS and a submodel operation by a client application:

- The calling application hast to be aware that endpoints may change at any time. If the application has cached an endpoint that is no longer vivid, the application needs to start the interaction to resolve the appropriate endpoint again from beginning.
- Endpoints for infrastructure interfaces like registries for AAS or repository are known at design time of the client application or configured manually before start up (further interaction diagram “automatically configure infrastructure“ to be created – repository endpoints will not be part of a mandatory client application-interaction).
- The Endpoint information of the submodel registry must be known to the client application. Subject to discussion for future interaction versions:
    a. will it be accessible via the AAS interface and therefore become mandatory part of a standard interaction
    b. how much “control” about submodels is implemented in the AAS and how are distributed submodels handled that are deployed in network areas not accessible by the AAS server application.
- AAS server application itself is instantiated and registered by calling an AAS registry interface (separate interaction diagram "instantiate and register" to be created)
- AAS-ID is known to the calling application (separate interaction diagram "Publish in discovery" to be created).

- Access to any API is allowed only if authenticated (mechanisms for authentication are to be described separately) and response follows a defined access rights model for all calls (separate interaction diagram "check access rights" to be created)
- direct access of subordinate structures will be made available via the definition of „superpaths“ (separate interactions to be defined see comment at bottom of diagram)

In the below depicted diagram, the interaction starts with a client application resolving the interface enpoint of an Asset Administration Shell with a known ID from the registry. AAS interface operations are used to identify appropriate submodels. In a last step the submodel interface endpoints are resolved via the submodel registry and defined submodel interface operations can be called.

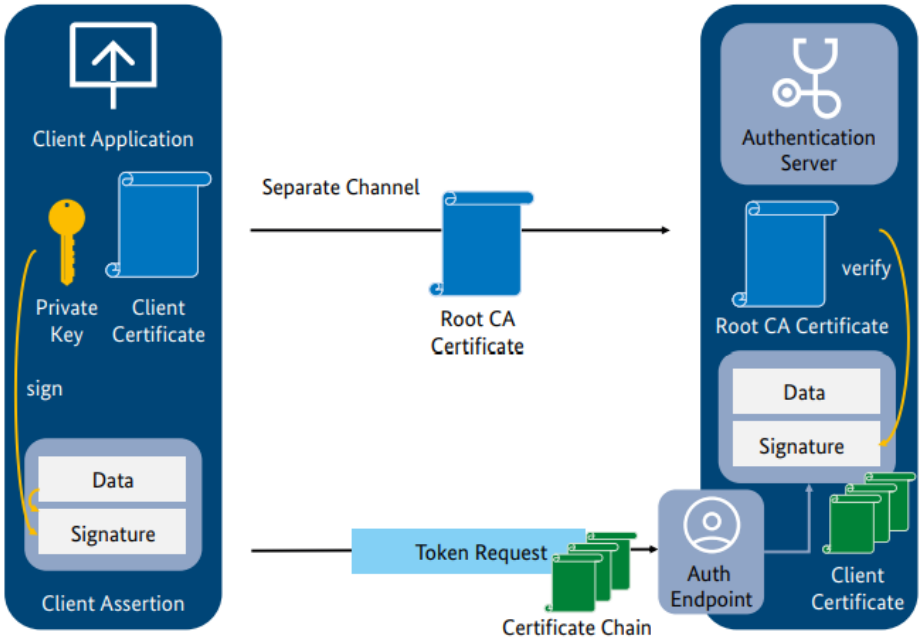


10.2 Security

In this clause the authentication by certificate chain is explained which has been developed by the security working group (AG3) of Plattform Industrie 4.0. Other authentication services (e.g. Username/Password, DID, Verifiable Credentials or IDS) may also be used to receive an Access Token for authorization.

In the following the most important steps for token based authentication of the HTTP/REST APIs are described. For more details see “Secure Downloadservice” (https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/secure_downloadservice.html). Figure 5 gives an overview.

Figure 5 The private_key_certchain_jwt method [...download service]



Source: Plattform Industrie 4.0

A Client Application uses a Client Certificate to create a Certificate Chain. The Certificate Chain can be checked on the Authentication Server by the corresponding Root CA Certificate, which is signed by a certification authority (CA). The Client Application sends the Certificate Chain to the Authentication Server as Token Request by a JSON Web Token (JWT). The JWT is signed by the client’s Private Key corresponding to the Client Certificate (JWT = Data + Signature).

If the authentication gets approved the Client Application receives an Access Token from the authentication server (not shown in Figure 1-2).

Such Access Token contains attributes from the client certificate (e.g. username, email address) which will be sent as HTTP Header Bearer Token to the AAS Server Application. The AAS Server Application will check, if the Access Token is signed by a trusted Authentication Server and will make the authorization according to the AAS security metamodel.

A running demo is explained in “Secure Downloadservice”. A corresponding server can be seen on https://admin-shell-io.com/5011/ with a related Security AAS on the bottom.

The AAS security metamodel does not deal with authentication but assumes that the user is already authenticated. The example security AAS is not standardized but only created for demonstration purposes. Since the used version of the AASX Package Explorer does not yet support the AAS security metamodel the needed information in subsequent steps like the access permission rules for AAS are modelled as a submodel.

The different security and authentication steps are explained in the video https://admin-shell-io.com/screencasts/security/Industrie_40_Security_with_AASX_Server.mp4.

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 interfaces are specified in a technology neutral way before defining technology specific APIs.

In this version of the specification HTTP/REST APIs are defined and mapped to the technology neutral specification.

In subsequent versions of this specification APIs using other technologies are planned to be supported, e.g. 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 in more detail is the 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. The operation provides its functionality through the following input and output parameters: • Input Parameter 1: human understandable description of the purpose of the input parameter 1 • … • Input Parameter N: human understandable description of the purpose of input parameter N • Output Parameter 1: human understandable description of the purpose of output parameter 1: human understandable description of the purpose of the input parameter 1 • …

Operation Name:	Name of the Operation: All individual words in the operation name are capitalized	
	• Output Parameter N: human understandable description of the purpose of output parameter N: 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 8.2.6. If no ***payload*** is mentioned as output parameter, the status code shall be SuccessNoContent in case of success, otherwise see Clause 8.2.6. 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. The Class has following attributes: • Attribute 1: human understandable description of the purpose of the attribute 1 • … • Attribute N: human understandable description of the purpose of the attribute N 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)	Explanation	Type	Kind	Card.
attribute1	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.	Type of the attribute 1	Kind of attribute 1 is defined with semantics of UML (for details see Annex Legend for UML Modelling): • attr: attribute (Type is no object type but a data type, it is just a value) • aggr: composite aggregation (composition) (does not exist independent of its parent) • ref*: shared aggregation (does exist independent of its parent)	Cardinality of the attribute 1
…				
attributeN	Human understandable description of the attribute N of the class. Note:	Type of the attribute N	Kind of attribute N is defined with semantics of UML (for details see Annex Legend for UML Modelling): • attr:	Cardinality of the attribute N

Class Name	Name of the Class: All individual words in the clas name are capitalized			
Explanation	Human understandable description of the class. The Class has following attributes: • Attribute 1: human understandable description of the purpose of the attribute 1 • … • Attribute N: human understandable description of the purpose of the attribute N 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.			
	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.		attribute (Type is no object type but a data type, it is just a value) • aggr: composite aggregation (composition) (does not exist independent of its parent) • ref*: shared aggregation (does exist independent of its parent)	

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. The Enumeration has following literals: • Literal 1: human understandable description of the purpose of the literal 1 • … • Literal N: human understandable description of the purpose of the literal N 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 6 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 13. In this case the instance of *Class2* is navigable via the instance of the owning class *Class1*.[6]

Figure 6 Class

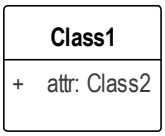


Figure 7 shows that *Class4* is inheriting all member elements from *Class3*. Or in other word, *Class3* is a generalization of *Class4, Class4* is a specialization of *Class3*. This means that each instance of *Class4* is also an instance of *Class3*. An instance of the *Class4* has the attributes *attr1* and *attr2* whereas instances of *Class3* only have the attribute *attr1*.

[6] „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]"

Figure 7 Inheritance/Generalization

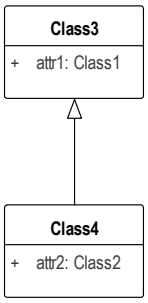


Figure 8 defines the required and allowed multiplicity/cardinality within an association between instances of *Class1* and *Class2*. In this example an instance of *Class2* is always related to exactly one instance of *Class1*. An instance of *Class1* is either related to none, one or more (unlimited, i.e. no constraint on the upper bound) instances of *Class2*. The relationship can change over time.

Multiplicity constraints can also be added to attributes and aggregations.

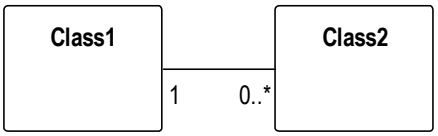
The notation of multiplicity is as follows:

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

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

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

Figure 8 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 9 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 9 Ordered Multiplicity

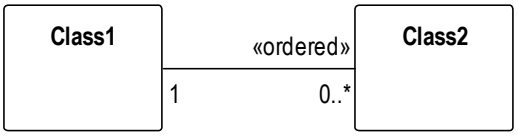


Figure 10 shows that the member ends of an association can be named as well. I.e. an instance of *Class1* can be in relationship “relation” to an instance of *Class2*. Vice versa the instance of *Class2* is in relationship “reverseRelation” to the instance of *Class1*.

Figure 10 Association

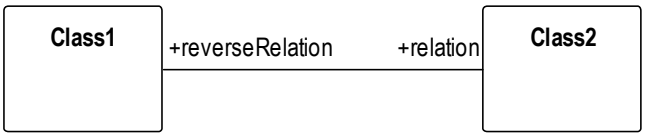


Figure 11 shows a composition, also called a composite aggregation. A composition is a binary association. It groups a set of instances. The individuals in the set are typed as specified by *Class2*. The multiplicity of instances of *Class2* to *Class1* is always 1 (i.e. upper-bound and lower-bound have 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 12 the composition is shown using an association relationship with a filled diamond as composition adornment.

Figure 11 Composition (composite aggregation)

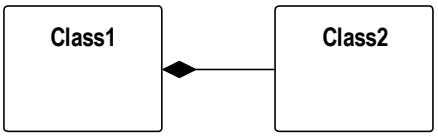


Figure 12 show an aggregation. An aggregation is a binary association. In contrast to a composition an instance of *Class2* can be shared by several instances of *Class1*. In Figure 12 the shared aggregation is shown using an association relationship with a hallow diamond as aggregation adornment.

Figure 12 Aggregation

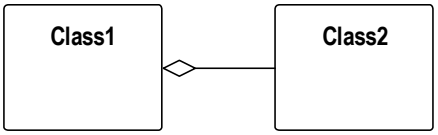


Figure 13 shows that the attribute notation can be used for an association end owned by a class. In this example the attribute name is “attr” and the elements of this attribute are typed with *Class2.* The multiplicity, here “0..*”, is added in square brackets. If the aggregation is ordered then this is added in curly brackets like in this example.

Figure 13 Navigable Attribute Notation for Associations

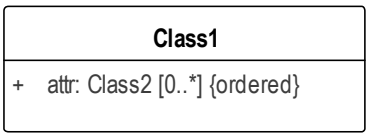


Figure 14 shows that there is a dependency relationship between *Class1* and *Class2*. In this case the dependency means that *Class1* depends on *Class2.* Why is this: because the type of attribute *attr* depends on the specification of class *Class2.* A dependency is shown as dashed arrow between two model elements.

Figure 14 Dependency

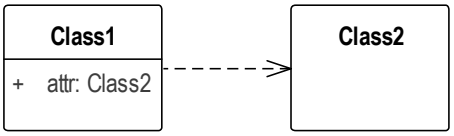


Figure 15 shows an abstract class. It uses the stereotype <<abstract>>. There are no instances of abstract classes. They are typically used to specific member elements that are then inherited by non-abstract classes.

Figure 15 Abstract Class

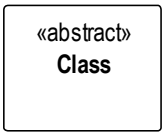


Figure 16 shows a package with name “Package2”. A package is a namespace for its members. In this example the member belonging to *Package2* is class *Class2*.

Figure 16 Package

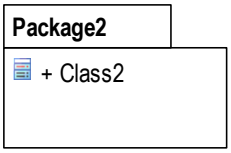
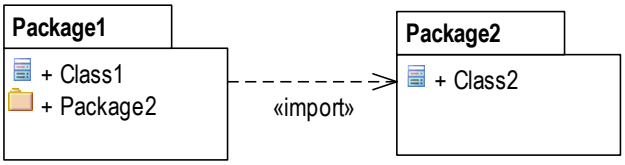


Figure 17 shows that all elements in *Package2* are imported into the namespace defined by *Package1*. This is a special dependency relationship between the two packages with stereotype <<import>>.

Figure 17 Imported Package



An enumeration is a data type whose values are enumerated as literals. Figure 18 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 18 Enumeration

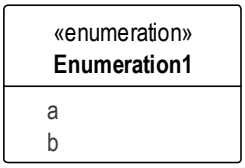


Figure 19 show the definition of the data type with name “DataType1”. A data type is a type whose instances are identified only by their value. It is a class with stereotype <<dataType>>.

Figure 19 Data Type

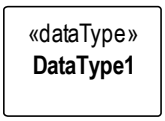


Figure 20 shows a primitive data type with name “int”. Primitive data types are predefined data types, without any substructure. The primitive data types are defined outside UML.

Figure 20 Primitive Data Type

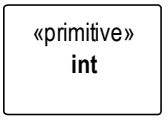


Figure 21 shows how a note can be attached to an element, in this example to class “Class1”.

Figure 21 Note

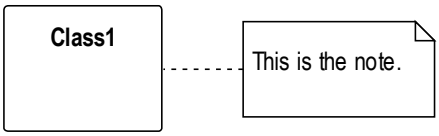
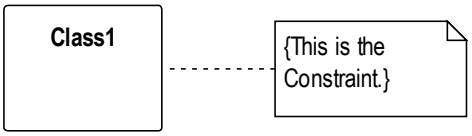


Figure 22 shows how a constraint is attached to an element, in this example to class “Class1”.

Figure 22 Constraint



ii. Notes to Graphical Representation

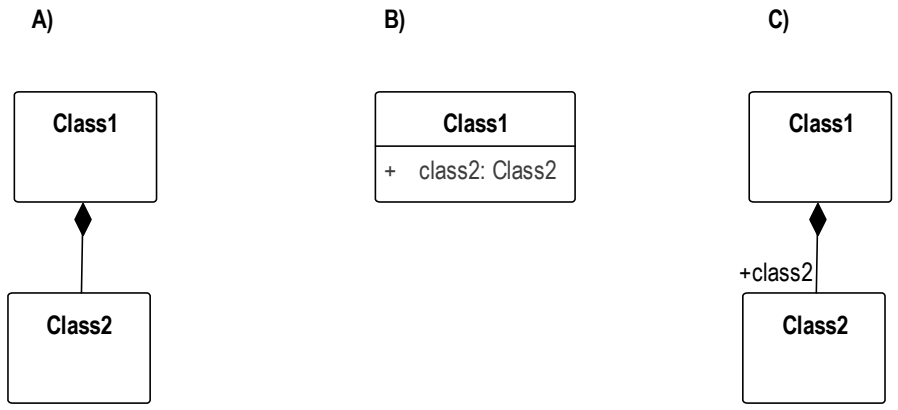
In the following specific graphical modelling rules used in this specification are explained that are not included in this form in [5].

Figure 23 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 23 Graphical Representations of Composite Aggregation/Composition



In Figure 24 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 24 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 24 Graphical Representation of Shared Aggregation

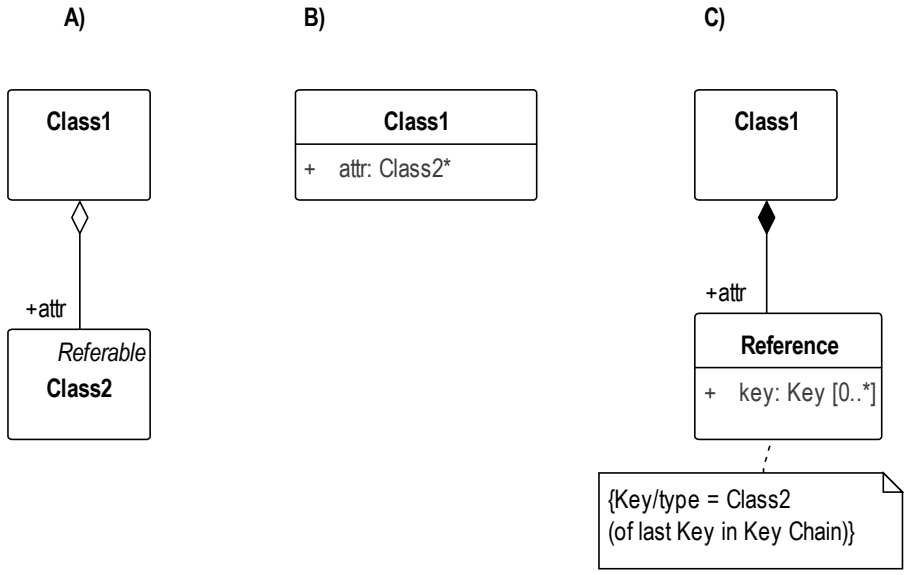
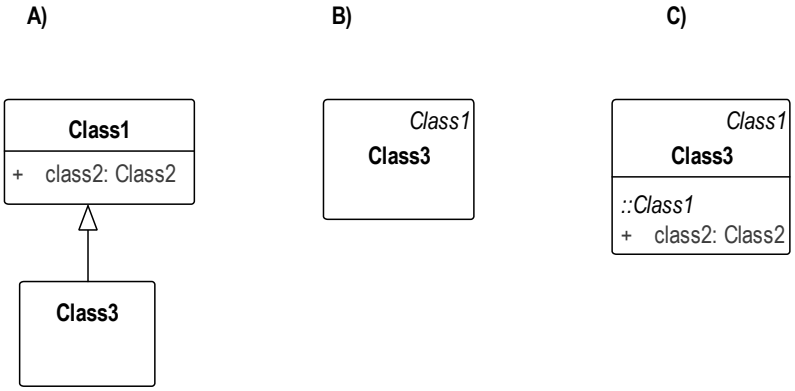


Figure 25 show different graphical representations of generalization. Variant A) is the classical graphical representation as defined in [5]. Variant B) is a short form if *Class1* is not on the same diagram. To see from which class *Class3* is inheriting the name of the class is depicted in the upper right corner.

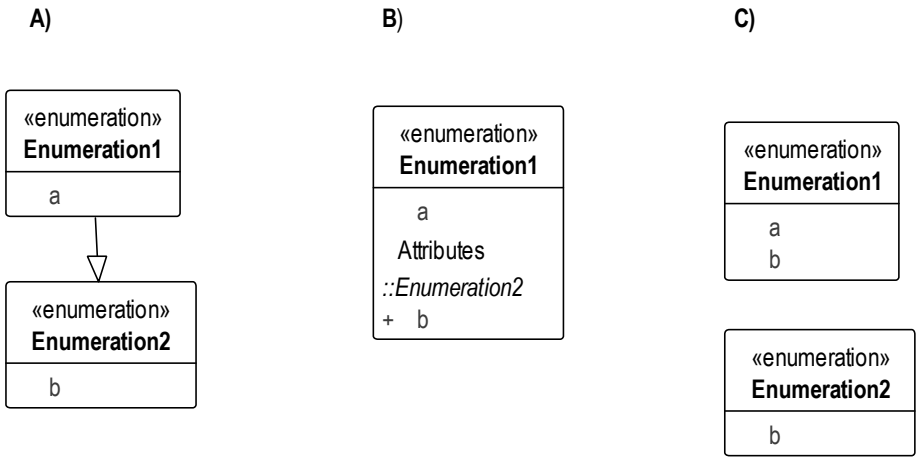
Variant C) is not only showing from which class Class3 instances are inheriting but also what they are inheriting. This is depicted by the class name it is inheriting from followed by "::" and then the list of all inherited elements – here attribute *class2*. Typically, the inherited elements are not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

Figure 25 Graphical Representation of Generalization/Inheritance



In Figure 26 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 26 Graphical Representation for Enumeration with Inheritance



Annex C. ValueOnly-Serialization Example

The following example shows the ValueOnly-serialization for an entire Submodel that validates against the JSON-schema specified in 9.4.3.


```
{
  "PropertyIdShortNumber": 5000,
  "PropertyIdShortString": "MyTestStringValue",
  "PropertyIdShortBoolean": true,
  "MyMultiLanguageProperty": {
    "de": "Das ist ein deutscher Bezeichner",
    "en": "That's an English label"
  },
  "MyRange": {
    "min": 3,
    "max": 15
  },
  "MyFile": {
    "mimeType": "application/pdf",
    "value": "SafetyInstructions.pdf"
  },
  "MyBlob": {
    "mimeType": "application/octet-stream",
    "value": "VGhpcyBpcyBteSBibG9i"
  },
  "MyEntity": {
    "statements": {
      "MaxRotationSpeed": 5000
    },
    "entityType": "SelfManagedEntity",
    "globalAssetId": ["http://customer.com/demo/asset/1/1/MySubAsset"]
  },
  "MyReference": [
    {
      "type": "Submodel",
      "value": "http://customer.com/demo/aas/1/1/1234859590"
```

```
    },
    {
      "type": "Property",
      "value": "MaxRotationSpeed"
    }
  ],
  "MyBasicEvent": {
    "observed": [
      {
        "type": "Submodel",
        "value": "http://customer.com/demo/aas/1/1/1234859590"
      },
      {
        "type": "Property",
        "value": "CurrentValue"
      }
      ],
  },
  "MyRelationship": {
    "first": [
      {
        "type": "Submodel",
        "value": "http://customer.com/demo/aas/1/1/1234859590"
      },
      {
        "type": "Property",
        "value": "PlusPole"
      }
    ],
    "second": [
      {
        "type": "Submodel",
        "value": "http://customer.com/demo/aas/1/0/1234859123490"
      },
```

```
      {
        "type": "Property",
        "value": "MinusPole"
      }
    ]
  },
  "MyAnnotatedRelationship": {
    "first": [
      {
        "type": "Submodel",
        "value": "http://customer.com/demo/aas/1/1/1234859590"
      },
      {
        "type": "Property",
        "value": "PlusPole"
      }
    ],
    "second": [
      {
        "type": "Submodel",
        "value": "http://customer.com/demo/aas/1/0/1234859123490"
      },
      {
        "type": "Property",
        "value": "MinusPole"
      }
    ],
    "annotation": [
      {
        "AppliedRule" : "TechnicalCurrentFlowDirection"
      }
    ]
  }
}
```

Annex D. 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/IP/Redaktion/DE/Downloads/Publikation/Details_of_the_Asset_Administration_Shell_Part1_V3.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 3.0RC02. 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/

[6] 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

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

Annex E. Change Notes

1. General

- * Means not backward compatible
- (*) means not backward compatible but just renaming

2. Interface Changes w.r.t. V1.0RC01

BWC	Interface Change	Kind of Change	Comment
*	Asset Administration Shell	changed	Renamed: RemoveSubmodelReference to DeleteSubmodelReference Removed: PutSubmodelReference, PatchAssetAdministrationShell New: GetAssetInformation PutAssetInformation GetAllSubmodelReferences PostSubmodelReference
*	Submodel	changed	Removed: GetAllSubmodelElementsByParentPathAndSemanticId, GetAllSubmodelElementsBySemanticId New: PutSubmodel, PostSubmodelElement, PostSubmodelElementByPath
*	Asset Administration Shell Serialization	changed	Renamed: GetSerializationByIds to GenerateSerializationByIds Removed: GetAASX
	AASX File Server	added	New interface
(*)	Asset Administration Shell Registry	changed	Renamed: PutAssetAdministrationShellDescriptor to PutAssetAdministrationShellDescriptorById New: PostAssetAdministrationShellDescriptor

(*)	Submodel Registry	changed	Renamed: PutSubmodelDescriptor to PutSubmodelDescriptorById New: PostSubmodelDescriptor
(*)	Asset Administration Shell Repository	changed	Renamed: GetAllAssetAdministrationShellsById to GetAssetAdministrationShellById, PutAssetAdministrationShell to PutAssetAdministratioShellById New: PostAssetAdministrationShell
(*)	Submodel Repository	changed	Renamed: PutSubmodel to PutSubmodelById New: PostSubmodel
(*)	Asset Administration Shell Basic Discovery	changed	Removed: GetAllAssetAdministrationShellIdsByAssetId, PutAssetId New: GetAllAssetAdministrationShellIdsByAssetLink, GetAllAssetLinksById, PutAllAssetLinksById, DeleteAllAssetLinksById
(*)	Submodel Discovery Basic	deleted	
(*)	Concept Description Repository	changed	Renamed: GetAllConceptDescriptionsWtihDataSpecificationReference to GetAllConceptDescriptionsByDataSpecificationReference, PutConceptDescription to PutConceptDescriptionById New: PostConceptDescription

3. Operation Changes w.r.t. V1.0RC01

Operation Change Old	Operation Change New	Kind of Change	Comment
PatchAssetAdministrationShell		removed	
PutSubmodelReference		removed	Substituted by PostSubmodelReference
	PostSubmodelReference	New	For PutSubmodelReference
RemoveSubmodelReference	DeleteSubmodelReference	rename	
	GetAllSubmodelReferences	New	
	PostSubmodelReference	New	
	GetAssetInformation	New	
	PutAssetInformation	New	
	PutSubmodel	new	
	PostSubmodelElement	new	
	PostSubmodelElementByPath	new	
GetAllSubmodelElementsByParentPathAndSemanticId		removed	
GetAllSubmodelElementsBySemanticId		removed	
GetAASX		removed	
GetSerializationByIds	GenerateSerializationByIds	rename	
	GetAllAASXPackageIds	new	
	GetAASXByPackageId	new	
	PostAASXPackage	new	
	PutAASXByPackageId	new	
	DeleteAASXByPackageId	new	

PutAssetAdministrationShellDescriptor	PutAssetAdministrationShellDescriptorById	rename	Naming pattern byId
	PostAssetAdministrationDescriptor	new	
PutSubmodelDescriptor	PutSubmodelDescriptorById	rename	Naming pattern byId
	PostSubmdeoDescriptor	new	
GetAllAssetAdministrationShellsById	GetAssetAdministrationShellById	rename	Naming pattern ressource singular
	PostAssetAdministrationShell	new	
PutAssetAdministrationShell	PutAssetAdministrationShellById	rename	Naming pattern byId
PutSubmodel	PutSubmodelById	rename	Naming pattern byId
	PostSubmodel	new	
GetAllAssetAdministrationShellIdsByAssetId		removed	substituted by GetAllAssetAdministrationShellIdsByAssetLink and GetAllAssetLinksById
PutAssetId		removed	Substituted by PutAllAssetLinksById and DeleteAllAssetLinksById
	GetAllAssetAdministrationShellIdsByAssetLink	new	Before: GetAllAssetAdministrationShellIdsByAssetId
	GetAllAssetLinksById	new	
	PutAllAssetLinksById	new	
	DeleteAllAssetLinksById	new	
GetAllSubmodelIdsBySemanticId		removed	
GetAllConceptDescriptionsWithDataSpecificationReference	GetAllConceptDescriptionsByDataSpecificationReference	rename	Renaming With ➔ By
PutConceptDescription	PutConceptDescriptionById	rename	Naming pattern byId
	PostConceptDescription	new	

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 joint 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“ and the working group “Open Technology” of the Industrial Digital Twin Association.

www.plattform-i40.de