Contents

1 Preamble

1.1 Editorial notes

This document was developed from November 2021 to June 2022 by the joint working group “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.

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

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.

1.3 Structure of the Document

An introduction to the topic is given in Clause 2. General topics are discussed in Clause 3. The technology neutral specification of the interfaces of the Asset Administration Shell can be found in Clause 4 to 9.

In Clause 10 the API specification for HTTP/REST is defined. Annex B 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
IDTA	Industrial Digital Twin Association
IEC	International Electrotechnical Commission
IRDI	International Registration Data Identifier
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
MIME	Multipurpose Internet Mail Extensions
OPC	Open Packaging Conventions (ECMA-376, ISO/IEC 29500-2)
OPC	Open Platform Communications
OPCF	OPC Foundation
OPC UA	OPC Unified Architecture
PDF	Portable Document Format
RAMI4.0	Reference Architecture Model Industrie 4.0
RDF	Resource Description Framework
REST	Representational State Transfer
RFC	Request for Comment
ROA	Resource Oriented Architecture
SOA	Service Oriented Architecture
UML	Unified Modeling Language
URI, URL, URN	Uniform Resource Identifier, Locator, Name
VDE	Verband der Elektrotechnik Elektronik Informationstechnik e. V.
VDI	Verein Deutscher Ingenieure e.V.
VDMA	Verband Deutscher Maschinen- und Anlagenbau e.V.
W3C	World Wide Web Consortium
XML	eXtensible Markup Language
ZIP	archive file format that supports lossless data compression
ZVEI	Zentralverband Elektrotechnik- und Elektronikindustrie e. V.

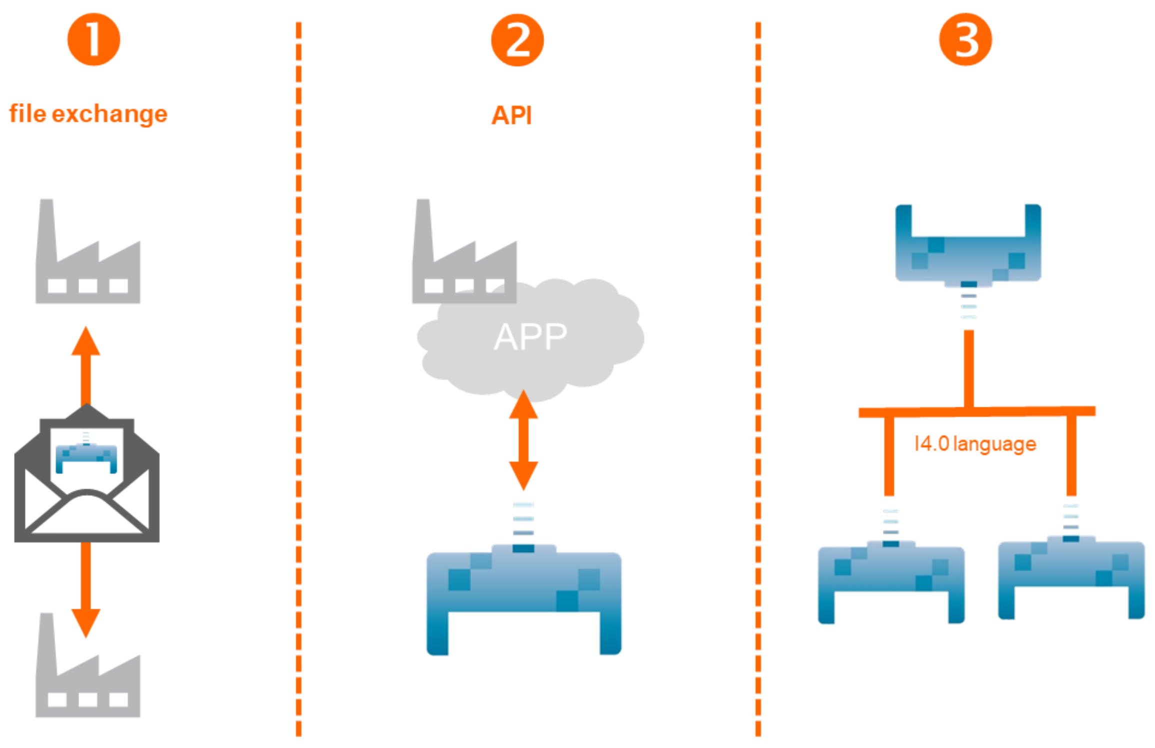
2 Introduction

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 Model 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:

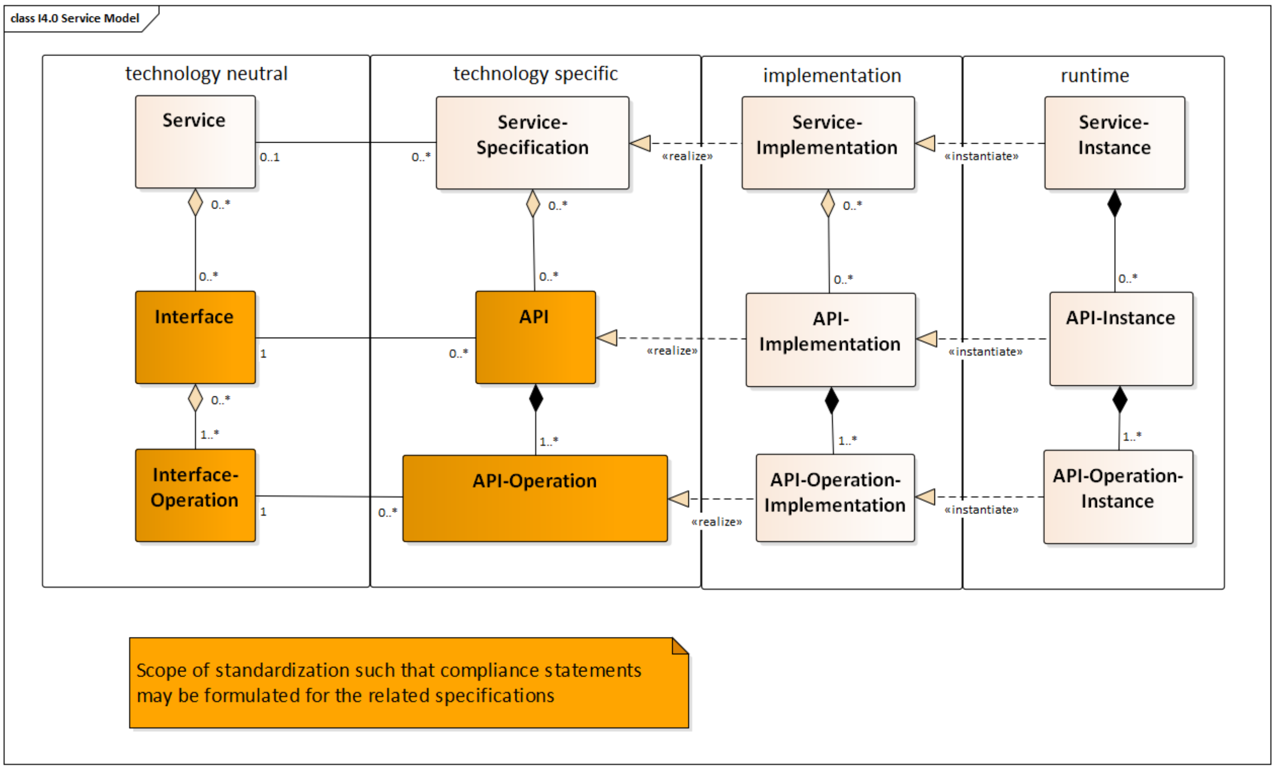
- Service-Implementation: service realized in a selected implementation language following the specification in the Service Specification description on the technology-specific level.
- API-Implementation: set of operations realized in a selected implementation language following the specification in 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 4 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.

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/RC03

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

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
SpecificAssetId	The name of the specific asset identifier or the predefined name “*globalAssetId*” that would refer to the *AssetInformation/globalAssetId*.
SemanticId	Identifier of the semantic definition

3.6 Relation of interfaces

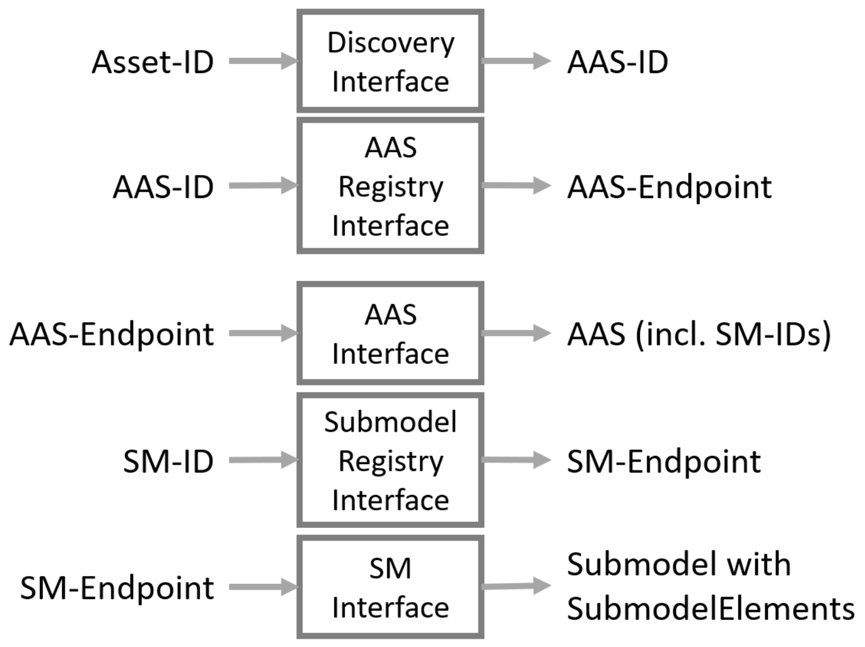
In the following chapters several interfaces are defined, which work together as a system, and which support different deployment scenarios.

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 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 (https://app.swaggerhub.com/apis/Plattform_i40/AssetAdministrationShell-Environment/V1.0RC03):

/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}/attachment
/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}/attachment
/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 (https://app.swaggerhub.com/apis/Plattform_i40/AssetAdministrationShell-API/V1.0RC03):

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

Note: Identifiers are BASE64-URL-encoded in the API, i.e. {aas-identifier}, {submodel-identifier} and {cd-identifier}. The {idShortPath} is URL-encoded in the API..

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/RC03	
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/RC03	
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/RC03	
Name	Type	Description
Input Parameter		
outputModifier	OutputModifier	Determines the result format filtering of the response
Output Parameter		
statusCode	StatusCode	Status code
payload	References[0..*]	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/RC03	
Name	Type	Description
Input Parameter		
submodelRef	Reference	Reference to the Submodel
Output Parameter		
statusCode	StatusCode	Status code
payload	Reference	Created Submodel Reference

4.2.6 Operation DeleteSubmodelReference

Operation Name	DeleteSubmodelReference	
Explanation	Deletes the Submodel Reference from the Asset Administration Shell	
semanticId	https://admin-shell.io/aas/API/DeleteSubmodelReference/1/0/RC03	
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/RC03

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/RC03	
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
GetFileByPath	Returns a specific file content from the Submodel at a specified path
PutSubmodel	Updates the Submodel

Interface: Submodel	
Operation Name	**Description**
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 must be handled in a proprietary way.
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 must be handled in a proprietary way.
PutSubmodelElementByPath	Updates an existing submodel element at a specified path within the submodel elements hierarchy
PutFileByPath	Updates the file content of an existing submodel element at a specified path within the submodel elements hierarchy
SetSubmodelElementValueByPath	Sets the value of the submodel element at a specified path according to the protocol-specific RAW-value payload
DeleteSubmodelElementByPath	Deletes a submodel element at a specified path within submodel elements hierarchy
InvokeOperationSync	Synchronously invokes an Operation at a specified path with a client timeout in ms
InvokeOperationAsync	Asynchronously invokes an Operation at a specified path with a client timeout in ms
GetOperationAsyncResult	Returns the OperationResult of an asynchronously invoked operation

528

529 ### 4.3.2 Operation GetSubmodel

Operation Name	GetSubmodel
Explanation	Returns the Submodel
semanticId	https://admin-shell.io/aas/API/GetSubmodel/1/0/RC03

Operation Name	GetSubmodel	
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/RC03	
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/RC03	
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

534

535 ### 4.3.5 Operation GetFileByPath

Operation Name	GetFileByPath	
Explanation	Returns a specific file content from the Submodel at a specified path	
semanticId	https://admin-shell.io/aas/API/GetFileByPath/1/0/RC03	
Name	Type	Description
Input Parameter		
path	Key[1..*]	IdShort-Path via relative Reference/Keys to a submodel element
Output Parameter		
statusCode	StatusCode	Status code
payload	File Content	Requested content of file

536

537 ### 4.3.6 Operation PutSubmodel

Operation Name	PutSubmodel	
Explanation	Updates the Submodel	
semanticId	https://admin-shell.io/aas/API/PutSubmodel/1/0/RC03	
Name	Type	Description
Input Parameter		
submodel	Submodel	Submodel object
Output Parameter		

statusCode	StatusCode	Status code
payload	Submodel	Updated submodel

538 ### 4.3.7 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 must be handled in a proprietary way.	
semanticId	https://admin-shell.io/aas/API/PostSubmodelElement/1/0/RC03	
Name	Type	Description
Input Parameter		
submodelElement	SubmodelElement	Submodel element object
Output Parameter		
statusCode	StatusCode	Status code
payload	SubmodelElement	Created submodel element

539 ### 4.3.8 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 must be handled in a proprietary way.	
semanticId	https://admin-shell.io/aas/API/PostSubmodelElementByPath/1/0/RC03	
Name	Type	Description
Input Parameter		
path	Key[0..*]	IdShort-Path via relative Reference/Keys to a submodel element.
submodelElement	SubmodelElement	Submodel element object
Output Parameter		
statusCode	StatusCode	Status code

Operation Name	PostSubmodelElementByPath	
payload	SubmodelElement	Created submodel element

4.3.9 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/RC03	
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

4.3.10 Operation PutFileByPath

Operation Name	PutFileByPath	
Explanation	Updates the file content of an existing submodel element at a specified path within the submodel elements hierarchy	
semanticId	https://admin-shell.io/aas/API/PutFileByPath/1/0/RC03	
Name	Type	Description
Input Parameter		
path	Key[1..*]	IdShort-Path via relative Reference/Keys to a submodel element
payload	File Content	Updated content of file (as file stream and not as JSON)
Output Parameter		

statusCode	StatusCode	Status code

4.3.11 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/RC03	
Name	Type	Description
Input Parameter		
path	Key[1..*]	IdShort-Path via relative Reference/Keys to a submodel element
payload	Corresponding type of submodel element value	The new value of the submodel element to be set
Output Parameter		
statusCode	StatusCode	Status code

4.3.12 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/RC03	
Name	Type	Description
Input Parameter		
path	Key[1..*]	IdShort-Path via relative Reference/Keys to a submodel element
Output Parameter		
statusCode	StatusCode	Status code

548 4.3.13 Operation InvokeOperationSync

Operation Name	InvokeOperationSync	
Explanation	Synchronously invokes an Operation at a specified path	
semanticId	https://admin-shell.io/aas/API/InvokeOperationSync/1/0/RC03	
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		
statusCode	StatusCode	Status code
payload	OperationResult	Operation Result

549 4.3.14 Operation InvokeOperationAsync

Operation Name	InvokeOperationAsync	
Explanation	Asynchronously invokes an Operation at a specified path	
semanticId	https://admin-shell.io/aas/API/InvokeOperationAsync/1/0/RC03	
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

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

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

4.3.15 Operation GetOperationAsyncResult

Operation Name	GetOperationAsyncResult	
Explanation	Returns the OperationResult of an asynchronously invoked operation	
semanticId	https://admin-shell.io/aas/API/GetOperationAsnycResult/1/0/RC03	
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).

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

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/RC03	
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/RC03	
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/RC03	
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

563 ### 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/RC03	
Name	Type	Description
Input Parameter		
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
packageId	String	New Package ID

564 ### 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/RC03	
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/RC03	
Name	Type	Description
Input Parameter		
packageId	String	Package ID from the package list
Output Parameter		
statusCode	StatusCode	Status code

5 Interfaces Registration

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	
semanticId	https://admin-shell.io/aas/API/GetAllAssetAdministrationShellDescriptors/1/0/RC03	
Name	**Type**	**Description**
Input Parameter		
Output Parameter		
statusCode	StatusCode	Status code

Operation Name	GetAllAssetAdministrationShellDescriptors	
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/RC03	
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/RC03	
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/RC03	
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	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/RC03	
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

Interface:Submodel Registry	
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/RC03	
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/RC03	
Name	Type	Description
Input Parameter		
submodelIdentifier	Identifier	The Submodel’s unique id
Output Parameter		

Operation Name	GetSubmodelDescriptorById	
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/RC03	
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/RC03	
Name	Type	Description
Input Parameter		
submodel Descriptor	SubmodelDescriptor	Object containing the Submodel's identification and endpoint information
Output Parameter		
statusCode	StatusCode	Status code

Operation Name	PutSubmodelDescriptorById	
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/RC03	
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 idShort is out of scope and must be handled in a proprietary way.
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/RC03

Operation Name	GetAllAssetAdministrationShells	
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/RC03	
Name	Type	Description
Input Parameter		
id	Identifier	The Asset Administration Shell’s unique id
outputModifier	OutputModifier	Determines the result format filtering of the response
Output Parameter		
statusCode	StatusCode	Status code
payload	AssetAdministrationShell	Requested Asset Administration Shell

6.2.4 Operation GetAllAssetAdministrationShellsByAssetId

Operation Name	GetAllAssetAdministrationShellsByAssetId	
Explanation	Returns all Asset Administration 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/RC03	
Name	Type	Description
Input Parameter		

Operation Name	GetAllAssetAdministrationShellsByAssetId	
key	string	The name of the specific asset identifier or the predefined name “*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

619 **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/RC03	
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

620

621 **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 idShort is out of scope and must be handled in a proprietary way.
semanticId	https://admin-shell.io/aas/API/PostAssetAdministrationShell/1/0/RC03

Operation Name	PostAssetAdministrationShell	
Name	Type	Description
Input Parameter		
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/RC03	
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/RC03	
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. Note: The creation of the idShort is out of scope and must be handled in a proprietary way.
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/RC03	
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/RC03	
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/RC03	
Name	Type	Description
Input Parameter		
semanticId	Reference	Identifier of the semantic definition
outputModifier	OutputModifier	Determines the result format filtering of the response
Output Parameter		
statusCode	StatusCode	Status code
payload	Submodel[0..*]	Requested Submodels

6.3.5 Operation GetAllSubmodelsByIdShort

Operation Name	GetAllSubmodelsByIdShort	
Explanation	Returns all Submodels with a specific *idShort*	
semanticId	https://admin-shell.io/aas/API/GetAllSubmodelsByIdShort/1/0/RC03	
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 idShort is out of scope and must be handled in a proprietary way.	
semanticId	https://admin-shell.io/aas/API/PostSubmodel/1/0/RC03	
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/RC03	
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/RC03	
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 idShort is out of scope and must be handled in a proprietary way.
PutConceptDescriptionById	Updates an existing Concept Description
DeleteConceptDescriptionById	Deletes a Concept Description

642

643 ### 6.4.2 Operation GetAllConceptDescriptions

Operation Name	GetAllConceptDescriptions	
Explanation	Returns all Concept Descriptions	
semanticId	https://admin-shell.io/aas/API/GetAllConceptDescriptions/1/0/RC03	
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

644 ### 6.4.3 Operation GetConceptDescriptionById

Operation Name	GetConceptDescriptionById
Explanation	Returns a specific Concept Description

semanticId	https://admin-shell.io/aas/API/GetConceptDescriptionById/1/0/RC03	
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

645

646 ### 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/RC03	
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

647

648 ### 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/RC03	
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/RC03	
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 idShort is out of scope and must be handled in a proprietary way.	
semanticId	https://admin-shell.io/aas/API/PostConceptDescription/1/0/RC03	
Name	Type	Description

Operation Name	PostConceptDescription	
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/RC03	
Name	Type	Description
Input Parameter		
conceptDescription	ConceptDescription	Concept Description object
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/RC03	
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-pairs	
semanticId	https://admin-shell.io/aas/API/GetAllAssetAdministrationShellIdsByAssetLink/1/0/RC03	
Name	Type	Description
Input Parameter		
assetIds	SpecificAssetId [1..*]	The specific sssetId 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*.

Operation Name	GetAllAssetAdministrationShellIdsByAssetLink	
Output Parameter		
statusCode	StatusCode	Status code
payload	Identifier[0..*]	Identifiers of all Asset Administration Shells which contain all asset identifier key value pairs in their asset information, i.e. AND-match of key value pairs per Asset Administration Shell

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/RC03	
Name	Type	Description
Input Parameter		
aasIdentifier	string	The Asset Administration Shell's unique id
Output Parameter		
statusCode	StatusCode	Status code
payload	SpecificAssetId	Requested Asset identifier, which could be the globalAssetId or specificAssetIds. Note: The name of the SpecificAssetId for the globalAssetId is defined in chapter 3.5. It is the predefined name "*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/RC03

Operation Name	PostAllAssetLinksById	
Name	Type	Description
Input Parameter		
aasIdentifier	string	The Asset Administration Shell's unique id
assetLinks	SpecificAssetId	Asset identifier, which could be the globalAssetId or specificAssetIds. Note: The name 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	SpecificAssetId	Asset identifier 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/RC03	
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, e.g., AssetAdministrationShell, Submodel, Identifier etc., that are specified in Part 1, please refer to the specification in [1]. In this clause, only additional classes are defined that are needed for the communication with the API.

8.2 Metamodel Specification Details: Designators

The following type definitions are used to describe specific metamodel elements like Asset Administration Shells and Submodels regarding their network and 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/RC03			
Attribute	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/RC03			
Attribute	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.	SpecificAssetId	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/RC03

Attribute	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/RC03			
Attribute	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": "application/soap+xml"
        },
        "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 any needed updates will be made.This class is not part of the metamodel.			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/ProtocolInformation/1/0/RC03			
Attribute	Explanation	Type	Kind	Card.
endpointAddress	The endpoint address as an URL (also denoted as href)	string	attr	1
endpointProtocol	Either scheme of endpointAdress or scheme + further information. Scheme denotes the highest level of doubtless transmission.	string	attr	0..1
endpointProtocolVersion	Array of strings, each entry represents one supported version at this very endpoint, the entry shall be formatted according to the regulations of the protocol specified in the href	string	attr	0..1
subprotocol	Allows for referencing Sub-protocols that may be used in the	string	attr	0..1

	context of that endpoint e.g. “OPC Basic SOAP” or UA Binary			
subprotocolBody	If the sub-protocol field is present a subprotocolBody might be given to hold extra information, e.g. node and namespace in an OPC UA server	string	attr	0..1
subprotocolBodyEncoding	IF subprotocolBody is present the encoding might be explicitly defined, otherwise it shall default to subprotocols encoding scheme	string	attr	0..1
securityAttributes	Array of securityAttribute objects, each attribute has 3 properties: { **type** = Enum security type or standard: • ‘NONE’, • 'RFC_TLSA' - TLSA according to rfc6698 • 'W3C_DID' - W3C DID document , **key** = security attribute key according to standard definitions of the security type, **value** = security attribute value e.g. DANE TLSA Ressource Record } The securityAttribute objects are treated as possible alternatives (logical “or”)	string	attr	1..*

700 ### 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.

8.2.6.1 Generic Status Codes

Successful operations return one of the success status codes and their respective payload. Unsuccessful operations return one of the failure status codes and a result object as defined in Clause 8.2.6.2.

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

8.2.6.2 General Result Object

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

Class Name	Result			
Explanation	The result object			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/Result/1/0/RC03			
Attribute	Explanation	Type	Kind	Card.
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/RC03			
Attribute	Explanation	Type	Kind	Card.

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/RC03			
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/RC03
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

8.2.6.3 Operation Objects

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

8.2.6.3.1 OPERATIONREQUEST

Class Name	OperationRequest
Explanation	The operation request object
Inherits from	--

semanticId	https://admin-shell.io/aas/API/DataTypes/OperationRequest/1/0/RC03			
Attribute	**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

8.2.6.3.2 OPERATIONRESULT

Class Name	OperationResult			
Explanation	The operation's invocation result object			
Inherits from	--			
semanticId	https://admin-shell.io/aas/API/DataTypes/OperationResult/1/0/RC03			
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

8.2.6.3.3 ENUMERATION EXECUTIONSTATE

Enumeration	ExecutionState
Explanation	The operation's invocation result state

semanticId	https://admin-shell.io/aas/API/DataTypes/ExecutionState/1/0/RC03
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

8.2.6.3.4 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/RC03			
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.
Metadata	Only metadata of an element or child elements but not the value is returned.
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 SubmodelElementCollection, a SubmodelElementList, a AnnotatedRelationshipElement or an Entity.

3. Enumeration: Extent

The third enumeration *Extent* indicates to which extent the response content is being serialized. Please note that at this stage the listed values could also be represented as binary values on BLOB-elements, but for the sake of extension this is kept as a generic extent value.

4. Value	5. 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 due to their generic nature. 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.

Resource Name	Level Modifier	Content Modifier	Extent Modifier
Asset Administration Shell	No	Normal/Reference	No
Submodel Reference	No	No	No
Submodel	Deep/Core	Normal/ Metadata/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
SubmodelElements			
SubmodelElementCollection	Deep/Core	Normal/ Metadata/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
SubmodelElementList	Deep/Core	Normal/ Metadata/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
Entity	Deep/Core	Normal/ Metadata/Value/Reference/Path	WithoutBLOBValue/ WithBLOBValue
BasicEventElement	No	Normal/ Metadata/Value/Reference	No
Capability	No	Normal/Reference	No
Operation	No	Normal/Reference	No
DataElements			
Property	No	Normal/ Metadata/Value/Reference	No
MultilanguageProperty	No	Normal/ Metadata/Value/Reference	No
Range	No	Normal/ Metadata/Value/Reference	No
RelationshipElement	No	Normal/ Metadata/Value/Reference	No

Resource Name	Level Modifier	Content Modifier	Extent Modifier
AnnotatedRelationshipElement	No	Normal/ Metadata/Value/Reference	No
Blob	No	Normal/ Metadata/Value/Reference	WithoutBLOBValue/ WithBLOBValue
File	No	Normal/ Metadata/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, that can be separately requested via *Content* modifier set to *Metadata*, in each 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 SubmodelElementCollection resp. for their included SubmodelElements,
- ReferenceElement,
- RelationshipElement + AnnotatedRelationshipElement,
- Entity
- BasicEventElement

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 an array of JSON objects for each language of the *MultiLanguageProperty* with the language as name and the corresponding localized string as value of the respective JSON property. 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 content type named `${File/contentType}` resp. `${Blob/contentType}`. The seconds refers to the value named "value" `${File/value}` resp. `${Blob/value}`.

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

    - *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 a *SpecificAssetId*. The third property is named “entityType” and contains a string representation of `${Entity/entityType}`.

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

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

• 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	JSON Type	Value Range	Sample Values
Core Types	xs:string	string	Character string	"Hello world", "Καλημέρα κόσμε", "コンニチハ"
	xs:boolean	boolean	true, false	true, false
	xs:decimal	number	Arbitrary-precision decimal numbers	-1.23, 126789672374892739424.543233, +100000.00, 210
	xs:integer	number	Arbitrary-size integer numbers	-1, 0, 126789675432332938792837429837429837429, +100000
IEEE-floating-point numbers	xs:double	number	64-bit floating point numbers incl. ±Inf, ±0, NaN	-1.0, +0.0, -0.0, 234.567e8, -INF, NaN
	xs:float	number	32-bit floating point numbers incl. ±Inf, ±0, NaN	-1.0, +0.0, -0.0, 234.567e8, -INF, NaN
	xs:date	string	Dates (yyyy-mm-dd) with or without timezone	"2000-01-01","2000-01-01Z", "2000-01-01+12:05"

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

Time and data	xs:time	string	Times (hh:mm:ss.sss…) with or without timezone	"14:23:00", "14:23:00.527634Z", "14:23:00+03:00"
	xs:dateTime	string	Date and time with or without timezone	"2000-01-01T14:23:00", "2000-01-01T14:23:00.66372+14:00"
	xs:dateTimeStamp	string	Date and time with required timezone	"2000-01-01T14:23:00.66372+14:00"
Recurring and partial dates	xs:gYear	string	Gregorian calendar year	"2000", "2000+03:00"
	xs:gMonth	string	Gregorian calendar month	"--04", "--04+03:00"
	xs:gDay	string	Gregorian calendar day of the month	"---04", "---04+03:00"
	xs:gYearMonth	string	Gregorian calendar year and month	"2000-01", "2000-01+03:00"
	xs:gMonthDay	string	Gregorian calendar month and day	"--01-01", "--01-01+03:00"
	xs:duration	string	Duration of time	"P30D", "-P1Y2M3DT1H", "PT1H5M0S"
	xs:yearMonthDuration	string	Duration of time (months and years only)	"P10M", 'P5Y2M"
	xs:dayTimeDuration	string	Duration of time (days, hours, minutes, seconds only)	"P30D", 'P1DT5H", 'PT1H5M0S"
Limited-range integer numbers	xs:byte	number	-128…+127 (8 bit)	-1, 0, 127
	xs:short	number	-32768…+32767 (16 bit)	-1, 0, 32767
	xs:int	number	2147483648…+2147483647 (32 bit)	-1, 0, 2147483647
	xs:long	number	-9223372036854775808…+9223372036854775807 (64 bit)	-1, 0, 9223372036854775807
	xs:unsignedByte	number	0…255 (8 bit)	0, 1, 255
	xs:unsignedShort	number	0…65535 (16 bit)	0, 1, 65535
	xs:unsignedInt	number	0…4294967295 (32 bit)	0, 1, 4294967295
	xs:unsignedLong	number	0…18446744073709551615 (64 bit)	0, 1, 18446744073709551615
	xs:positiveInteger	number	Integer numbers >0	1, 7345683746578364857368475638745
	xs:nonNegativeInteger	number	Integer numbers ≥0	0, 1, 7345683746578364857368475638745
	xs:negativeInteger	number	Integer numbers <0	-1, -23487263847628376482736487263847
	xs:nonPositiveInteger	number	Integer numbers ≤0	-1, 0, -93845837498573987498798987394

Encoded binary data	xs:hexBinary	string	Hex-encoded binary data	"6b756d6f77617368657265"
	xs:base64Binary	string	Base64-encoded binary data	"a3Vtb3dhc2hlcmU="
Miscellaneous types	xs:anyURI	string	Absolute or relative URIs and IRIs	"http://customer.com/demo/aas/1/1/1234859590", "urn:example:company:1.0.0"
	rdf:langString	string	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:

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

Note 1: 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 `xs:decimal`, `xs:integer`, `xs:positiveInteger`, `xs:nonNegativeInteger`, `xs:negativeInteger`, `xs:nonPositiveInteger` and the bounded type `xs:unsignedLong`. Other numeric types are not affected.[5]

Note 2: The valueOnly serialization uses JSON native data types, AAS in general uses XML Schema Built-in Datatypes for Simple Data Types and ValueDataType. In case of booleans, JSON accepts only literals true and false, whereas xs:boolean also accepts 1 and 0, respectively. In case of double, JSON number is used in valueOnly, but JSON number does not support INF (= Infinity), which is supported by xs:double. (See https://datatracker.ietf.org/doc/html/rfc8259#section-6)

Examples conformant to [3]:

Full serialization of single submodel element *Property*:

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

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

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

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

For a *SubmodelElementCollection* 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:

```
{
    "MaxRotationSpeedReference":
    {
        "type": "ModelReference",
        "keys": [
            {
                "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":
    {
        "type": "GlobalReference",
        "keys": [
            {
                "type": "GlobalReference",
                "value": "0173-1#02-BAA120#008"
            }
        ]
    }
}
```

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

```
{
    "Document": {
        "contentType": "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": {
        "contentType": "application/octet-stream"
    }
}
```

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

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

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

```
{
    "CurrentFlowsFrom": {
        "first": {
            "modelType": "ModelReference",
            "keys": [
                {
                    "type": "Submodel",
                    "value": "http://customer.com/demo/aas/1/1/1234859590"
                },
                {
                    "type": "Property",
                    "value": "PlusPole"
                }
            ]
        },
        "second": {
            "modelType": "ModelReference",
            "keys": [
                {
                    "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": {
            "modelType": "ModelReference",
            "keys": [
                {
                    "type": "Submodel",
                    "value": "http://customer.com/demo/aas/1/1/1234859590"
                },
                {
                    "type": "Property",
                    "value": "PlusPole"
                }
            ]
        },
        "second": {
            "modelType": "ModelReference",
            "keys": [
                {
                    "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": {
            "modelType": "GlobalReference",
            "keys": [
                {
                     "type": "GlobalReference",
                     "value": "http://customer.com/demo/asset/1/1/MySubAsset"
                }
            ]
        }
    }
}
```

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

```
{
    "MyBasicEvent": {
        "observed": {
            "modelType": "ModelReference",
            "keys": [
                {
                    "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 *SubmodelElementCollections*. Since *SubmodelElementCollections* 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 than *SubmodelElementCollections*. Hence, for each *SubmodelElementCollection* 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. However, the *SubmodelElementCollection* is added to the following schema for completeness and clarity, but it is not referenced from the *SubmodelElementValue*-oneOf-Enumeration due to the reasons mentioned above.
See Annex B for an example that validates against this schema.

```
{
  "$schema": "https://json-schema.org/draft/2019-09/schema",
  "title": "ValueOnly-Serialization-Schema",
  "$id": "http://www.admin-shell.io/schema/valueonly/json/V1.0RC03",
  "definitions": {
    "PropertyValue": {
      "oneOf": [
        {
          "$ref": "#/definitions/StringValue"
        },
        {
          "$ref": "#/definitions/NumberValue"
        },
        {
          "$ref": "#/definitions/BooleanValue"
        }
      ]
    },
    "MultiLanguagePropertyValue": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/LangString"
      },
      "additionalProperties": false
    },
    "LangString": {
      "type": "object",
      "patternProperties": {
        "^[a-z]{2,4}(-[A-Z][a-z]{3})?(-([A-Z]{2}|[0-9]{3}))?$": {
          "type": "string"
        }
      },
      "additionalProperties": false
    },
    "RangeValue": {
      "type": "object",
      "properties": {
        "min": {
          "type": "number"
        },
        "max": {
          "type": "number"
        }
      },
      "required": [
        "min",
        "max"
      ],
      "additionalProperties": false
    },
    "FileBlobValue": {
      "type": "object",
      "properties": {
        "contentType": {
          "type": "string"
        },
        "value": {
          "type": "string"
        }
      },
      "required": [
        "contentType",
        "value"
      ],
      "additionalProperties": false
    },
    "ReferenceElementValue": {
      "$ref": "#/definitions/ReferenceValue"
    },
    "ReferenceValue": {
      "type": "object",
      "properties": {
        "type": {
          "type": "string",
          "enum": ["ModelReference", "GlobalReference"]
        },
        "keys": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/Key"
          }
        }
      },
      "additionalProperties": false
    },
    "Identifier": {
      "type": "string"
    },
    "BasicEventElementValue": {
      "type": "object",
      "properties": {
        "observed": {
          "$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": {
            "$ref": "#/definitions/SpecificAssetIdValue"
          }
        }
      },
      "required": [
        "statements",
        "entityType"
      ],
      "additionalProperties": false
    },
    "SpecificAssetIdValue": {
      "type": "object",
      "patternProperties": {
        "(.*?)": {
          "type": "string"
        }
      }
    },
    "RelationshipElementValue": {
      "type": "object",
      "properties": {
        "first": {
          "$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
    },
    "SubmodelElementCollectionValue": {
      "$ref": "#/definitions/ValueOnly"
    },
    "SubmodelElementListValue": {
      "type": "array",
      "items": {
        "$ref": "#/definitions/SubmodelElementValue"
      }
    },
    "SubmodelElementValue": {
      "oneOf": [
        {
          "$ref": "#/definitions/BasicEventElementValue"
        },
        {
          "$ref": "#/definitions/RangeValue"
        },
        {
            "$ref": "#/definitions/MultiLanguagePropertyValue"
          },
          {
            "$ref": "#/definitions/FileBlobValue"
          },
          {
            "$ref": "#/definitions/ReferenceElementValue"
          },
          {
            "$ref": "#/definitions/RelationshipElementValue"
          },
          {
            "$ref": "#/definitions/AnnotatedRelationshipElementValue"
          },
          {
            "$ref": "#/definitions/EntityValue"
          },
          {
            "$ref": "#/definitions/PropertyValue"
          },
          {
            "$ref": "#/definitions/SubmodelElementListValue"
          }
        ]
      },
      "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 SubmodelElementCollection 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 *MySubmodelElementCollection* with OutputModifier level = deep, the list of idShort paths is returned as follows:

Submodel: MySubmodel

- ⇨ Property: MyTopLevelProperty
- ⇨ SMC: MySubmodelElementCollection
    - ○ Property: MySubProperty1
    - ○ Property: MySubProperty2
    - ○ SMC: MySubSubmodelElementCollection
        - ▪ Property: MySubSubProperty1
        - ▪ Property: MySubSubProperty2
    - ○ SML: MySubSubmodelElementList
        - ▪ Property: "MySubTestValue1",
        - ▪ Property: "MySubTestValue2",

```
[
    "MySubmodelElementCollection",
    "MySubmodelElementCollection.MySubProperty1",
    "MySubmodelElementCollection.MySubProperty2",
    "MySubmodelElementCollection.MySubSubmodelElementCollection",
    "MySubmodelElementCollection.MySubSubmodelElementCollection.MySubSubProp-
erty1",
    "MySubmodelElementCollection.MySubSubmodelElementCollection.MySubSubProp-
erty2",
    "MySubmodelElementCollection.MySubSubmodelElementList[0]",
    "MySubmodelElementCollection.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.

To clearly separate the different parts of the AAS model, the model has been split into several HTTP/REST APIs.

The schema for the metamodel of part 1 is available at: https://app.swaggerhub.com/domains/Plattform_i40/Part1-MetaModel-Schemas/V3.0RC02# This schema includes general objects which are used in the further defined APIs.

Additional objects are needed for part 2, e.g. for the value only serialization or the descriptors for the registry. The related schema of part 2 objects for all APIs is vailable at: https://app.swaggerhub.com/domains/Plattform_i40/Part2-API-Schemas/V1.0RC03# This schema includes general objects which are used in the further defined APIs.

AAS uses data specifications from IEC 61360. The schema for these data specification templates is available at: https://app.swaggerhub.com/domains/Plattform_i40/IEC61360-Schemas/V3.0RC02# This schema includes general objects which are used in the further defined APIs.

The definition on endpoints ist based on the DIN SPEC 16593. The related schema for DIN SPEC 16593 is available at: https://app.swaggerhub.com/domains/Plattform_i40/DINSPEC16593-Schemas/V1.0RC03# This schema includes general objects which are used in the further defined APIs. Based on these objects above the part 2 APIs are defined.

All individual APIs, collected in one document, are available at: https://app.swaggerhub.com/domains/Plattform_i40/Part2-API-Schemas/V1.0RC03# This document is just a list of single separate APIs and not a comprehensive Service Specification of the Industrie 4.0 Service Model as introduced in chapter 3.1. Several APIs can be combined and nested in so called “superpaths” (see 10.2 below).

The AAS API with Submodel API and Serialization APIs included is available at: https://app.swaggerhub.com/apis/Plattform_i40/AssetAdministrationShell-API/V1.0RC03 This is a combination of APIs which forms a Service Specification according to the Industrie 4.0 Service Model in chapter 3.1.

The AAS Repository API with AAS API, Submodel API, Submodel Repository API, Concept Description Repository API and Serialization APIs included is available at: https://app.swaggerhub.com/apis/Plattform_i40/AssetAdministrationShell-Environment/V1.0RC03 This is a combination of APIs which forms a Service Specification according to the Industrie 4.0 Service Model in chapter 3.1.

Registry and discovery are independent from the other APIs. In the future, registry and discovery will be implemented at endpoints, but to actually simplify the implementation both have been combined. An AAS Registry with an AAS Discovery API included is available at: https://app.swaggerhub.com/apis/Plattform_i40/Registry-and-Discovery/V1.0RC03 This is a combination of APIs which forms a Service Specification according to the Industrie 4.0 Service Model in chapter 3.1.

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

The swagger APIs above are just examples of how different APIs can be combined to Service Specifications. Further combinations of APIs may form further Service Specifications. A Service Specification is not the same as a profile. Profiles will e.g. define which APIs are mandatory or optional in a Service Specification. Profiles will be defined in a future version of this specification.

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.
- An endpoint of the HTTP/REST API shall always use HTTPS (Port 443) with an up-to-date level of encryption.
- 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=metadata". In these metadata objects selected elements are left off in the payload. Please see clause 10.5.
- 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.
- Notice that BASE-64-URL-encoding is slightly different to BASE-64-encoding and has been specifically defined for passing URLs. An appropriate BASE-64-URL implementation needs to be used for encoding/decoding. See RFC 4648 for further details.
- When BASE64-URL or BASE64 encoding is mentioned in connection with string values (e.g., Identifiers), the UTF-8 decoded byte array representation of that string is used for the BASE64-URL or BASE64 encoding.
- 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 SpecificAssetId s, 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=W3sia2V5IjogImdsb2JhbEFzc2V0SWQiLCJ2YWx1ZSI6ICJodHRwOi8vZXhhbXBsZS5jb

21wYW55L215QXNzZXQifSx7ImtleSI6ICJteU93bkludGVybmFsQXNzZXRJZCIsInZhbHVlIjogIjEyMzQ1QUJDIn1d”.
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.
- The recursive nature of the Reference class (Reference/referredSemanticId points to Reference again) can not be represented in SwaggerHub due to a bug in the SwaggerUI code. Therefore, the additional class “ReferenceParent" has been added. “ReferenceParent" shall not be used in productive operations and is only a placeholder for “Reference”. When implementing generated code originating from the SwaggerHub schemas, please delete “ReferenceParent” and add its attributes to “Reference”.

10.3 API Versioning

API versioning provides a way to deal with different versions of the same API at the same time. This way older versions may still be accessible on the same server to provide services to legacy clients without breaking existing funcationality.

There are different solutions regarding API versioning involving URL-based versioning, query parameter-based versioning as well as HTTP header-oriented solutions using custom or standard headers.
As different solutions also provide different advantages and disadvantages, **URL-based versioning** has been selected as the most suitiable method for the AAS API. Among other advantages implementation complexity on clients as well as servers is rather low and different versions can be easily accessed through browsers without the need for specific development tools or extensions.

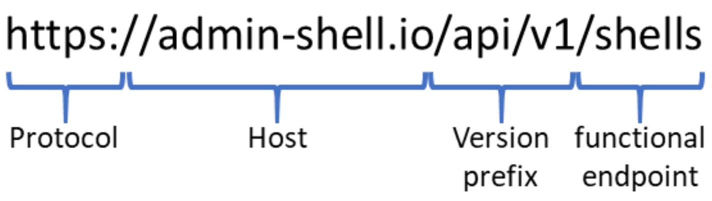


Figure 4 - Generic URL Scheme for AAS API versioning

Upcoming implementations of AAS related servers need to implement the version prefix “**api/v<X>/**” to provide information of the specific major version regarding AAS Part 2 version, where <X> denotes the implemented version, e.g. “api/v1/”.

Note: All URLs mentioned in this document regarding the REST mapping of the AAS APIs have to be understood with this prefix in mind.

The versioning scheme for AAS API related services follows semantic versioning[6]. Very briefly this defines version numbers as a format following: <MAJOR>.<MINOR>.<PATCH>.

The major version changes if there are breaking or incompatible changes which need to be addressed by clients. Minor versions add (new) functionality in a backwards compatible way and allow clients with lower minor versions to keep their existing functionality. Patch versions only include backwards compatible bug fixes.

AAS api versioning mainly only use the major version as described above, sind minor and patch define upwards compatible versions.

Additionally, “Release candidates” are variants of the implementation of the denoted major version. For example, “1.0.0 RC2” should be interpreted as the second (alternative) release candidate for version 1.0.0. This will still result in the version prefix “/api/v1/”.

As multiple versions will be supported in the future, an AAS ecosystem consisting of Registry / Discovery service as well as AAS Repository, Submodel (standalone), AAS (standalone) interfaces should share a consistent version. Therefore, it is intended to provide a consistent interface description as OpenAPI package with each such major version.

Upcoming compatibility constraints regarding newer versions will be elaborated in further iterations of this document and related technical descriptions (OpenAPI specification).

Lastly to further denote information about APIs / servers capabilities it is intended to include into each service an additional "profile” endpoint. This endpoint will provide information about the detailed API version (e.g. minor and patch version) as well the used meta-model version and additional capability information (e.g. pagination).

10.4 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.

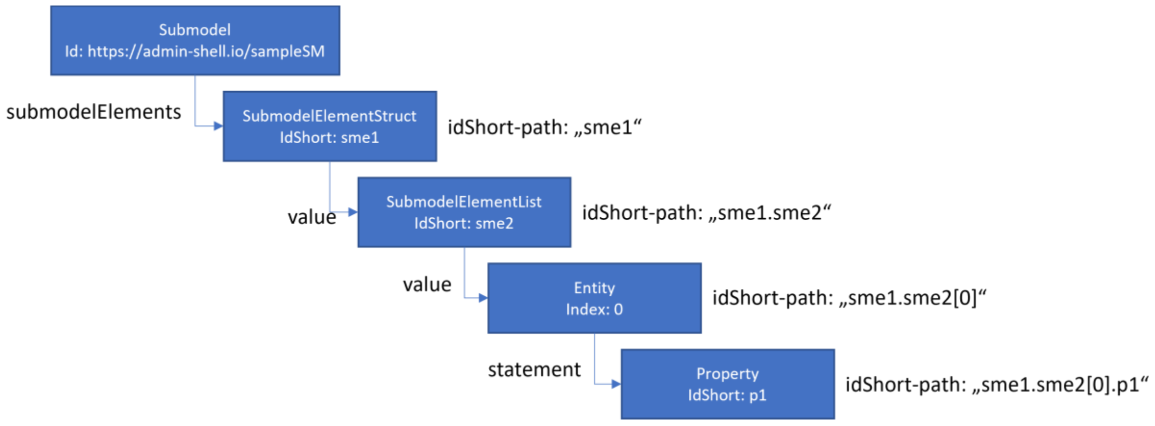


Figure 5 example hierarchy

[6] http://semver.org

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, SubmodelElementCollection, 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
SubmodelElementCollection	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]”.

10.5 Metadata Objects

Metadata objects are defined for scenarios where a client only wants to access the metadata of an object but not the value. **Metadata objects are only part of HTTP/REST and do not change the metamodel.** Metadata 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 metadata object does not contain any additional fields in relation to its full object representation, only some fields are left offThe 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 metadata object is the same as for the original full object, but without the left off fields.

Class Name	Fields not available in metadata representation
Identifiables	
AssetAdministrationShell	assetInformation, submodels

Submodel	submodelElements
SubmodelElements	
SubmodelElementCollection	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, contentType
File	value, contentType

Example

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

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

Table 1 AssetAdministrationShell JSON serialization example

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

     …

    "derivedFrom": {…}
    "assetInformation": {…},
    "submodels": […]
  }
```

Table 2 AssetAdministrationShell metadata JSON serialization example

```
→ {
→         "idShort": "TestAssetAdministrationShell",
→         "description": […],
→         "id": {…}
→
→         …
→
→         "derivedFrom": {…}
→         }
→     }
```

10.6 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 10.9.

10.7 Modifiers

To use metadata objects as described in section 10.5. Modifiers are implemented as HTTP Query parameters. For example a request for a specific submodel may look like:
GET /submodel?level=deep&content=value&extent=withBlobValue

In combination with the level modifier the following rules apply:

- If Level=Core and Content=Value, then only the requested object and the children without their value (empty value) will be returned in value serialization.

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.

Modifiers can not be used for POST operations.

In general, the combination of Level=Deep and Content=Reference is not allowed. If a client application sends an invalid combination of modifiers, the server must respond with the appropriate error code (405 Method not allowed).

10.8 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/metadata/reference
PutAssetAdministrationShell	PUT	/aas	?content=normal/metadata
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/metadata/value/reference/path ?extent=WithoutBLOBValue/WithBLOBValue
PutSubmodel	PUT	/submodel	?level=deep/core ?content=normal/metadata/value ?extent=WithoutBLOBValue/WithBLOBValue
GetAllSubmodelElements	GET	/submodel/submodel-elements	?level=deep/core ?content= normal/metadata/value/reference/path ?extent=WithoutBLOBValue/WithBLOBValue
GetSubmodelElementByPath	GET	/submodel/ submodel-elements/{ idShortPath}	use seperated idshort path of this element ?level=deep/core ?content= normal/metadata/value/reference/path ?extent=WithoutBLOBValue/WithBLOBValue URL-encoded IdShortPath
GetFileByPath	GET	/submodel/ submodel-elements/{ idShortPath}/attachment	use seperated idshort path of this element URL-encoded IdShortPath
PostSubmodelElement	POST	/submodel/submodel-elements	Output modifiers are not used with POST
PostSubmodelElementByPath	POST	/submodel/submodel-elements/{idShortPath}	use seperated idshort path of the parent element Output modifiers are not used with POST
PutSubmodelElementByPath	PUT	/submodel/ submodel-elements/{idShortPath}	use seperated idshort path of this element ?level=deep/core ?content=normal/metadata/value ?extent=WithoutBLOBValue/WithBLOBValue URL-encoded IdShortPath
PutFileByPath	PUT	/submodel/ submodel-elements/{ idShortPath}/attachment	use seperated idshort path of this element 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	/shell-descriptors	
GetAssetAdministrationShellDescriptorById	GET	/shell-descriptors/{aasIdentifier}	BASE64-URL-encoded identifier
PostAssetAdministrationShellDescriptorById	POST	/shell-descriptors/{aasIdentifier}	BASE64-URL-encoded identifier
PutAssetAdministrationShellDescriptorById	PUT	/shell-descriptors/{aasIdentifier}	BASE64-URL-encoded identifier

DeleteAssetAdministrationShellDescriptorById	DELETE	/shell-descriptors/{aasIdentifier}	BASE64-URL-encoded identifier
Submodel Registry Interface	*	/shell-descriptors/{aasIdentifier}/submodelDescriptors/*	recommended: Submodel Registry Interface for SuperPath
Submodel Registry Interface			
GetAllSubmodelDescriptors	GET	/submodel-descriptors	
GetSubmodelDescriptorById	GET	/submodel-descriptors/{submodelIdentifier}	BASE64-URL-encoded identifier
PostSubmodelDescriptor	POST	/submodel-descriptors/{submodelIdentifier}	BASE64-URL-encoded identifier
PutSubmodelDescriptorById	PUT	/submodel-descriptors/{submodelIdentifier}	BASE64-URL-encoded identifier
DeleteSubmodelDescriptorById	DELETE	/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

10.9 Mapping of Status Codes

The following table shows the mapping of the generic status codes to HTTP status codes according to IETF RFC 7231 (see chapter 6.1 https://datatracker.ietf.org/doc/html/rfc7231#section-6)

Generic Status Code	Meaning	HTTP status code	Explanation
Success	Success	200 (OK)	Standard response for successful requests
SuccessCreated	Creation of a new resource successful	201 (Created)	Successful request resulting in the creation of a new resource, e.g., SubmodelElement
SuccessNoContent	Success with explicitly no content in the payload	204 (No Content)	Successful request with no content in return, e.g., used for updating existing resources
ClientForbidden	Request is unauthorized	403 (Forbidden)	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 (Bad Request)	The server does not / cannot process the request due to a

			general client error, e.g., malformed request
ClientErrorResourceNotFound	Resource not found	404 (Not Found)	The requested resource was not found
ClientMethodNotAllowed	Operation request is not allowed	405 (Method Not Allowed)	The server rejected the request for the requested resource, e.g., /invoke only for Operation submodel element
ServerInternalError	Unexpected error	500 (Internal Server Error)	General server internal error due to an unexpected condition
ServerNotImplemented	Not implemented	501 (Not Implemented)	The server does not support the functionality to fulfill the request
ServerErrorBadGateway	Bad Gateway	502 (Bad Gateway)	The primarily addressed server that was acting as gateway or proxy received an invalid response from subsequent systems/servers.

10.10 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.

10.10.1 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.
packageId*	File server specific package id	string	attr	1
aasId	Asset Administration Shell unique identifier	Identifier	attr	0..*

10.11 Interactions

Interactions describe the sequence of calls of operations by a client application to achieve a defined goal in a use case. 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.

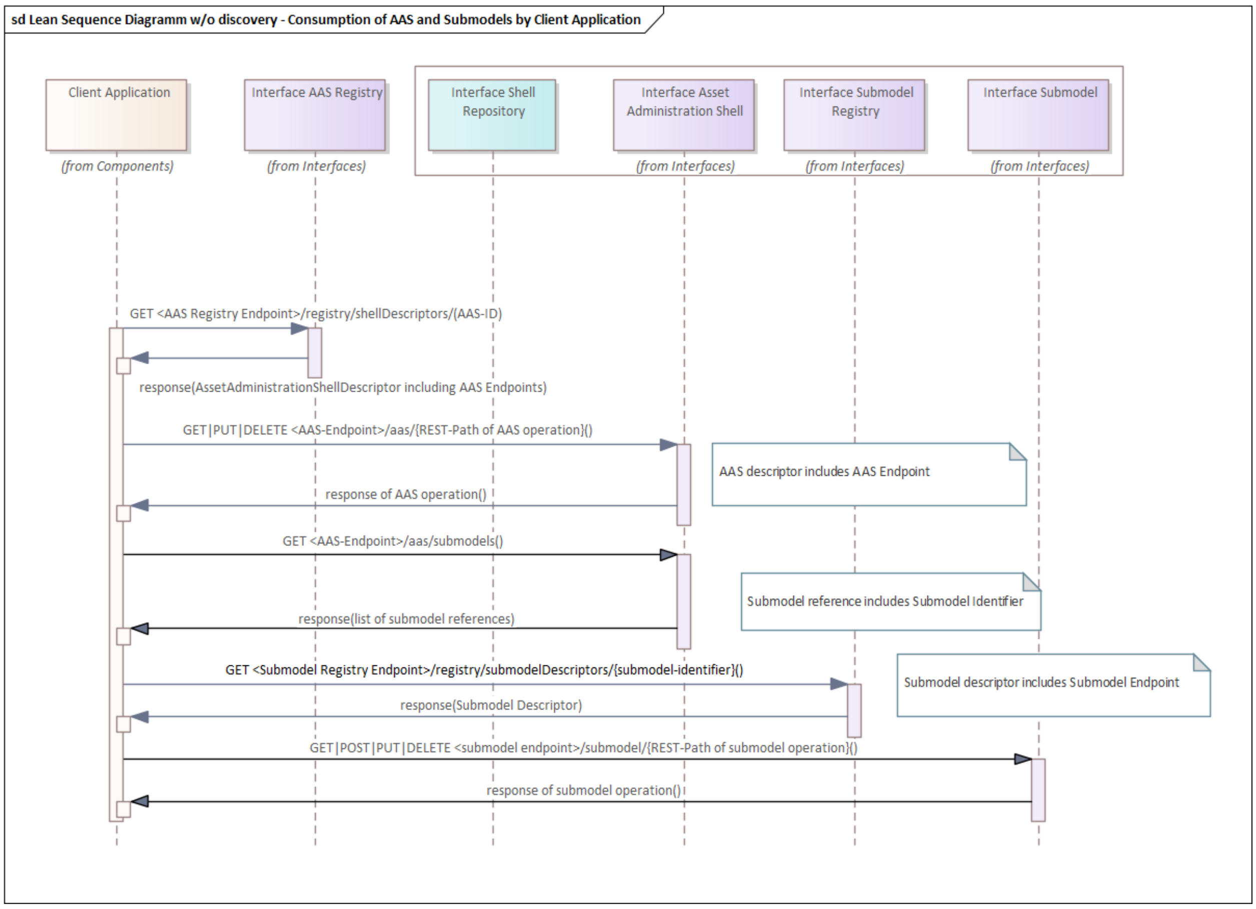
As the interaction diagram in the current version just describes a first subset of interactions, some constraints and assumptions are made according to the configuration and qualities of the system. In future versions of the document further interactions will be described (mentioned below as “to be created”),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

Figure 6 Interaction for client application using AAS and Submodels

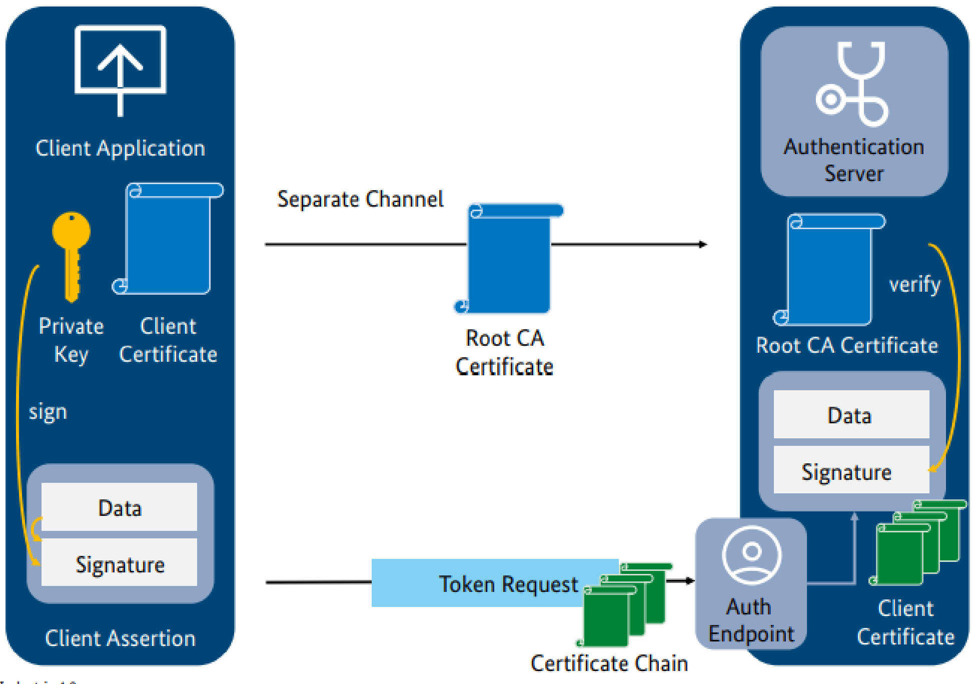

sd Lean Sequence Diagramm w/o discovery - Consumption of AAS and Submodels by Client Application
Client Application
(from Components)
Interface AAS Registry
(from Interfaces)
Interface Shell Repository
Interface Asset Administration Shell
(from Interfaces)
Interface Submodel Registry
(from Interfaces)
Interface Submodel
(from Interfaces)
GET <AAS Registry Endpoint>/registry/shellDescriptors/{AAS-ID}
response(AssetAdministrationShellDescriptor including AAS Endpoints)
GET|PUT|DELETE <AAS-Endpoint>/aas/{REST-Path of AAS operation}()
AAS descriptor includes AAS Endpoint
response of AAS operation()
GET <AAS-Endpoint>/aas/submodels()
Submodel reference includes Submodel Identifier
response(list of submodel references)
GET <Submodel Registry Endpoint>/registry/submodelDescriptors/{submodel-identifier}()
Submodel descriptor includes Submodel Endpoint
response(Submodel Descriptor)
GET|POST|PUT|DELETE <submodel endpoint>/submodel/{REST-Path of submodel operation}()
response of submodel operation()

10.12 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=Decentralized Identifiers, Verifiable Credentials or IDS=International Data Spaces) 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 7 gives an overview.

Figure 7 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 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.

12 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 • … • 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.

Operation Name:	Name of the Operation: All individual words in the operation name are capitalized	
	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: All words in the attribute name are written together in italics without a blank in between.	Type of the attribute N	Kind of attribute N 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	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.

	The first letter of the first word is lower case, all other words upper case.		not exist independent of its parent) • ref*: shared aggregation (does exist independent of its parent)	

1307

1308 **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. 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": {
    "contentType": "application/pdf",
    "value": "SafetyInstructions.pdf"
  },
  "MyBlob": {
    "contentType": "application/octet-stream",
    "value": "VGhpcyBpcyBteSBibG9i"
  },
  "MyEntity": {
    "statements": {
      "MaxRotationSpeed": 5000
    },
    "entityType": "SelfManagedEntity",
    "globalAssetId": {
      "type": "GlobalReference",
      "keys": [
        {
          "type": "GlobalReference",
          "value": "http://customer.com/demo/asset/1/1/MySubAsset"
        }
      ]
    }
  },
  "MyReference": {
    "type": "ModelReference",
    "keys": [
      {
        "type": "Submodel",
        "value": "http://customer.com/demo/aas/1/1/1234859590"
      },
      {
        "type": "Property",
        "value": "MaxRotationSpeed"
      }
    ]
  },
  "MyBasicEvent": {
    "observed": {
      "type": "ModelReference",
      "keys": [
        {
          "type": "Submodel",
          "value": "http://customer.com/demo/aas/1/1/1234859590"
        },
        {
          "type": "Property",
          "value": "CurrentValue"
        }
      ]
    }
  },
  "MyRelationship": {
    "first": {
      "type": "ModelReference",
      "keys": [
        {
          "type": "Submodel",
          "value": "http://customer.com/demo/aas/1/1/1234859590"
        },
        {
          "type": "Property",
          "value": "PlusPole"
        }
      ]
    },
    "second": {
      "type": "ModelReference",
      "keys": [
        {
          "type": "Submodel",
          "value": "http://customer.com/demo/aas/1/0/1234859123490"
        },
        {
          "type": "Property",
          "value": "MinusPole"
        }
      ]
    }
  },
  "MyAnnotatedRelationship": {
    "first": {
      "type": "ModelReference",
      "keys": [
        {
          "type": "Submodel",
          "value": "http://customer.com/demo/aas/1/1/1234859590"
        },
        {
          "type": "Property",
          "value": "PlusPole"
        }
      ]
    },
    "second": {
      "type": "ModelReference",
      "keys": [
        {
          "type": "Submodel",
          "value": "http://customer.com/demo/aas/1/0/1234859123490"
        },
        {
          "type": "Property",
          "value": "MinusPole"
        }
      ]
    },
    "annotation": [
      {
        "AppliedRule": "TechnicalCurrentFlowDirection"
      }
    ]
  },
  "MySubmodelElementIntegerPropertyList": [
    1,
    2,
    30,
    50
  ],
  "MySubmodelElementFileList": [
    {
      "contentType": "application/pdf",
      "value": "MyFirstFile.pdf"
    },
    {
      "contentType": "application/pdf",
      "value": "MySecondFile.pdf"
    }
  ]
}
```

Annex C. Bibliography

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

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

Change Notes

1. General

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

1. Interface Changes w.r.t. V1.0RC02 to V1.0RC03

BWC	Interface Change	Kind of Change	Comment
*	Discovery	Change	IndentifierKeyValuePair to SpecificAssetId
*	Submodel	Change	SubmodelElementStruct remains as SubmodelElementCollection
*	Submodel	Change	ModelReference and GlobalReference are combined back to Reference
*	Submodel	Change	Rename trimmed to metadata
	Submodel	New	Add GetFileByPath
	Submodel	New	Add PutFileByPath
*	Submodel	Change	InvokeOperationAsync
	Registry	Update	Endpoint
*	Registry	Change	Remove /registry from REST path
*	All	New	API Versioning adds a prefix to all interfaces

2. Operation Changes w.r.t. V1.0RC02 to V1.0RC03

Operation Change Old	Operation Change New	Kind of Change	Comment
InvokeOperationAsync		Change	inputArgument and inoutputArgument are OperationVariable
GetAllAssetAdministrationShellIdsByAssetLink		Change	IndentifierKeyValue Pair to SpecificAssetId
GetAllAssetLinksById		Change	IndentifierKeyValue Pair to SpecificAssetId
PostAllAssetLinksById		Change	IndentifierKeyValue Pair to SpecificAssetId

3. Interface Changes w.r.t. V1.0RC01 to V1.0RC02

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

4. Operation Changes w.r.t. V1.0RC01 to V1.0RC02

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 resource 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	

1353