FA^{3}ST Service

Fraunhofer IOSB

Apr 23, 2024

BASICS

The **F**raunhofer **A**dvanced **A**sset **A**dministration **S**hell **T**ools (**FA³ST**) Service enables creation of Digital Twins (DTs) in accordance to the Asset Administration Shell (AAS) specification . FA³ST Service is a software that, when started, offers one or more AAS-compliant APIs to interact with a DT. Optionally, it can be started given an existing AAS model file and/or a configuration file. FA³ST Service also allows synchronizing a DT with the asset(s) it represent via so-called *AssetConnection*.

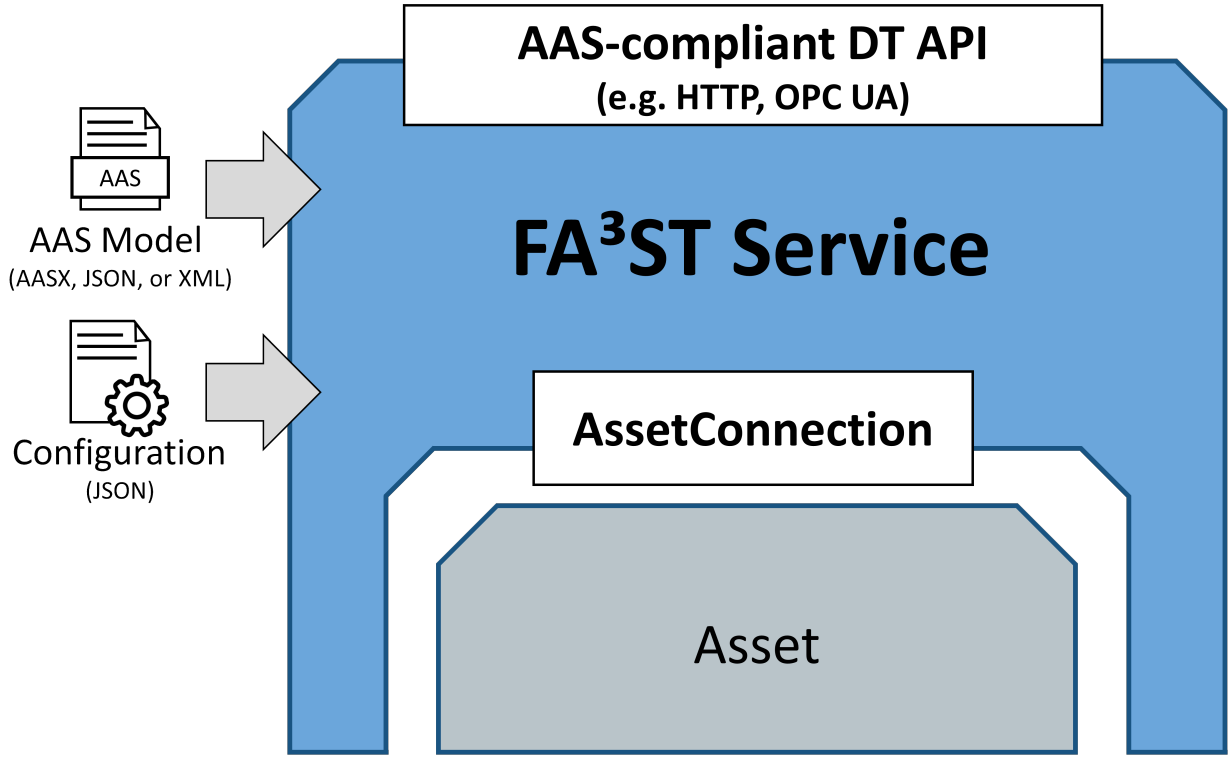

Fig. 1: FA³ST Service: Non-Technical View.

CHAPTER

ONE

IMPLEMENTED AAS SPECIFICATIONS

- Details of the Asset Administration Shell - Part 1: Metamodel v3.0 (specification)
- Details of the Asset Administration Shell - Part 2: Application Programming Interfaces v3.0.1 (specification) (OpenAPI)

CHAPTER

TWO

FEATURES

- Easy to use even for non-developers via command-line interface (CLI), docker container, or embedded library
- Configuration via a single JSON file
- Open Architecture: easily extendable and configurable
- Asset Synchronization: synchronize your assets and DTs using arbitrary communication protocols
- Allows accessing your DT using multiple endpoints at the same time, e.g., HTTPS and OPC UA
- Uses existing open source implementation of AAS datamodel and de-/serializers Eclipse AAS4J
- Supports several dataformats for the Asset Administration Shell Environment: `AASX, JSON, XML`

Caution: At the moment there is no security specification available for the AAS. Therefore FA^{3}ST does not implement any security mechanisms. They will be implemented as soon as a security specification is available. We strongly recommend to be careful when using external AAS models or submodels.

2.1 Getting Started

FA^{3}ST Service uses the concept of an open architecture. This means, it is designed to be easily extenadable and customizable.

The main components of FA^{3}ST Service are `AAS Model`, which basically is a representation of the meta model classes of the AAS such as *Asset Administration Shell*, *Submodel*, *SubmodelElement*, or *Property*, and `Core`, which implements all the processing logic. Besides those two central components, FA^{3}ST Service offers multiple interfaces that each can have different and/or custom implementations. FA^{3}ST Service already ships with a number of so-called *default implementations* of these interfaces depicted by the light-grey boxes to the left and right in the figure.

The interfaces provide the following functionalities:

- `Endpoint`: Communication with the DT from the outside
- `MessageBus`: Communication & synchronization between FA^{3}ST Service components
- `De-/Serializer`: De-/Serialization of AAS models in from/to data formats
- `Persistence`: Persistent storage of data (model + values)
- `FileStorage`: Peristent storage of complementary files (e.g. PDF files linked from the AAS)
- `AssetConnection`: Synchronization with underlying asset(s)

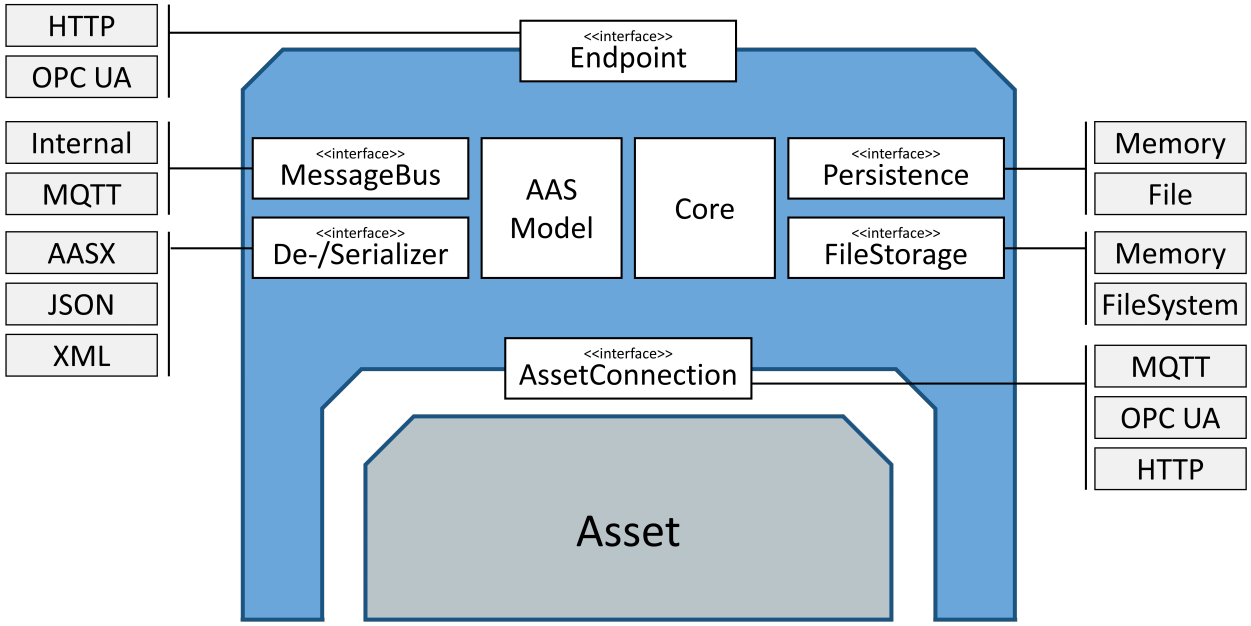

Fig. 1: High-Level Architecture of FA³ST Service.

2.2 Installation

2.2.1 Requirements

- Java Runtime 17 or newer

2.2.2 Precompiled JAR

`Latest RELEASE version (1.0.1)`

`Latest SNAPSHOT version (1.1.0-SNAPSHOT)`

2.2.3 Maven Dependency

```
<dependency>
        <groupId>de.fraunhofer.iosb.ilt.faaast.service</groupId>
        <artifactId>starter</artifactId>
        <version>1.0.1</version>
</dependency>
```

2.2.4 Gradle Dependency

```
implementation 'de.fraunhofer.iosb.ilt.faaast.service:starter:1.0.1'
```

2.2.5 Build from Source

```
git clone https://github.com/FraunhoferIOSB/FAAAST-Service
cd FAAAST-Service
mvn clean install
```

2.3 Usage

2.3.1 Command-Line Interface (CLI)

To start FA³ST Service from command-line you need to run the `starter` module by calling

```
> java -jar starter-{version}.jar
```

When started without arguments, FA³ST Service will try to auto-detect a configuration file named `config.json` and a model file named `model.[ext]` where `[ext]` is a supported file extension like `json`, `xml`, or `aasx`.

To manually pass a model file `my-model.aasx` and a configuration file `my-config.json` run the following command:

```
> java -jar starter-{version}.jar --model my-model.aasx --config my-config.json
```

Table 1: Supported CLI arguments and environment variables.

CLI (shor	CLI (long)	Environment variable	AllowedValues	Description	Default-Value
-c	–config	faaast_config		The config file to use.	config.json
-e	–empty-model			Starts the FAST service with an empty Asset Administration Shell Environment.	
	–endpoint		HTTPOPCUA	Additional endpoints that should be started.	
-h	–help			Print help message and exit.	
	–loglev external	faaast_loglevel_exter	TRACEDEBUGINFOWARNERROR	Sets the log level for external packages.This overrides the log level defined by other commands such as *-q* or *-v*.	WARN
	–loglev faaast	faaast_loglevel_faaas	TRACEDEBUGINFOWARNERROR	Sets the log level for FA³ST packages.This overrides the log level defined by other commands such as *-q* or *-v*.	WARN
-m	–model			The model file to load.	model.*
	–no-validati	faaast_no_validation		Disables all validation, overrides validation defined in the configuration Environment.	
-q	–quite			Reduces log output (*ERROR* for FAST packages, *ERROR* for all other packages).Default information about the starting process will still be printed.	
-v	–verbose			Enables verbose logging (*INFO* for FAST packages, *WARN* for all other packages).	
-V	–version			Print version information and exit.	
-vv				Enables very verbose logging (*DEBUG* for FAST packages, *INFO* for all other packages).	
-vvv				Enables very very verbose logging (*TRACE* for FAST packages, *DEBUG* for all other packages).	
	{key}=	faaast_config_extensi *{key}* separated by _	any	Additional properties to override values of configuration using JSONPath notation without starting *$*.	

2.3.2 Docker

FA³ST Service is available on DockerHub with multiple tags

- `latest`: The latests released version, equals to the latests tag `major.minor.bugfix`
- `major.minor.0-SNAPSHOT`: Snapshot build of the current code on the `main` branch of FA³ST Service. This includes all upcoming features not yet relased.
- `major.minor.bugfix`: This tag is available for each officially released version of FA³ST. It is stable, i.e., no updates or bugfixes will ever be applied.
- `major.minor`: This tag is available for each minor release of FA³ST Service and will be updated with bugfixes over time. It is therefore recommended to use these tags over the `major.minor.bugfix` ones.

To run FA³ST Service via docker with an empty model and default configuration execute

```
> docker run fraunhoferiosb/faaast-service
```

To make you of the full power of docker and FA³ST Service, you can also mount files to the container and pass arguments via CLI or environment variables like this

```
> docker run -v {path to your model file}:/model.json -e faaast.model=model.json␣
↪fraunhoferiosb/faaast-service '--no-validation'
```

FA³ST Service also comes with a docker compose file located at `/misc/docker/docker-compose.yml` which can be executed by navigation to the directory `/misc/docker` and execute `docker-compose up`.

2.3.3 From Java Code

You can run FA³ST Service directly from your Java code as embedded library. This way, you can create your configuration and model directly in code and don't have to create them as files (you can still load them from files if you want to). The following code snippet shows how to create and run a new FA³ST Service from code using a model file.

Listing 1: Create a FA³ST Service from code.

```
Service service = new Service(ServiceConfig.builder()
        .core(CoreConfig.builder()
                .requestHandlerThreadPoolSize(2)
                .build())
        .persistence(PersistenceInMemoryConfig.builder()
                .initialModelFile(new File("{pathTo}\\FAAAST-Service\\misc\\examples\\
↪model.aasx"))
                .build())
        .endpoint(HttpEndpointConfig.builder().build())
        .messageBus(MessageBusInternalConfig.builder().build())
        .fileStorage(FileStorageInMemoryConfig.builder().build())
        .build());
service.start();
```

2.4 Configuration

Configuration in FA³ST Service happens primamirly via a single JSON file. However, it is also possible to override configuration properties through command-line arguments and environment variables, where command-line arguments have precedence over environment variables while both override properties defined in the configuration file.

The configuration file contains a `core` section as well as multiple sections telling FA³ST Service which implementations to use for the available interfaces and how to configure them.

Listing 2: Structure of the configuration file.

```
{
        "core" : { },              // core configuration not related to interfaces
        "endpoints" : [ ],         // [0..*] default: HTTP
        "persistence" : { },       // [0..1] default: in-memory
        "fileStorage" : {},        // [0..1] default: in-memory
        "messageBus" : { },        // [0..1] default: internal
```

(continues on next page)

(continued from previous page)

```
        "assetConnections": [ ]    // [0..*] default: none
}
```

A configuration base is always based on the default configuration, meaning that it only needs to contain properties that differ from the default configuration. For example, providing only the `core` section is a valid configuration and will contain default values for all other sections. This is a common scenario if you want to quickly setup FA³ST Service for your first experiments.

Listing 3: Configuration file with only `core` section. All other sections will use default values.

```
{
        "core" : {
                // custom core settings
        }
}
```

2.4.1 Core Configuration

The `core` configuration block contains properties not related to the implementation of any interface.

Table 2: Configuration properties of `core` configuration section.

Name	Allowed Values	Description	Default Value
requestHandlerThreadPoolSize*(optional)*	Integer	Number of concurrent thread that can execute API requests	2
assetConnectionRetryInterval*(optional)*	Long	Interval in ms in which to retry establishing asset connections	1000
validationOnLoad*(optional)*	Object	Validation rules to use when loading the AAS model at startup	all enabled
validationOnCreate*(optional)*	Object	Validation rules to use when creating new elements via API	constraints validation disabled
validationOnUpdate*(optional)*	Object	Validation rules to use when updating elements via API	constraints validation disabled

Listing 4: Example `core` configuration

```
{
        "core" : {
                "requestHandlerThreadPoolSize": 2,
                "assetConnectionRetryInterval": 1000,
                "validationOnLoad": {
                        "validateConstraints": true,        // currently ignored because␣
↪AAS4J does not yet implement validation for AAS v3.0
                        "idShortUniqueness": true,
                        "identifierUniqueness": true
                },
                "validationOnCreate": {
                        "validateConstraints": false,        // currently ignored␣
```

(continues on next page)

(continued from previous page)

```
↪because AAS4J does not yet implement validation for AAS v3.0
                        "idShortUniqueness": true,
                        "identifierUniqueness": true
                },
                "validationOnUpdate": {
                        "validateConstraints": false,        // currently ignored␣
↪because AAS4J does not yet implement validation for AAS v3.0
                        "idShortUniqueness": true,
                        "identifierUniqueness": true
                }
        },
        // ...
}
```

2.4.2 Configuring Interface Implementations

For each interface in the architecture, you can choose one (or sometimes multiple) interface(s) to be used. As every interface implementation may require different configuration properties which FA³ST does not know about (as the implementation may be developed by 3rd parties at any time), the configuration section for each interface implementation uses the following structure

Listing 5: Common structure for configuring an interface implementation.

```
{
        "@class" : "...",     // fully-qualified Java class name of the class␣
↪implementing the interface
        // implementation-specific configuration properties
}
```

Which properties are available for each implementation should be documented, e.g., for all default implementations these properties are documented in the corresponding page of the documentation for each of the implementations.

The following shows an example of a configuration using and HTTP endpoint with port 443.

Listing 6: Example configuration with HTTP endpoint using port 443.

```
{
        "endpoints" : {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.endpoint.http.
↪HttpEndpoint",
                "port" : 443
        },
        // ...
}
```

2.4.3 Using 3rd Party Interface Implementations

For FA³ST to be able to load an implementation that is not pre-packaged with FA³ST, you need to put a JAR file containing the respective class in the same directory as the FA³ST Service JAR. Furthermore, all dependencies of that class need also be resolvable. This can be achieved by either packaging them into the same JAR (e.g. using the Maven Shade Plugin) or manually providing the required JAR files alongside the implementation.

2.4.4 Providing certificates in configuration

Multiple components of FA³ST Service make use of certificates, either by using them for their own services or by trusting the provided certificates. The default way to exchange certificates in FA³ST Service is via Java KeyStores. To simplify configuration, the same configuration object is re-used across different components, for example in the *HTTP Endpoint*. The structure of the certificate-related configuration object is explained in the following.

Table 3: Configuration properties of generic certificate section.

Name	Allowed Values	Description	Default Value
keyPassword	String	The password for the key.**Warning: may cause unexpected behavior if not set or set to empty string in some cases**	
keyStorePassword	String	The password for the key store	
keyStorePath	String	File containing the key store	
keyAlias*(optio*	String	The alias to use, e.g. when loading a certificate and the key store contains multiple entries	null, i.e. first entry will be used
keyStoreType*(optional)*	String	Type of the KeyStore, e.g. PKCS12 or JKS	PKCS12

Listing 7: Example certificate information

```
{
        "keyStoreType": "PKCS12",
        "keyStorePath": "C:\faaast\MyKeyStore.p12",
        "keyStorePassword": "changeit",
        "keyAlias": "server-key",
        "keyPassword": "changeit"
}
```

2.4.5 Overriding Config Properties

As indicated by the last row in the above table, any config property can be overridden both via CLI or via environment variables.

Via CLI

Via CLI this is done by using the JSONPath expression to the property within the config file but without the `$.` part JSONPath expression typically start with.

For example, to override the `requestHandlerThreadPoolSize` property call FA³ST Service like this

```
> java -jar starter-{version}.jar [any other CLI arguments] core.
↪requestHandlerThreadPoolSize=42
```

To access configuration properties inside an array or list use array notation, e.g., `endpoints[0].port=8081`

Via Environment Variables

Overriding configuration properties via environment variables is similar to overriding them via CLI with two differences

1. Add the prefix *faaast_config_extension_*
2. Replace `.` that separate the JSONPath with `_`

Applying the previous examples yields `faaast_config_extension_core_requestHandlerThreadPoolSize=42` to update the property `requestHandlerThreadPoolSize` and `faaast_config_extension_endpoints[0]_port=8081` to update the port of the HTTP endpoint.

2.5 Endpoint

The `Endpoint` interface is responsible for communication with the AAS from the outside, e.g. users or external applications. An instance of FA³ST Service can serve multiple endpoints at the same time. Endpoints will be synchronized, meaning if a FA³ST Service offers multiple endpoint such as HTTP(S) and OPC UA at the same time, changes done via one of the endpoints like updating a value is reflected in the other.

The following is an example of the relevant part of the configuration part comprising both an HTTP(S) and OPC UA endpoint

Listing 8: Example configuration for running both an HTTP and OPC UA endpoint.

```
{
        "endpoints": [
                {
                        "@class": "de.fraunhofer.iosb.ilt.faaast.service.endpoint.http.
↪HttpEndpoint",
                        "port": 443,
                        "corsEnabled": true
                },
                {
                        "@class": "de.fraunhofer.iosb.ilt.faaast.service.endpoint.opcua.
↪OpcUaEndpoint",
```

(continues on next page)

(continued from previous page)

```
                        "tcpPort": 8081
                }
        ],
        // ...
}
```

2.5.1 HTTP

The HTTP Endpoint allows accessing data and execute operations within the FA³ST Service via REST-API. In accordance to the specification, only HTTPS is supported since AAS v3.0. The HTTP Endpoint is based on the document Details of the Asset Administration Shell - Part 2: Application Programming Interfaces v3.0 and the corresponding OpenAPI documentation v3.0.1.

Configuration

Table 4: Configuration properties of HTTP Endpoint.

Name	Allowed Value	Description	Default Value
certificate*(option*	*Certificate-Info*	The HTTPS certificate to use.	self-signed certificate
corsEnabled*(optio*	Boolean	If Cross-Origin Resource Sharing (CORS) should be enabled.Typically required if you want to access the REST interface from any machine other than the one running FA³ST Service.	false
port*(option*	Integer	The port to use.	443
sniEnabled*(optio*	Boolean	If Server Name Identification (SNI) should be enabled.**This should only be disabled for testing purposes as it may present a security risk!**	true

Listing 9: Example configuration section for HTTP Endpoint.

```
{
        "endpoints": [ {
                "@class": "de.fraunhofer.iosb.ilt.faaast.service.endpoint.http.
↪HttpEndpoint",
                "port": 443,
                "corsEnabled": true,
                "sniEnabled": true,
                "certificate": {
                        "keyStoreType": "PKCS12",
                        "keyStorePath": "C:\faaast\MyKeyStore.p12",
                        "keyStorePassword": "changeit",
                        "keyAlias": "server-key",
                        "keyPassword": "changeit"
                }
        } ],
```

(continues on next page)

(continued from previous page)

```
        // ...
}
```

API

FA³ST Service supports the following APIs as defined by the OpenAPI documentation v3.0.1

- Asset Administration Shell API
- Submodel API
- Asset Administration Shell Repository API
- Submodel Repository API
- Concept Description API
- Asset Administration Shell Basic Discovery API
- Serialization API
- Description API

Using HTTP PATCH

As the AAS specification is currently does not properly specify show HTTP PATCH requests are expected to work, FA³ST Service follows the well-established RFC 7386 JSON Merge Patch. In short, this means that as payload you can send a JSON document that only contains the properties of the original document you want to update. To delete elements set the value explicitely to null.

As a consequence, URLs for all different content modifiers, i.e. `/$metadata`, `/$value`, as well as the call without any modifiers, are redundant and provide exactly the same functionality in FA³ST Service.

Caution: Arrays in JSON objects can only be replaced, i.e. if you want to update a single element within an array you first need to get the current value of the array, modify the element to be updated and then send the whole array as part of the PATCH payload.

Invoking Operations

To invoke an operation, make a `POST` request according to this URL example: `/submodels/{submodelId (base64-URL-encoded)}/submodel-elements/{idShortPath to operation}/invoke`.

Tip: You can invoke operations asynchronuously by calling `.../invoke-async` instead of `.../invoke` in which case you get back a `handleId` instead of the result. To monitor the execution state call `.../operation-status/{handleId}` and once finished you can get the result calling `.../operation-results/{handleId}` or `.../operation-results/{handleId}/$value` for the ValueOnly serialization.

Depending on the in & inoutput arguments, the payload should look like this.

Listing 10: Example payload for invoking operations synchronously

```
{
        "inputArguments": [ {
                "value": {
                        "modelType": "Property",
                        "value": "4",
                        "valueType": "xs:int",
                        "idShort": "in"
                },
                // additional input arguments
        } ],
        "inoutputArguments": [ {
                "value": {
                        "modelType": "Property",
                        "value": "original value",
                        "valueType": "xs:string",
                        "idShort": "note"
                },
                // additional inoutput arguments
        } ],
        "clientTimeoutDuration": "PT10S"   // ISO8601 duration, here: 10 seconds
}
```

An easier, or at least less verbose way, of invoking operations is by using the ValueOnly serialization. For this, add `/$value` to the end of the URL, i.e. resulting in either `.../invoke/$value` or `.../invoke-async/$value`. The payload will be simplified and look similar to this

Listing 11: Example payload for invoking operations with ValueOnly

```
{
        "inputArguments": {
                "in": 4
        },
        "inoutputArguments": {
                "note": "original value"
        },
        "clientTimeoutDuration": "PT10S"
}
```

2.5.2 OPC UA

The OPC UA Endpoint allows accessing data and execute operations within the FA³ST Service via OPC UA.

Unfortunately, there is currently no official mapping of the AAS API to OPC UA for AAS v3.0. Nevertheless, FA³ST Service decided to still provide an OPC UA endpoint even though it is not (yet) standard-compliant. This implementation is based on the OPC UA Companion Specification OPC UA for Asset Administration Shell (AAS) which defines a mapping between AAS and OPC UA for AAS v2.0 enriched with some custom adjustments and extensions to be used with AAS v3.0.

The OPC UA Endpoint is built with the Prosys OPC UA SDK for Java which means in case you want to compile the OPC UA Endpoint yourself, you need a valid license for the SDK (which you can buy here. For evaluation purposes, you also have the possibility to request an evaluation license. However, this is not necessary for using the OPC UA

Endpoint we already provide a pre-compiled version that is used by default when building FA³ST Service from code. The developers of the Prosys OPC UA SDK have been so kind to allow us to publish that pre-compiled version as part of this open-source project under the condition that all classes related to their SDK are obfuscated.

Configuration Parameters

OPC UA Endpoint configuration supports the following configuration parameters

Name	Allowed Value	Description	Default Value
discoveryServerUrl(o	String	URL of the discovery server.If empty, discovery server registration is disabled.	
secondsTillShutdown(optio	Integer	The number of seconds the server waits for clients to disconnect	2
serverCertificateBasePath(o	String	Path where the server application certificates are stored	PKI/CA
supportedAuthentications(option	AnonymousUserNameCertificate	List of supported authentication types	Anonymous
supportedSecurityPolicies(optionc	NONEBASIC128RSA15BASIC256BASIC2	List of supported security policies	NONE,BASIC256SHA256,AES128_SHA25 SS
tcpPort(optionc	Integer	The port to use for TCP	4840
userMap(op	Map<String, String>	A map containing usernames and password.If *UserName* is not included in `supportedAuthentications`, this property is ignored.	*empty*
userCertificateBasePath(o	String	Path where the certificates for user authentication are saved	USERS_PKI/CA

Certificate Management

The path provided with the `serverCertificateBasePath` configuration property stores the server and client application certificates and contains the following subdirectories

- `/certs`: trusted client certificates
- `/crl`: certificate revocation list for client certificates
- `/issuers/certs`: certificates of trusted CAs
- `/issuers/crl`: certificate revocation list for CA certificates
- `/issuers/rejected`: rejected CA certificates
- `/private`: certificates for the OPC UA server
- `/rejected`: unkown/rejected client certificates

To provision the OPC UA Endpoint to use an existing certificate for the server, save the certificate file as `{serverCertificateBasePath}/private/Fraunhofer IOSB AAS OPC UA Server@{hostname}_2048.der` and the private key as `{serverCertificateBasePath}/private/Fraunhofer IOSB AAS OPC UA Server@{hostname}_2048.pem` where `{hostname}` is the host name of your machine.

When an unkown client connects to the OPC UA Endpoint, the connection will be rejected and its client certificate will be stored in `/rejected`. To trust the certificate of a client and allow the connection, move the file to `/certs`.

The path provided with the `userCertificateBasePath` configuration property stores the user certificates and contains the following subdirectories

- `/certs`: trusted user certificates
- `/crl`: certificate revocation list for user certificates
- `/issuers/certs`: certificates of trusted CAs
- `/issuers/crl`: certificate revocation list for CA certificates
- `/issuers/rejected`: rejected CA certificates
- `/rejected`: unkown/rejected client certificates

Similar to the client certificates, unknown user certificates are stored in `/rejected` the first time a new certificate is encountered. To trust this certificate, simply move it to `/certs`.

and `userCertificateBasePath` point to directories where the corresponding certificates are stored. These directories contain the following subdirectories:

Listing 12: Example configuration for OPC UA Endpoint.

```
{
        "endpoints": [ {
                        "@class": "de.fraunhofer.iosb.ilt.faaast.service.endpoint.opcua.
↪OpcUaEndpoint",
                        "tcpPort" : 18123,
                        "secondsTillShutdown" : 5,
                        "discoveryServerUrl" : "opc.tcp://localhost:4840",
                        "userMap" : {
                          "user1" : "secret"
                        },
                        "serverCertificateBasePath" : "PKI/CA",
                        "userCertificateBasePath" : "USERS_PKI/CA",
                        "supportedSecurityPolicies" : [ "NONE", "BASIC256SHA256",
↪"AES128_SHA256_RSAOAEP" ],
                        "supportedAuthentications" : [ "Anonymous", "UserName" ]
        } ],
        //...
}
```

OPC UA Client Libraries

To connect to the OPC UA Endpoint, you need an OPC UA Client. Here are some example libraries and tools you can use:

- Eclipse Milo: Open Source SDK for Java.
- Unified Automation UaExpert: Free OPC UA test client (registration on website required for download).
- Prosys OPC UA Browser: Free OPC UA test client (registration on website required for download).
- Official Samples from the OPC Foundation: C#-based sample code from the OPC Foundation.

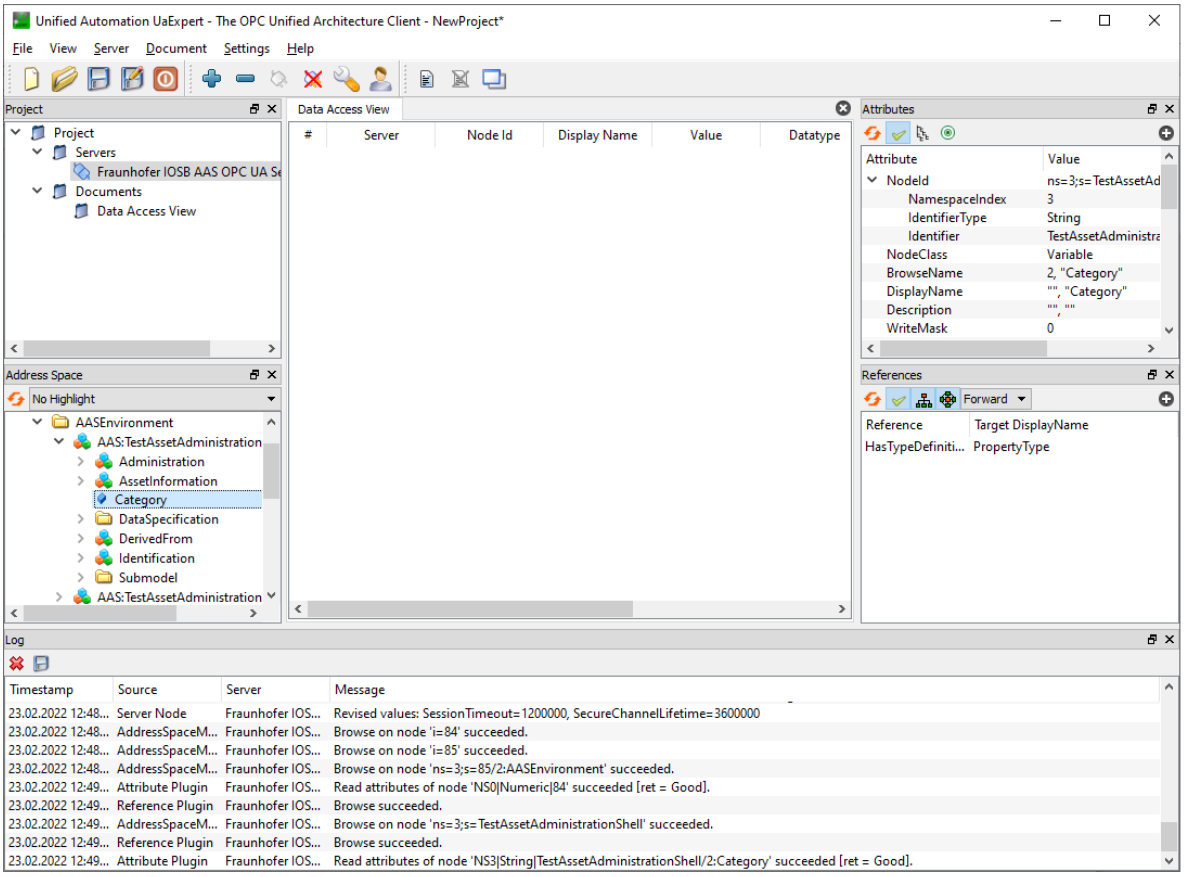

Fig. 2: Screenshot showing UaExpert connected to a FA³ST Service via OPC UA Endpoint.

API

As stated, there is currently no official mapping of the AAS API to OPC UA for AAS v3.0 but FA³ST Service implements its proprietary adaption of the mapping for AAS v2.0.

Supported Functionality

- Writing values for the following types
  - Property
  - Range
  - Blob
  - MultiLanguageProperty
  - ReferenceElement
  - RelationshipElement
  - Entity
- Operations (OPC UA method calls). Exception: Inoutput-Variables are not supported in OPC UA.

Not (yet) Supported Functionality

- Updating the model, i.e., adding new elements at runtime is not possible
- Writing values for the following types
  - DataSpecifications
  - Qualifier
  - Category
  - ModelingKind
- AASDataTypeDefXsd
  - Base64Binary
  - UnsignedInt
  - UnsignedLong
  - UnsignedShort
  - UnsignedByte

2.6 AssetConnection

The AssetConnection interface is responsible for synchronizing values of the model with assets. Although asset synchronization is not part of the AAS specification, we believe this functionality is essential for digital twins, at least when located on edge-level, i.e., close to an actual machine/asset.

The following figure depicts how asset synchronization works in more detail.

The top half shows a examplenary AAS model that we want to synchronize with the underlying asset. In the center we have the AssetConnection interface which holds multiple of so-called *Provider*s. There are three types of providers:

- ValueProvider: for reading data from and writing data to to asset whenever the value of the corresponding AAS element is read/written
- OperationProvider: for forwarding operation invocation requests to the asset and translating the response back to be AAS-compliant

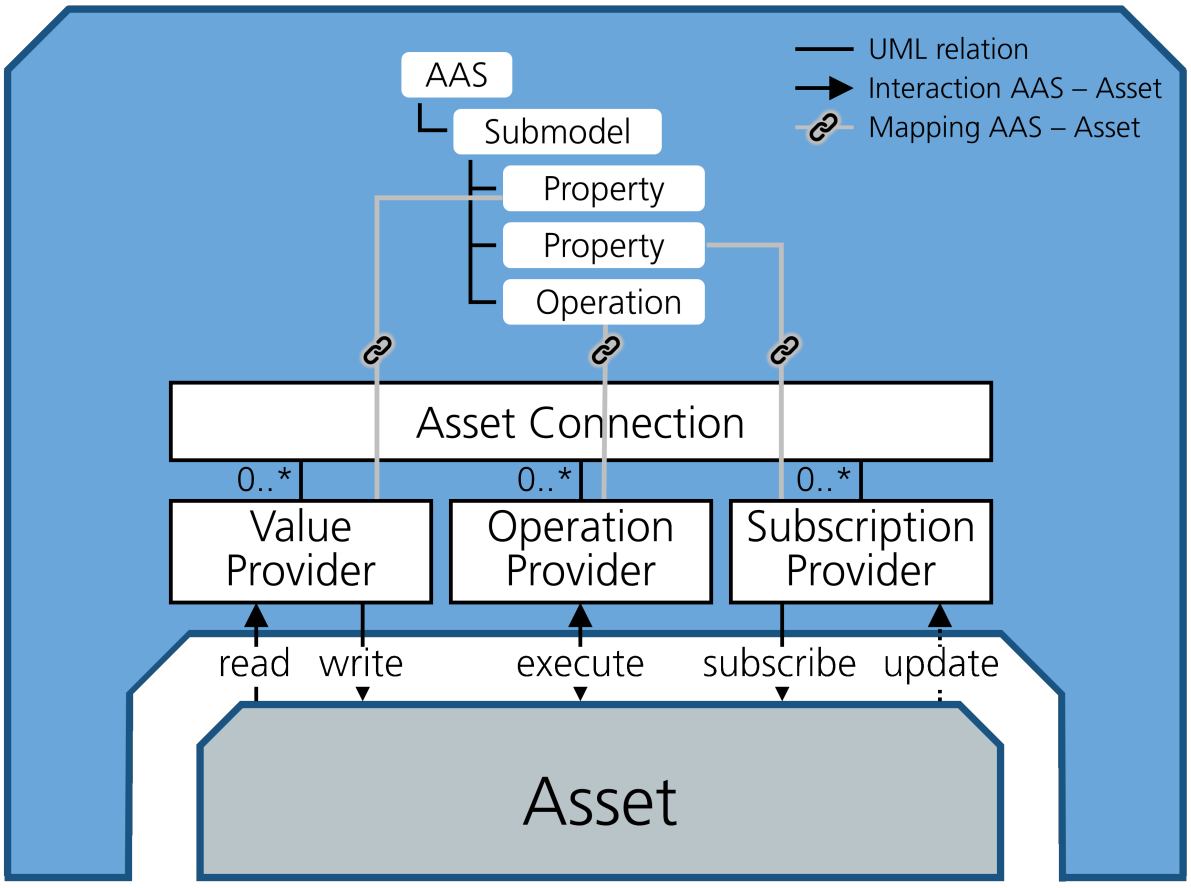

Fig. 3: How AssetConnection works in FA³ST Service.

- SubscriptionProvider: for subscribing to changes on the asset and therefore continuously updating the value of the corresponding AAS element

The mapping between AAS elements and providers is defined in the configuration of the AssetConnection. Therefore, the configuration section for all implementations of the AssetConnection interface share the following common structure.

Listing 13: Common configuration structure for all AssetConnection implementations.

```
{
        "assetConnections": [ {
                "@class": "...",
                // connection-level configuration
                "valueProviders":
                {
                        "{serialized Reference of AAS element}":
                        {
                                // value provider configuration
                        }
                },
                "operationProviders":
                {
                        "{serialized Reference of AAS element}":
                        {
                                // operation provider configuration
                        }
                },
                "subscriptionProviders":
                {
                        "{serialized Reference of AAS element}":
                        {
                                // subscription provider configuration
                        }
                }
        }],
        //...
}
```

The value of `{serialized Reference of AAS element}` is the Reference to the AAS element serialized using the rules described in Section 7.2.3 of AAS Specification - Part 1. An example value could look like this `[ModelRef](Submodel)urn:aas:id:example:submodel:1, (Property)Property1`.

Important: The format for serializing references has changed with AAS v3.0 resp. FA³ST Service v1.0. For example, the id type is now no longer part of the serialization and path elements are now separated by `, ` (comma followed by space) instead of `,` (comma).

The available configuration properties for connection-level and the providers are implementation-specific. This is necessary because different protocols require different types of information, e.g. for OPC UA an AAS element could be mapped to an OPC UA node which means the configuration must contain the node ID, while for MQTT we need a topic on which to listen and maybe even information about the payload format.

Note: An implementation does not have to implement all three provider types. In fact, it is often not possible to

implement all of them for a given network protocol as most protocols do not support pull-based and pub/sub mechanisms at the same time (e.g. HTTP, MQTT).

Tip: You can define both a ValueProvider and a SubscriptionProvider for the same element. This allows you to reflect in the asset changes in near real-time in your AAS and at the same time to update the value on the asset via the AAS API. This is especially useful when starting FA³ST with an OPC UA endpoint as it allows users to subscribe to changes or AAS properties via OPC UA.

2.6.1 OperationProvider Configuration

All OperationProvider share the following common set of configuration properties.

Table 5: Common configuration properties of OperationProviders.

Name	Allowed Value	Description	Default Value
inputValidationMode*(optional)*	NONEREQUIRE_PRESENTREQUIRE_	Validation mode for input arguments	REQUIRE_PRESENT_OR_DEFAULT
inoutputValidationMode*(optional)*	NONEREQUIRE_PRESENTREQUIRE_	Validation mode for inoutput arguments	REQUIRE_PRESENT_OR_DEFAULT
outputValidationMode*(optional)*	NONEREQUIRE_PRESENTREQUIRE_	Validation mode for ouput arguments	REQUIRE_PRESENT_OR_DEFAULT

Validation of operation arguments

Validation of operation argument can be configured independently for in-, out-, and inoutput arguments to be one of the following values

- NONE: no validation at all is performed
- REQUIRE_PRESENT: requires all arguments defined for the operation to be provided in the call and all arguments provided to be defined for the operation. This check works only on argument name (idShort) and not argument datatype.
- REQUIRE_PRESENT_OR_DEFAULT: sets all arguments defined for the operation but not provided in the call to the default value, i.e. the value given in the definition of the argument. Similar to REQUIRE_PRESENT, this requires the call to only contain arguments that are defined for the operation and works only on argument name ignoring the argument datatype.

2.6.2 HTTP

Supported Providers

- ValueProvider
  - read ✓
  - write ✓
- OperationProvider ✓
- SubscriptionProvider ✓ (via polling)

Configuration

Connection-Level

Table 6: Configuration properties of HTTP AssetConnection.

Name	Allowed Value	Description	Default Value
baseUrl	String	Base URL of the HTTP server, e.g. *http://example.com*.	
headers*(optional)*	Map<String,Str	Headers to send with each request.	*empty list*
password*(optional)*	String	Password for connecting to the HTTP server.	
trustedCertificates*(optional)*	*CertificateInfo*	Trusted certificates, i.e. when connecting to a server that is using self-signed certificates.	
username*(optional)*	String	Username for connecting to the HTTP server.	

Value Provider

Table 7: Configuration properties of HTTP AssetConnection Value Provider.

Name	Allowed Value	Description	Default Value
format	JSONXML	Content format of the payload.	
headers*(optional)*	Map<String	Headers to send with each request.Overrides connection-level headers.	*empty list*
path	String	Path for the HTTP request, relative to the `baseUrl` of the connection.	
query*(optionc*	String	Additional information how to extract actual value from received messages.Depends on `format`, e.g. for JSON this is a JSONPath expression.	
template*(optiona*	String	Template used to format payload when sending via HTTP.	
writeMethod(	GETPUT-POST	HTTP method to use when writing a value to HTTP.	PUT

Listing 14: Example configuration section for HTTP AssetConnection.

```
{
        "format": "JSON",
        "path": "/foo",
        "headers": {
                "foo": "bar"
        },
        "query": "$.foo",
        "template": "{\"foo\" : \"${value}\"}",
```

(continues on next page)

(continued from previous page)

```
        "writeMethod": "POST"
}
```

Operation Provider

Table 8: Configuration properties of HTTP AssetConnection Operation Provider.

Name	Allowed Value	Description	Default Value
format	JSONXML	Content format of the payload.	
headers*(optional,*	Map<String,String>	Headers to send with each request.Overrides connection-level headers.	*empty list*
inputValidationMode*(optior*	NONEREQUIRE_PRE	Validation mode for input arguments	REQUIRE_PRESENT_OR_DEFAU
inoutputValidationMode*(optior*	NONEREQUIRE_PRE	Validation mode for inoutput arguments	REQUIRE_PRESENT_OR_DEFAU
method*(opti*	PUTPOST	HTTP method to use.	POST
outputValidationMode*(optior*	NONEREQUIRE_PRE	Validation mode for ouput arguments	REQUIRE_PRESENT_OR_DEFAU
path	String	Path for the HTTP request, relative to the `baseUrl` of the connection.	
queries*(opti*	Map<String,String>	Map of result variable idShorts and corresponding query expressions to fetch them from returned valueQuery expressions depend on `format`, e.g. for JSON this is a JSONPath expression.	
template*(optiona*	String	Template used to format payload when sending via HTTP.	

Listing 15: Example configuration section for HTTP OperationProvider for an Operation with input parameters `in1` and `in2` and output parameters `out1` and `out2`.

```
{
        "format": "JSON",
        "path": "/foo/execute",
        "headers": {
                "foo": "bar"
        },
        "method": "POST",
        "template": "{\"input1\" : \"${in1}\", \"input2\" : \"${in2}\"}",
        "queries": {
                "out1": "$.output1",
                "out2": "$.output2"
```

(continues on next page)

(continued from previous page)

```
        }
}
```

Subscription Provider

Table 9: Configuration properties of HTTP AssetConnection Subscription Provider.

Name	Allowed Value	Description	Default Value
format	JSONXML	Content format of the payload.	
headers*(optiona*	Map<String,	Headers to send with each request.Overrides connection-level headers.	*empty list*
interval*(optiona*	long	Interval to poll the server for changes (in ms).	100
method*(op*	GETPUT-POST	HTTP method to use when writing a value to HTTP.	GET
path	String	Path for the HTTP request, relative to the `baseUrl` of the connection.	
payload*(option*	String	Static content to send with each request.	
query*(optic*	String	Additional information how to extract actual value from received messages.Depends on `format`, e.g. for JSON this is a JSONPath expression.	

Listing 16: Example configuration section for HTTP Subscription-Provider.

```
{
        "path": "/foo",
        "headers": {
                "foo": "bar"
        },
        "interval": "500",
        "method": "GET",
        "template": "{\"foo\" : \"bar\"}"
}
```

2.6.3 MQTT

Supported Providers

- ValueProvider
  - read
  - write ✓
- OperationProvider
- SubscriptionProvider ✓

Configuration

Connection-Level

Table 10: Configuration properties of MQTT AssetConnection.

Name	Allowed Value	Description	Default Value
clientId*(optional)*	String	Id of the MQTT client used to connect to the server	*randomly generated*
password*(optional)*	String	Password for connecting to the MQTT server	
serverUri	String	URL of the MQTT server, e.g. *tcp://localhost:1883*	
username*(optional)*	String	Username for connecting to the MQTT server	

Value Provider

Table 11: Configuration properties of MQTT AssetConnection Value Provider.

Name	Allowed Value	Description	Default Value
format	JSONXML	Content format of the payload.	
topic	String	MQTT topic to use.	
template*(optional)*	String	Template used to format payload.	

Listing 17: Example configuration section for MQTT ValueProvider.

```
{
        "format": "JSON",
        "topic": "example/myTopic",
        "template": "{\"foo\" : \"${value}\"}"
}
```

Subscription Provider

Table 12: Configuration properties of MQTT AssetConnection Subscription Provider.

Name	Allowed Value	Description	Default Value
format	JSONXM	Content format of the payload.	
topic	String	MQTT topic to use.	
query*(opti*	String	Additional information how to extract actual value from received messages.Depends on `format`, e.g. for JSON this is a JSONPath expression.	

Listing 18: Example configuration section for MQTT SubscriptionProvider.

```
{
        "format": "JSON",
        "topic": "example/myTopic",
        "query": "$.foo"
}
```

2.6.4 OPC UA

Supported Providers

- ValueProvider
  - read ✓
  - write ✓
- OperationProvider ✓
- SubscriptionProvider ✓

Configuration

Connection-Level

Table 13: Configuration properties of OPC UA AssetConnection.

Name	Allowed Value	Description	Default Value
acknowledgeTimeout*(optional)*	Integer	Timeout for acknowledgement (in ms).	10000
applicationCertificate*(optional)*	*CertificateInfo*	The application certificate.	
authenticationCertificate*(optional)*	*CertificateInfo*	The authentication/user certificate.	
host	String	URL of the OPC UA server, e.g. *opc.tcp://localhost:4840*	
password*(optional*	String	Password for connecting to the OPC UA server.This value is required if `userTokenType` is set to `UserName`.	
requestTimeout*(optional)*	int	Timeout for requests (in ms)	3000
securityBaseDir*(optic*	String	Base directory for the certificate handling.	.
securityMode*(optionc*	NoneSignSignAndEncrypt	Security Mode for the connection to the OPC UA server.	None
securityPolicy*(optional)*	NoneBasic256Sha256Aes128_Sha256_RsaOa	Desired Security Policy for the connection to the OPC UA server.	None
transportProfile*(optional)*	TCP_UASC_UABINARYHTTPS_UABINAR	Transport Profile for the connection to the OPC UA server.	TCP_UASC_UABINARY
username*(optiona*	String	Username for connecting to the OPC UA server.This value is required if `userTokenType` is set to `UserName`.	
userTokenType*(optional*	AnonymousUserNameCertificate	User Token Type for connecting to the OPC UA server.	Anonymous

Remarks on certificate management

In OPC UA , certificates can be used for two purposes:

- encryption & signing of messages, and
- authentication of a client.

We call the certificate used for encryption *application certificate* and the one used for authenticating a client *authentication certificate*. You can choose to use only one of these options or both. If using both, you can use different or the same certificates.

Application Certificate

An application certificate is required if the property `securityMode` is set to `Sign` or `SignAndEncrypt`.

Which application certificate to use is determined by the following steps:

- `applicationCertificate.keyStorePath` if it is an absolute file path and the file exists (default: application.p12)
- `{securityBaseDir}/{applicationCertificate.keyStorePath}` if the file exists (default: `./{applicationCertificate.keyStorePath}`)
- otherwise generate self-signed certificate and store it at `applicationCertificate.keyStorePath` (if `applicationCertificate.keyStorePath` is an absolute file path) or else `{securityBaseDir}/{applicationCertificate.keyStorePath}`. The generated keystore will not be password protected.

You also need to make sure that the OPC UA client (which in this case is the FA³ST Service OPC UA asset connection) knows and trusts the server certificate and vice versa.

For the client to trust the server you need to do one of these steps depending on the certificate of the server:

- Self-signed-certificate: Put server certificate in {securityBaseDir}/pki/trusted/certs
- CA Certificate: put the CA root certificate in {securityBaseDir}/pki/issuers/certs and the corresponding certificate revocation list (CRL) in {securityBaseDir}/pki/issuers/crl.

If you don't have the server certificate at hand you can start FA³ST Service without providing/trusting the server certificate. On start-up FA³ST Service will try to connect to the server which will fail because the server certificate is not trusted yet. After that you will find the relevant files at `{securityBaseDir}/pki/rejected`. Copy them to the respective directories as described above. Once FA³ST Service tries to reconnect the connection should be established successfully.

For the server to trust your client application certificate please refer to the documentation of your OPC UA server.

Authentication Certificate

Which authentication certificate is used is determined by a similar logic as for the application certificate besides that this certificate is not auto-generated if not present:

- `authenticationCertificate.keyStorePath` if it is an absolute file path and the file exists (default: application.p12)
- `{securityBaseDir}/{authenticationCertificate.keyStorePath}` if the file exists (default: `./{authenticationCertificate.keyStorePath}`)

Value Provider

Table 14: Configuration properties of OPC UA AssetConnection Value Provider.

Name	Allowed Value	Description	Default Value
arrayIndex*(optional)*	String	Index of the desired array element if the node is an array.Can be multi-dimensional.	
nodeId	String	NodeId of the the OPC UA node to read/write in ExpandedNodeId format	

Listing 19: Example configuration section for OPC UA ValueProvider.

```
{
        "nodeId": "nsu=com:example;s=foo",
        "arrayIndex" : "[2]"
}
```

Operation Provider

Table 15: Configuration properties of OPC UA AssetConnection Operation Provider.

Name	Allowed Value	Description	Default Value
inputArgumentMapping*(optional*	List	List of mappings for input arguments between the idShort of a SubmodelElement and an argument name	*empty list*
inputValidationMode*(optiona*	NONEREQUIRE_PRES	Validation mode for input arguments	REQUIRE_PRESENT_OR_DEFAU
inoutputValidationMode*(optiona*	NONEREQUIRE_PRES	Validation mode for inoutput arguments	REQUIRE_PRESENT_OR_DEFAU
nodeId	String	NodeId of the the OPC UA node to read/write in ExpandedNodeId format	
outputArgumentMapping*(optional*	List	List of mappings for output arguments between the idShort of a SubmodelElement and an argument name	*empty list*
outputValidationMode*(optiona*	NONEREQUIRE_PRES	Validation mode for ouput arguments	REQUIRE_PRESENT_OR_DEFAU
parentNodeId*(optiona*	String	NodeId of the OPC UA object in ExpandedNodeId format, in which the method is contained.When no parentNodeId is given here, the parent object of the method is used.	

Listing 20: Example configuration section for OPC UA Operation Provider.

```
{
        "nodeId": "nsu=com:example;s=foo",
        "parentNodeId": "nsu=com:example;s=fooObject",
        "inputArgumentMapping": [ {
                "idShort": "ExampleInputId",
                "argumentName": "ExampleInput"
        } ],
        "outputArgumentMapping": [ {
                "idShort": "ExampleOutputId",
                "argumentName": "ExampleOutput"
```

(continues on next page)

(continued from previous page)

```
        } ]
}
```

Subscription Provider

Table 16: Configuration properties of OPC UA AssetConnection Subscription Provider.

Name	Allowed Value	Description	Default Value
arrayIndex*(optional)*	String	Index of the desired array element if the node is an array.Can be multi-dimensional.	
interval	long	Interval to poll the server for changes (in ms)**Currently not used**	1000
nodeId	String	NodeId of the the OPC UA node to read/write in ExpandedNodeId format	

Listing 21: Example configuration section for OPC UA Subscription Provider.

```
{
        "nodeId": "nsu=com:example;s=foo",
        "interval": 1000,
        "arrayIndex" : "[2]"
}
```

Complete Example

Listing 22: Complete example configuration section for OPC UA Asset Connection.

```
{
        "@class": "de.fraunhofer.iosb.ilt.faaast.service.assetconnection.opcua.
↪OpcUaAssetConnection",
        "host": "opc.tcp://localhost:4840",
        "securityPolicy": "None",
        "securityMode" : "None",
        "applicationCertificate": {
                "keyStoreType": "PKCS12",
                "keyStorePath": "C:\faaast\MyKeyStore.p12",
                "keyStorePassword": "changeit",
                "keyAlias": "app-cert",
                "keyPassword": "changeit"
        },
        "authenticationCertificate": {
                "keyStoreType": "PKCS12",
                "keyStorePath": "C:\faaast\MyKeyStore.p12",
                "keyStorePassword": "changeit",
                "keyAlias": "auth-cert",
```

(continues on next page)

(continued from previous page)

```
                "keyPassword": "changeit"
        },
        "valueProviders": {
                "[ModelRef](Submodel)urn:aas:id:example:submodel:1, (Property)Property1
↪": {
                        "nodeId": "some.node.id.property.1"
                },
                "[ModelRef](Submodel)urn:aas:id:example:submodel:1, (Property)Property2
↪": {
                        "nodeId": "some.node.id.property.2"
                }
        },
        "operationProviders": {
                "[ModelRef](Submodel)urn:aas:id:example:submodel:1, (Operation)Operation1
↪": {
                        "nodeId": "some.node.id.operation.1"
                }
        },
        "subscriptionProviders": {
                "[ModelRef](Submodel)urn:aas:id:example:submodel:1, (Property)Property3
↪": {
                        "nodeId": "some.node.id.property.3",
                        "interval": 1000
                }
        }
}
```

2.7 Persistence

The Persistence interface is responsible for storing the AAS model.

Each Persistence configuration supports at least the following configuration properties:

Table 17: Common configuration properties of for all Persistence implementations.

Name	Allowed Value	Description	Default Value
initialModel*(op*	String	An `Environment` object containing the model to load initially.This can only be set when used via code, not via configuration file.This has precedence over `initialModelFile` when both are set.	
initialModelFile*(option*	String	Path to a model file to load initially.	

2.7.1 In-Memory

The In-Memory Persistence keeps the AAS model in the local memory. This means, that once FA³ST Service is stopped or crashes, all changes made during runtime are lost.

Important: If you use In-Memory Persistence from code by setting the `initialModel` property, the passed instance of `Environment` will be modified directly (as always the case in Java with pass-by-reference). If you do not want the original instance to be modified by FA³ST Serivce, call `DeepCopyHelper.deepCopy(...)` with the `Environment` to create a copy before passing it to FA³ST.

Configuration

In-Memory Persistence has no additional configuration properties.

Listing 23: Example configuration for In-Memory Persistence.

```
{
        "persistence" : {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.persistence.memory.
↪PersistenceInMemory",
                "initialModel" : "{pathTo}/FAAAST-Service/misc/examples/model.json"
        },
        //...
}
```

2.7.2 File-based

The File-based Persistence stores the AAS model in a file according to the AAS specification. Therefore, changes are stored permanently even when FA³ST Service is stopped or crashes.

Important: Each modification of the model results in writing the whole model to the file which might become a performance issue for larger models.

Configuration

Table 18: Configuration properties of File-based Persistence.

Name	Allowed Value	Description	Default Value
dataDir(o	String	Path where the model file created by the persistence should be saved.	.
dataformat(optio	AASXJSONXM	Data format to use when storing.Ignored when `keepInitial` is set to `true`.	same as `initialModelFile`
keepInitial(optio	Boolean	If true, `initialModelFile` will not be modified but instead a copy will be created in `dataDir` where the changes will be saved. If false, all changes will be written directly to the `initialModelFile`.	true

Listing 24: Example configuration for File-based Persistence.

```
{
        "persistence" : {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.persistence.file.
↪PersistenceFile",
                "initialModelFile" : "{pathTo}/FAAAST-Service/misc/examples/model.json",
                "dataDir": ".",
                "keepInitial": true,
                "dataformat": "XML"
        },
        //...
}
```

2.8 FileStorage

The FileStorage interface is responsible for managing auxiliary files like thumbnails or files referenced by the AAS model.

2.8.1 In-Memory

The In-Memory FileStorage keeps all files stored in memory. This means, that once FA³ST Service is stopped or crashes, all changes made during runtime are lost.

Configuration

In-Memory FileStorage does not support any configuration parameters.

Example

Listing 25: Example configuration for In-Memory FileStorage.

```
{
        "fileStorage" : {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.filestorage.memory.
↪FileStorageInMemory"
        },
        //...
}
```

2.8.2 FileSystem

The FileSystem-based FileStorage keeps all files stored in the file system of the local machine. Any change request, such as changing a file, results in a change in the file system. Thus, changes are stored permanently.

Configuration

Table 19: Configuration properties of FileSystem FileStorage.

Name	Allowed Value	Description	Default Value
existingDataPath*(optional)*	String	A path/directory containing data that should be available on start-up.This data will never be modified or deleted.	
path*(optional)*	String	The path/directory to use for storing the files.	.

Example

Listing 26: Example configuration for FileSystem FileStorage.

```
{
        "fileStorage" : {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.filestorage.filesystem.
↪FileStorageFilesystem",
                "path": "./my/file/cache",
                "existingDataPath": "./my/initial/data"
        },
        //...
}
```

2.9 MessageBus

The MessageBus interface is used for communication between different components, for example to synchronize between endpoints. Therefore, the MessageBus is primarily designed for internal use but as it might also be useful for some applications and scenarios there might be implementations that expose the MessageBus to the outside world.

2.9.1 Events

The MessageBus works according to the publish/subscribe principle based on different types of events or event messages (which are subclasses of the abstract class `EventMessage`). Subscriptions are made to a kind of event, i.e. a subclass of `EventMessage` or even `EventMessage` itself (to receive all events). When subscribing to a class, all events of this class or any subclass are received.

This is the class hierarchy of available event classes/types

- `EventMessage` *(abstract)*: Superclass for all events, payload: a `Reference` to the subject element
  - `AccessEventMessage` *(abstract)*: Superclass for all types of access-based events
    - `ReadEventMessage` *(abstract)*: Superclass for all types of read-events, triggered each time an element is read via API
      - `ElementReadEventMessage`: Triggered when a `Referable` is read via API, payload: the referable (serialized according to the request, i.e. using the requested `SerializationModifier`)
      - `ValueReadEventMessage`: Triggered when the value of an element is read via API, payload: the element value
    - `ExecuteEventMessage` *(abstract)*: Superclass for all events related to executing operations
      - `OperationInvokeEventMessage`: Triggered when an operation is invoked/started, payload: input and inoutput parameters
      - `OperationFinishEventMessage`: Triggered when an operation is finished, payload: output and inoutput parameters
  - `ChangeEventMessage` *(abstract)*: Superclass for all types of changes
    - `ElementChangeEventMessage` *(abstract)*: Superclass for all types of structural changes, payload: the updated element
      - `ElementCreateEventMessage`: Triggered when an element is created
      - `ElementDeleteEventMessage`: Triggered when an element is deleted

- `ElementUpdateEventMessage`: Triggered when an element is updated
- `ValueChangeEventMessage`: Triggered when the value of an element is updated, payload: old value, new value

– `ErrorEventMessage`: Triggered when an error occurred, payload: message, error level (INFO, WARN, ERROR)

2.9.2 Internal

This is the default implementation of the MessageBus interface which is implemented using Java method calls. Therefore, it can only be accessed from code when FA³ST Service is used as an embedded library.

Configuration

This implementation does not offer any configuration properties.

Listing 27: Example configuration for Internal MessageBus.

```
{
        "messageBus": {
                "@class": "de.fraunhofer.iosb.ilt.faaast.service.messagebus.internal.
↪MessageBusInternal"
        },
        //...
}
```

2.9.3 MQTT

This implementation of the `MessageBus` interface publishes messages via MQTT either by hosting its own MQTT server or by using an externally hosted one.

Topics & Payload

Each message type is published on its own topic in the form of `[topicPrefix]/[className]`, e.g. `events/ValueChangeEventMessage`. The payload is a JSON serialization of the corresponding Java class with the following base structure

Listing 28: JSON structure of serialized MessageBus events.

```
{
        "@type": "[event type]",
        "element": {
                // [default JSON serialization of Reference]
        },
        // [event-specific properties]
}
```

An example `ValueChangeEvent` might look like this:

Listing 29: JSON serialization of an example ValueChangeEvent.

```
{
    "@type": "ValueChangeEvent",
    "element": {
        "keys": [
            {
                "idType": "Iri",
                "type": "Submodel",
                "value": "http://example.org/submodel"
            },
            {
                "idType": "IdShort",
                "type": "Property",
                "value": "property"
            }
    ] },
    "oldValue": {
        "modelType": "Property",
        "dataType": "int",
        "value": 0
    },
    "newValue": {
        "modelType": "Property",
        "dataType": "int",
        "value": 1
    }
}
```

For deserialization of events the class `JsonEventDeserializer` in module `dataformat-json` can be used.

Configuration

Table 20: Configuration properties of MQTT MessageBus.

Name	Allowed Value	Description	Default Value
clientCertificate*(optional)*	*CertificateInfo*	The client certificate to use. If not set, SSL will be disabled.	
clientId*(optional)*	String	ClientId to use when connecting to the MQTT server.	FAST MQTT MessageBus
host*(optional)*	String	The host name of the MQTT server without prefix, e.g., 192.168.0.1.	localhost
password*(optional)*	String	Password used to connect to the MQTT server.	
port*(optional)*	Integer	The port to use for TCP communication.	1883
serverCertificate*(optional)*	*CertificateInfo*	The server certificate to use. If not set, SSL will be disabled.	
sslPort*(optiona*	Integer	The port to use for secure TCP communication.	8883
sslWebsocketPort*(optional)*	Integer	The port to use for secure websocket communication.	443
topicPrefix*(optional)*	String	Prefix to use for the topic names.	events/
useInternalServer*(optic*	Boolean	If true, FA³ST Service starts its own MQTT server.If false, FA³ST Service uses external MQTT server.	true
username*(optional*	String	Username used to connect to the MQTT server.	
users*(optional)*	Map<String, String>	Map of usernames and passwords of users that are allowed to connect to the MQTT server.This is only used when `useInternalServer` is true	*empty list*
useWebsocket*(optiona*	Boolean	If true uses websocket, otherwise TCP.	false
websocketPort*(optional)*	Integer	The port to use for TCP communication	9001

Listing 30: Example configuration for MQTT MessageBus.

```
{
        "messageBus": {
                "@class": "de.fraunhofer.iosb.ilt.faaast.service.messagebus.mqtt.MessageBusMqtt",
                "useInternalServer": true,
                "port": 1883,
                "sslPort": 8883,
                "host": "localhost",
                "websocketPort": 9001,
                "sslWebsocketPort": 443,
                "serverCertificate": {
                        "keyStoreType": "PKCS12",
                        "keyStorePath": "C:\faaast\MyKeyStore.p12",
                        "keyStorePassword": "changeit",
```

(continues on next page)

(continued from previous page)

```
                        "keyAlias": "server-key",
                        "keyPassword": "changeit"
                },
                "clientCertificate": {
                        "keyStoreType": "PKCS12",
                        "keyStorePath": "C:\faaast\MyKeyStore.p12",
                        "keyStorePassword": "changeit",
                        "keyAlias": "client-key",
                        "keyPassword": "changeit"
                },
                "users": {
                        "user1": "password1"
                },
                "username": "messagebus-user",
                "password": "messagebus-password",
                "clientId": "CustomClientId",
                "topicPrefix": "faaast/events/"
        },
        //...
}
```

2.10 Release Notes

2.10.1 1.1.0-SNAPSHOT (current development version)

New Features & Major Changes

- General
  - Loading AAS models from JSON now fails on unknown JSON properties

Internal changes & bugfixes

- General
  - Added log message when starting to indicate that constraint validation is currently not supported

2.10.2 1.0.1

- General
  - fixed some date-related unit tests that failed during daylight saving time at thereby prevented compilation of project

2.10.3 1.0.0

Important: Version 1.0 is a major update and has breaking changes to all previous versions. When upgrading to v1.0 please make sure the AAS models and payload you use are compliant to AAS spec v3.0. Additionally, existing asset connection configurations must be updated in the config file as the serialization format for references has changed in the specificition, e.g., `(Submodel)[IRI]http://example.org/foo,(Property)[ID_SHORT]bar` in older configurations new becomes `(Submodel)http://example.org/foo, (Property)bar`, i.e. the id type has been removed and segments are not separated only by `,` but by `, ` (comma followed by additional space).

New Features & Major Changes

- General
  - Updated to AAS metamodel & API v3.0, i.e. older model file (compliant with v2.x, v3.xRCxx) can no longer be loaded with FA³ST Service as-is but have be converted to v3.0
  - Now requires Java 17+
  - Replaced AAS model & de-/serialization library, now using AAS4J (previously Java Model and Java Serializer
  - Default filename for model files changed to `model.*` (previously `aasenvironment.*`)
  - Unified way to configure certificate information (*See details*). Affected components: HTTP Asset Connection, OPC UA Asset Connection, HTTP Endpoint, MQTT MessageBus
  - Environment variables now use _ instead of `.` as a separator
  - Validation - **currently completely disabled as AAS4J does not yet offer validation support**
    - More fine-grained configuration of validation via configuration file
    - Enabled validation for API calls creating or updating elements (basic validation enabled by default)
    - Renamed CLI argument `--no-modelValidation` to `--no-validation`. It now enables any validation when used (overriding validation configuration in configuration file is present)
  - Renamed CLI argument `--emptyModel` to `--empty-model`
- Endpoint
  - HTTP
    - Updated to AAS API specification v3.0.1
      - HTTP no longer supported, only HTTPS
      - all URLs are now prefixed with /api/v3.0/
    - Added support for AASX serialization
    - Added support for uploading, deleting and modifying of asset thumbnails and file attachments through API
  - OPC UA Endpoint
    - Updated OPC UA Information model to AAS specification version 3.0. As there is no official mapping of AAS v3.0 to OPC UA, the current mapping is proprietary
    - Added support for configuring supported security policies (`NONE`, `BASIC128RSA15`, `BASIC256`, `BASIC256SHA256`, `AES128_SHA256_RSAOAEP`, `AES256_SHA256_RSAPSS`) and authentication methods (`Anonymous`, `UserName`, `Certificate`)
- MessageBus

  - MQTT-based MessagBus now available that supports running both as embedded MQTT server or using external one

- Asset Connection
  - HTTP
    * Now provides a way to explicitely trust server certificates, e.g. useful when servers are using a self-signed certificate

- File-storage
  - New file-storage interface provides functionality to store referenced files like thumbnails and files in SubmodelElements
  - Implementations for filesystem- and memory-based storages

Internal changes & bugfixes

- General
  - Fixed a `ConcurrentModificationException` that could occur when accessing a submodel with subscription-based asset connection via HTTP endpoint

- HTTP Endpoint
  - Now correctly uses base64URL-encoding for all HTTP requests (instead of base64-encoding for some)
  - No longer leaks sensitive server information in HTTP response headers (such as server version of the HTTP server library)

- Asset Connection
  - OPC UA
    * Unit tests no longer create temp files in source folders

- Starter
  - Improved error logging

2.10.4 0.5.0

New Features & Major Changes

- Improved exception handling in CLI - upon error starter application should now correctly terminate with error code 1

- OPC UA Endpoint
  - Additional parameters available in configuration

- Docker container now runs using a non-root user

- Base persistence configuration updated
  - changed `initialModel` from filename to `AASEnvironment` object
  - added `initialModelFile`
  - removed `decoupleEnvironment` property. To achieve previous behavior you need to manually decouple the model by making a deep copy, e.g. via `DeepCopyHelper.deepCopy(...)`

- Asset Connection
  - OPC UA

    * Support mapping to specific element in (multi-dimensional) array/vector
    * Additional parameters available in configuration: requestTimeout, acknowledgeTimeout, retries, securityPolicy, securityMode, securityBaseDir, transportProfile, userTokenType, applicationCertificateFile, applicationCertificatePassword, authenticationCertificateFile, authenticationCertificatePassword

Internal changes & bugfixes

- General
  - Improved startup process & console ouput
- HTTP Endpoint
  - DELETE requests now correctly return HTTP status code `204 No Content`. The following URL patterns are affected:
    * /submodels/{submodelIdentifier}
    * /submodels/{submodelIdentifier}/submodel/submodel-elements/{idShortPath}
    * /shells/{aasIdentifier}/aas/submodels/{submodelIdentifier}/submodel/submodel-elements/{idShortPath}
  - Using not allowed HTTP methods not correctly returns `405 Method Not Allowed` instead of `500 Internal Server Error`
  - Unsupported URLS (valid URLs with additional path elements) now correctly return `400 Bad Request` instead of `405 Method not allowed`
  - GET /shells/{aasIdentifier} now correctly returns status code `404 Not Found` when called with an existing ID that is not an AAS (instead of `500 Internal Server Error`)
- OPC UA Endpoint
  - Major code refactoring
- Persistence
  - Major code refactoring
- Asset Connection
  - Fixed endless feedback loop when adding a subscription provider and value provider to the same element
  - OPC UA
    * fixed deserialization error when using operation provider with argument mappings
  - HTTP
    * subscription provider now only fires when then value has changed (before that it fired with any read)
- Miscellaneous
  - Now using dockerfile to build docker container instead of jib maven plugin

2.10.5 0.4.0

New Features

- Improved logging (new CLI arguments `-q`, `-v`, `-vv`, `-vvv`, `--loglevel-faaast`, `--loglevel-external`)

Internal changes & bugfixes

- Asset Connection
  - OPC UA
    - Fixed problem converting DateTime values
- Fixed error related to JSONPath expressions that could occure in asset connections when using certain JSONPath expressions
- Fixed error in reference helper with setting proper type of key elements when an identifiable and a independant referable have the same idshort
- Removed dependencies on checks module which is only needed for codestyle check while compiling and therefore not released on maven. This caused a missing dependency exception when using any FA³ST module within your code.

2.10.6 0.3.0

New Features

- Asset Connection
  - OPC UA
    - Automatic reconnect upon connection loss
    - Add ParentNodeId to OpcUaOperationProviderConfig
    - Introduce mapping between IdShort and Argument Name in OpcUaOperationProviderConfig
  - MQTT
    - Automatic reconnect upon connection loss
  - HTTP
    - Now supports adding custom HTTP headers (on connection- & provier-level)
- Improved JavaDoc documentation
- Improved security through automatic vulnerabilities check before release
- Added example how to implement custom asset connection

Internal changes & bugfixes

- Dynamic loading of custom implementations (AssetConnection, Persistence, MessageBus, Endpoint and Dataformat) now works as expected. NOTE: This requires package your custom implementation as a fat jar and put it in the same location as the FA³ST starter jar.
- Streamlining dependencies
- Improved console output for file paths
- Added checks to ensure model paths provided are valid
- Asset Connection

  - OPC UA
    * Fix problem when InputArguments or OutputArguments node was not present for Operations
    * Use ExpandedNodeId to parse NodeId Strings
  - HTTP
    * Fixed problem when using HttpAssetConnection configuration
- Development
  - Enforce JavaDoc present at compile-time (through checkstyle)
  - No longer release `test` module
  - Create javadoc jar for parent POM

2.10.7 0.2.1

Bugfixes

- Asset connections could not be started with OperationProvider
- Returning wrong HTTP responses in some cases

2.10.8 0.2.0

New Features

- Persistence
  - File-based persistence added
  - Each persistence implementation can now be configured to use a given AAS model as initial value
- Asset Connection
  - HTTP asset connection added
  - Basic authentication (username & password) added for OPC UA, MQTT and HTTP
  - Introducing protocol-agnostic library for handling different payload formats including extracting relevant information from received messages as well as template-based formatting of outgoing messages (currently only implemented for JSON)
- HTTP Endpoint
  - API
    * `Submodel Interface` calls now also available in combination with `Asset Administration Shell Interface`, e.g. /shells/{aasIdentifier}/aas/submodels/{submodelIdentifier}/submodel
    * `Asset Administration Shell Serialization Interface` now supported (at /serialization)
  - Support for output modifier `content=path`
  - CORS support, can be enabled by setting `isCorsEnabled=true` in config (default: false)
  - now returns status code 405 Method Not Allowed if URL is correct but requested method is not supported
- Support for `valueType=DateTime`
- Support for Java 16
- Improved robustness (e.g. against common invalid user input or network issues)

- Improved console output (less verbose, always displays version info)
- Improved documentation

Internal changes & smaller bugfixes

- Validation now checks for unsupported datatypes
- Version info correctly displayed when started as docker container or via local build/debug
- Fixed potential crash when initializing value with empty string althtough that is not a valid value according to the value type, e.g. int, double, etc. (empty string value is treated the same as null)
- Asset Connection
  - Fixed error when using operation provider
  - OPC UA
    - subscription provider now syncs value upon initial connect instead of waiting for first value change on server
  - MQTT
    - print warning upon connection loss
    - properly handle invalid messages without crashing
- Added strict enforcement of valid output modifiers for each API call
- Dynamically allocate ports in unit tests
- Add builder classes for event messages & config classes
- Replace AASEnvironmentHelper with methods of EnvironmentSerialization

2.10.9 0.1.0

First release!

2.11 About the Project

The Reference Architecture of Industrie 4.0 (RAMI) presents the Asset Administration Shell (AAS) as the basis for interoperability. AAS is the digital representation of an asset that is able to provide information about this asset, i.e. information about properties, functionality, parameters, documentation, etc. The AAS operates as Digital Twin of the asset it represents. Furthermore, the AAS covers all stages of the lifecycle of an asset starting in the development phase, reaching the most importance in the operation phase and finally delivering valuable information for the decline/decomposition phase.

To guarantee the interoperability of assets Industrie 4.0 defines an information metamodel for the AAS covering all important aspects as type/instance concept, events, predefined data specification templates, security aspects, mapping of data formats and many more. Moreover, interfaces and operations for a registry, a repository, publish and discovery are specified. At first glance the evolving specification of the AAS seems pretty complex and a challenging task for asset providers. To make things easier, FA³ST provides an implementation of several tools to allow for easy and fast creation and management of AAS-compliant Digital Twins.

Important: FA³ST is currently in the process of becoming an Eclipse project which will be finalized after releasing v1.0.0 here.

2.11.1 Contact

faaast@iosb.fraunhofer.de

2.11.2 License

2.12 Contributing

Contributions are what make the open source community such an amazing place to learn, inspire, and create. Any contributions are **greatly appreciated**.

If you have a suggestion for improvements, please fork the repo and create a pull request. You can also simply open an issue. Don't forget to rate the project! Thanks again!

1. Fork the Project
2. Create your Feature Branch (`git checkout -b feature/AmazingFeature`)
3. Commit your Changes (`git commit -m 'Add some AmazingFeature'`)
4. Push to the Branch (`git push origin feature/AmazingFeature`)
5. Open a Pull Request

2.12.1 Code Formatting

The project uses *spotless:check* in the build cycle, which means the project only compiles if all code, *.pom and *.xml files are formatted according to the project's codestyle definitions (see details on spotless). You can automatically format your code by running

```
mvn spotless:apply
```

Additionally, you can import the eclipse formatting rules defined in */codestyle* into our IDE.

2.12.2 Third Party License

If you use additional dependencies please be sure that the licenses of these dependencies are compliant with our License. If you are not sure which license your dependencies have, you can run

```
mvn license:aggregate-third-party-report
```

and check the generated report in the directory `docs/third_party_licenses_report.html`.

2.12.3 Contributors

Name	Github Account
Michael Jacoby	mjacoby
Jens Müller	JensMueller2709
Klaus Schick	schick64
Tino Bischoff	tbischoff2
Friedrich Volz	fvolz

2.13 Recommended Documents/Links

- Asset Administration Shell Specifications
  Quick-links To Different Versions & Reading Guide
- Details of the Asset Administration Shell - Part 1, Nov 2021
  The publication states how companies can use the Asset Administration Shell to compile and structure information. In this way all information can be shared as a package (set of files) with partners at several levels of the value chain. It is not necessary to provide online access to this data from the very beginning.
- Details of the Asset Administration Shell - Part 2, Nov 2021
  This part extends Part 1 and defines how information provided in the Asset Administration Shell (AAS) (e.g. submodels or properties) can be accessed dynamically via Application Programming Interfaces (APIs).
- About OPC UA
- OPC UA Companion Specification OPC UA for Asset Administration Shell (AAS)