FA^{3}ST Service

Fraunhofer IOSB

Nov 04, 2022

GETTING STARTED

The **F**raunhofer **A**dvanced **A**sset **A**dministration **S**hell **T**ools (**FA³ST**) Service implements the Asset Administration Shell (AAS) specification by Plattform Industrie 4.0 and provides an easy-to-use re-active AAS (Type 2) hosting custom AAS models.

CHAPTER

ONE

IMPLEMENTED AAS SPECIFICATIONS

Specification	Version
Details of the Asset Administration Shell - Part 1The exchange of information between partners in the value chain of Industrie 4.0	Version 3.0RC01(based on admin-shell-io/java-model)
Details of the Asset Administration Shell - Part 2Interoperability at Runtime – Exchanging Information via Application Programming Interfaces	Version 1.0RC02

CHAPTER

TWO

FEATURES

- supports several dataformats for the Asset Administration Shell Environment: `json`, `json-ld`, `xml`, `aml`, `rdf`, `opcua nodeset`
- easy configuration via JSON file
- easily expandable with 3rd party implementations for `endpoint`, `messagebus`, `persistence`, `assetconnection`
- uses existing open source implementation of AAS datamodel and de-/serializers admin-shell-io java serializer and admin-shell-io java model
- synchronization between multiple endpoints
- connecting to assets using arbitrary communication protocols
- can be used via command-line interface (CLI), as docker container or embedded library

2.1 Getting Started

2.1.1 Prerequisites

- Java 11+

2.1.2 Usage

From precompiled JAR

Download latest version as precompiled JAR

As Maven Dependency

```
<dependency>
        <groupId>de.fraunhofer.iosb.ilt.faaast.service</groupId>
        <artifactId>starter</artifactId>
        <version>0.3.0</version>
</dependency>
```

As Gradle Dependency

```
implementation 'de.fraunhofer.iosb.ilt.faaast.service:starter:0.3.0'
```

A maven plugin we are using in our build script leads to an error while resolving the dependency tree in gradle. Therefore you need to add following code snippet in your `build.gradle`. This code snippet removes the classifier of the transitive dependency `com.google.inject:guice`.

```
configurations.all {
        resolutionStrategy.eachDependency { DependencyResolveDetails details ->
                if (details.requested.module.toString() == "com.google.inject:guice") {
                        details.artifactSelection{
                                it.selectArtifact(DependencyArtifact.DEFAULT_TYPE, null,␣
↪null);
                        }
                }
        }
}
```

2.1.3 Building from Source

Prerequisites

- Maven

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

2.1.4 Example

This example shows how to start a FA³ST Service given your custom AAS model (called `model.json` for simplicity but can be any relative or absolute path to an AAS model in any supported data format, e.g. JSON, XML, RDF or AASX). The service will expose an HTTP endpoint on default port 8080.

Via Command-line Interface (CLI)

```
cd /starter/target
java -jar starter-{version}.jar -m model.json
```

From Code (embedded)

```
Service service = new Service(ServiceConfig.builder()
                .core(CoreConfig.builder()
                                .requestHandlerThreadPoolSize(2)
                                .build())
                .persistence(PersistenceInMemoryConfig.builder()
                                .environment(AASEnvironmentHelper
                                                .fromFile(new File("{pathTo}\\FAAAST-
↪Service\\misc\\examples\\demoAAS.json")))
                                .build())
                .endpoint(HttpEndpointConfig.builder().build())
                .messageBus(MessageBusInternalConfig.builder().build())
                .build());
service.start();
```

2.2 Usage with Command Line

To start a FA³ST Service from the command line:

1. Move to the starter project and build the project

```
cd /starter
mvn clean package
```

2. Move to the generated `.jar` file

```
cd starter/target
```

3. Execute the `.jar` file to start a FA³ST Service directly with a default configuration. Replace the `{path/to/your/AASEnvironment}` with your file to the Asset Administration Shell Environment you want to load with the FA³ST Service. If you just want to play around, you can use an example AASEnvironment from us here.

```
java -jar starter-{version}.jar -m {path/to/your/AASEnvironment}
```

Currently we supporting following formats of the Asset Administration Shell Environment model:

json, json-ld, aml, xml, opcua nodeset, rdf

Following command line parameters could be used:

```
[<String=String>...]                    Additional properties to override values of␣
↪configuration using
                                JSONPath notation without starting '$.' (see https://
↪goessner.net/articles/JsonPath/)

-c, --config=<configFile>           The config file path. Default Value = config.json

--emptyModel                          Starts the FA³ST service with an empty Asset␣
↪Administration Shell Environment.
                                False by default

--endpoint=<endpoints>[,<endpoints>...]
```

(continues on next page)

(continued from previous page)

```
                                Additional endpoints that should be started.

-h, --help                         Show this help message and exit.

-m, --model=<modelFile>            Asset Administration Shell Environment FilePath.
                                Default Value = aasenvironment.*

--[no-]autoCompleteConfig
                                Autocompletes the configuration with default
                                values for required configuration sections. True
                                by default

--[no-]modelValidation                 Validates the AAS Environment. True by default

-V, --version                      Print version information and exit.
```

2.2.1 Change the Configuration

In general there are 3 ways to configure your FA³ST Service:

1. Default values
2. Commandline parameters
3. Environment Variables

The 3 kinds can be combined, e.g. by using the default configuration and customizing with commandline parameters and environment variables. If they conflict, environment variables are preferred over all and commandline parameters are preferred over the default values.

Without any manual customization a FA³ST Service with default configuration will be started. For details to the structure and components of the configuration please have a look at the configuration section.

Default Configuration:

```
{
        "core" : {
                "requestHandlerThreadPoolSize" : 2
        },
        "endpoints" : [ {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.endpoint.http.
↪HttpEndpoint",
                "port" : 8080
        } ],
        "persistence" : {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.persistence.memory.
↪PersistenceInMemory"
        },
        "messageBus" : {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.messagebus.internal.
↪MessageBusInternal"
        }
}
```

The FA³ST Service Starter consider following environment variables:

- `faaast.config` to use a own configuration file
- `faaast.model` to use a Asset Administration Environment file

Environment variables could also be used to adjust some config components in the configuration. Therefore, we are using JSONPath notation without starting '$.' (see here) with the prefix `faaast.config.extension.`:

- `faaast.config.extension.[dot.separated.path]`

If you want to change for example the requestHandlerThreadPoolSize in the core configuration, just set the environment variable `faaast.config.extension.core.requestHandlerThreadPoolSize=42`. To access configuration components in a list use the index. For example to change the port of the HTTP endpoint in the default configuration you can set the environment variable `faaast.config.extension.endpoints[0].port=8081`.

You could also use properties to adjust configuration components. To change the `requestHandlerThreadPoolSize` of the core component and the port of the http endpoint use

```
java -jar starter-{version}.jar -m {path/to/your/AASEnvironment} core.
↪requestHandlerThreadPoolSize=42 endpoints[0].port=8081
```

2.2.2 Special Parameters

The parameter `--endpoint` accepts a list of endpoints which should be started with the service. Currently supported is `http` and `opcua`. So a execution of

```
java -jar starter-{version}.jar -m {path/to/your/AASEnvironment} --endoint http
```

leads to a FA³ST Service with the HTTP endpoint implemented in class `de.fraunhofer.iosb.ilt.faaast.service.endpoint.http.HttpEndpoint`.

2.3 Configuration

This section gives a short introduction how the configuration file works.

The basic structure of a configuration is the following

```
{
        "core" : {
                "requestHandlerThreadPoolSize" : "number"
        },
        "endpoints" : [
                // endpoint configurations, multiple allowed
        ],
        "persistence" : {
                // persistence configuration
        },
        "messageBus" : {
                // message bus configuration
        },
        "assetConnections": [
                // asset connection configurations, multiple allowed
        ]
}
```

As FA³ST is designed to be easily extendable, the configuration supports to change the used implementation for any of those interfaces without the need to change or recompile the code. To tell the Service which implementation of an interface to use, each dynamically configurable configuration block contains the `@class` node specifying the fully qualified name of the implementation class. Each block then contains additionals nodes as defined by the configuration class associated with the implementation class. For example, the `HttpEndpoint` defines the property `port` in its configuration class (HttpEndpointConfig.java#L23).

Therefore, the configuration block for a `HttpEndpoint` on port 8080 would look like this:

```
{
        "@class" : "de.fraunhofer.iosb.ilt.faaast.service.endpoint.http.HttpEndpoint",
        "port" : 8080
}
```

For FA³ST to 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 which you can achieve 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.

A simple example configuration could look like this:

```
{
        "core" : {
                "requestHandlerThreadPoolSize" : 2
        },
        "endpoints" : [ {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.endpoint.http.
↪HttpEndpoint",
                "port" : 8080
        } ],
        "persistence" : {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.persistence.memory.
↪PersistenceInMemory"
        },
        "messageBus" : {
                "@class" : "de.fraunhofer.iosb.ilt.faaast.service.messagebus.internal.
↪MessageBusInternal"
        }
}
```

Each implementation should provide documentation about supported configuration parameters. When using FA³ST Service from your code instead of running it in standalone mode, you can also create the configuration file manually like this:

```
ServiceConfig serviceConfig = new ServiceConfig.Builder()
        .core(CoreConfig.builder()
                        .requestHandlerThreadPoolSize(2)
                        .build())
        .persistence(PersistenceInMemoryConfig.builder().build())
        .endpoint(HttpEndpointConfig.builder().build())
        .messageBus(MessageBusInternalConfig.builder().build())
        .build();
```

2.4 Usage with Docker

This section describes the usage with docker and docker compose.

2.4.1 Docker-Compose

Clone this repository, navigate to `/misc/docker/` and run this command inside it.

```
cd misc/docker
docker-compose up
```

To use your own AAS environment replace the model file `/misc/examples/demoAAS.json`. To modify the configuration edit the file `/misc/examples/exampleConfiguration.json`. You can also override configuration values using environment variables. For details have a look into the commandline section.

2.4.2 Docker CLI

To start the FA³ST Service with an empty AAS environment execute this command.

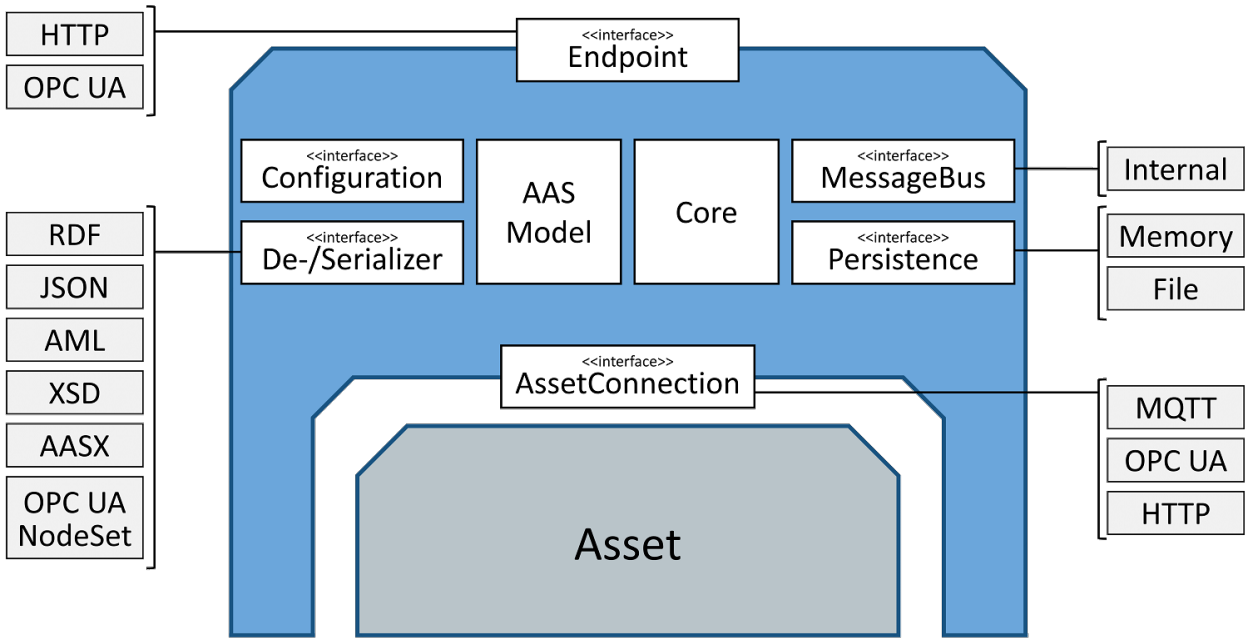
```
docker run --rm -P fraunhoferiosb/faaast-service '--emptyModel' '--no-modelValidation'
```

To start the FA³ST Service with your own AAS environment, place the JSON-file (in this example `demoAAS.json`) containing your enviroment in the current directory and modify the command accordingly.

```
docker run --rm -v ../examples/demoAAS.json:/AASEnv.json -e faaast.model=AASEnv.json -P␣
↪fraunhoferiosb/faaast-service '--no-modelValidation'
```

Similarly to the above examples you can pass more arguments to the FA³ST service by using the CLI or a configuration file as provided in the cfg folder (use the `faaast.config` environment variable for that).

2.5 Architecture



FA³ST Service uses an open architecture and defines interfaces for most functionality. This allows for easy extension by 3rd parties. However, FA³ST Service also includes one or more useful default implementations for each interface:

- HTTP Endpoint
- OPC UA Endpoint
- Internal Message Bus
- MQTT Asset Connection
- OPC UA Asset Connection

2.6 HTTP Endpoint

The HTTP Endpoint allows accessing data and execute operations within the FA³ST Service via REST-API. The HTTP Endpoint is based on the document Details of the Asset Administration Shell - Part 2, *Interoperability at Runtime – Exchanging Information via Application Programming Interfaces (Version 1.0RC02)*' , November 2021 and the OpenAPI documentation DotAAS Part 2 | HTTP/REST | Entire Interface Collection, Apr, 26th 2022

For detailed information on the REST API see DotAAS Part 2 | HTTP/REST | Entire Interface Collection, Apr, 26th 2022

In order to use the HTTP Endpoint the configuration settings require to include an HTTP Endpoint configuration, like the one below:

```
{
        "endpoints": [
                {
                        "@class": "de.fraunhofer.iosb.ilt.faaast.service.endpoint.http.
↪HttpEndpoint",
```

(continues on next page)

(continued from previous page)

```
                "port": 8080
            }
        ]
}
```

2.6.1 API

Supported API calls

- Asset Administration Shell Repository Interface
  - /shells
  - /shells/{aasIdentifier}
- Asset Administration Shell Interface
  - /shells/{aasIdentifier}/aas
  - /shells/{aasIdentifier}/aas/asset-information
  - /shells/{aasIdentifier}/aas/submodels
  - /shells/{aasIdentifier}/aas/submodels{submodeIdentifier}
- Submodel Repository Interface
  - /submodels
  - /submodels/{submodelIdentifier}
- Submodel Interface
  - /submodels/{submodelIdentifier}/submodel
  - /submodels/{submodelIdentifier}/submodel/submodel-elements
  - /submodels/{submodelIdentifier}/submodel/submodel-elements/{idShortPath}
  - /submodels/{submodelIdentifier}/submodel/submodel-elements/{idShortPath}/invoke
  - /submodels/{submodelIdentifier}/submodel/submodel-elements/{idShortPath}/operation-Results/{handle-Id}
- Submodel Interface (combined with Asset Administration Shell Interface)
  - /shells/{aasIdentifier}/aas/submodels/{submodelIdentifier}/submodel
  - /shells/{aasIdentifier}/aas/submodels/{submodelIdentifier}/submodel/submodel-elements
  - /shells/{aasIdentifier}/aas/submodels/{submodelIdentifier}/submodel/submodel-elements/{idShortPath} , ,
  - /shells/{aasIdentifier}/aas/submodels/{submodelIdentifier}/submodel/submodel-elements/{idShortPath}/invoke
  - /shells/{aasIdentifier}/aas/submodels/{submodelIdentifier}/submodel/submodel-elements/{idShortPath}/operation-Results/{handle-Id}
- Concept Description Repository Interface
  - /concept-descriptions
  - /concept-descriptions/{cdIdentifier}

- Asset Administration Shell Basic Discovery
  - /lookup/shells
  - /lookup/shells/{aasIdentifier}
- Asset Administration Shell Serialization Interface
  - /serialization

Optional query parameters

- level=deep|core
- content=normal|value|path
- extent=WithoutBLOBValue/WithBLOBValue
- InvokeOperation supports async=true/false

They are added to the URL as regular query params

```
http://url:port?level=deep&content=value
```

FA³ST Service currently supports only content=value and content=normal

Unsupported API calls

- Asset Administration Shell Registry Interface (out-of-scope)
- Submodel Registry Interface (out-of-scope)
- AASX File Server Interface (probably supported in future)
  - /packages
  - /packages/{packageId}

2.6.2 Example

Sample HTTP Call for Operation *GetSubmodelElementByPath* using the parameters

- *submodelIdentifier*: https://acplt.org/Test_Submodel (must be base64URL-encoded)
- *idShortPath*: ExampleRelationshipElement (must be URL-encoded)

using the query-parameters *level=deep* and *content=normal*.

> 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.

```
http://localhost:8080/submodels/aHR0cHM6Ly9hY3BsdC5vcmcvVGVzdF9TdWJtb2RlbA==/submodel/
↪submodel-elements/ExampleRelationshipElement?level=deep&content=normal
```

2.7 OPC UA Endpoint

The OPC UA Endpoint allows accessing data and execute operations within the FA³ST Service via OPC UA. For detailed information on OPC UA see About OPC UA

The OPC UA Endpoint is based on the OPC UA Companion Specification OPC UA for Asset Administration Shell (AAS). The release version of this Companion Specification is based on the document Details of the Asset Administration Shell - Part 1 Version 2.

This implementation is based on Details of the Asset Administration Shell - Part 1 Version 3, which is currently not yet released. Therefore, the current implementation is actually not compatible with the Companion Specification.

The OPC UA Endpoint is built with the Prosys OPC UA SDK for Java. If you want to build the OPC UA Endpoint, you need a valid license for the SDK.

You can purchase a Prosys OPC UA License. As the OPC UA Endpoint is a server, you need a "Client & Server" license.

For evaluation purposes, you also have the possibility to request an evaluation license.

In order to use the OPC UA Endpoint, the configuration settings require to include an OPC UA Endpoint configuration, like the one below:

```
{
        "endpoints": [
                {
                        "@class": "de.fraunhofer.iosb.ilt.faaast.service.endpoint.opcua.
↪OpcUaEndpoint",
                        "tcpPort" : 18123,
                        "secondsTillShutdown" : 5
                }
        ]
}
```

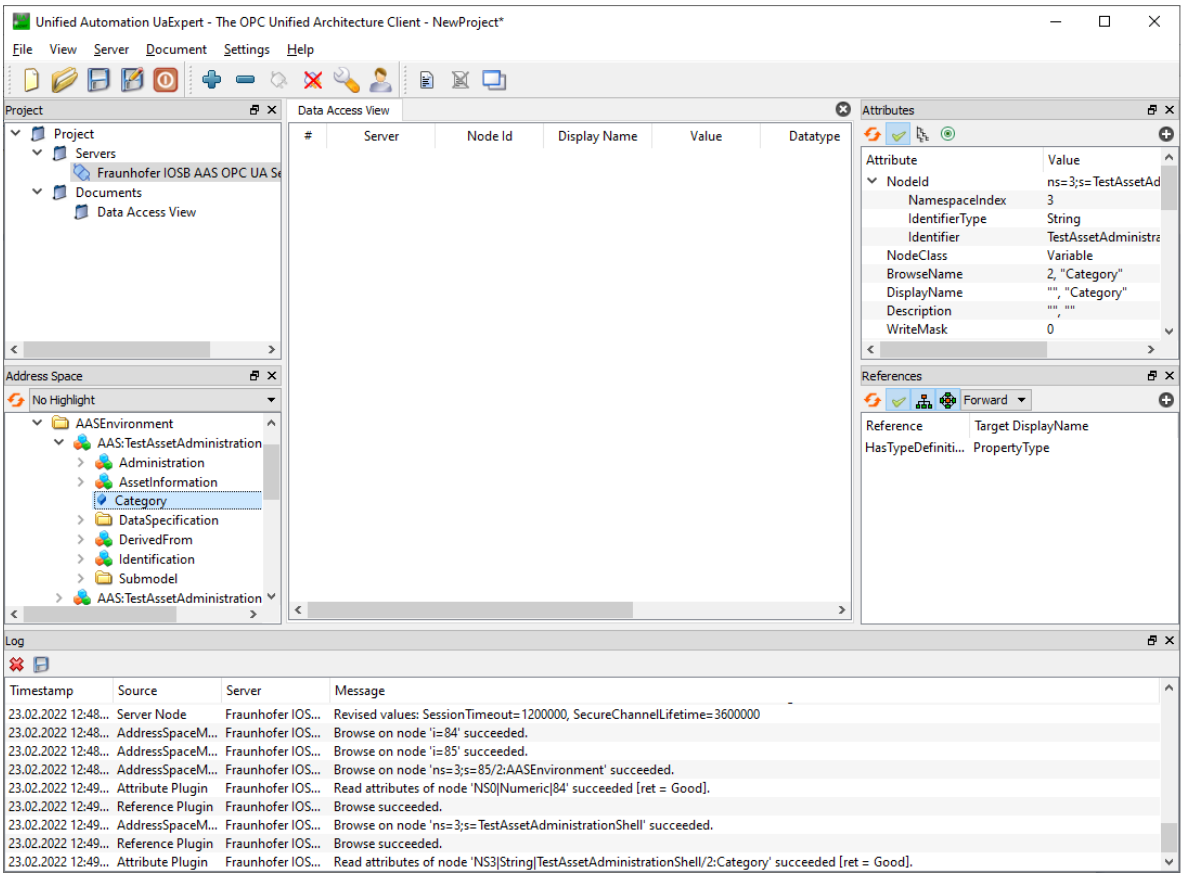
OPC UA Endpoint configuration supports the following configuration parameters

- `tcpPort` is the desired Port for the OPC UA TCP Protocol (opc.tcp). Default is 4840.
- `secondsTillShutdown` is the number of seconds the server waits for clients to disconnect when stopping the Endpoint. When the Endpoint is stopped, the server sends a predefined event to all connected clients, that the OPC UA Server is about to shutdown. Now, the OPC UA Server waits the given number of seconds before he stops, to give the clients the possibility to disconnect from the Server. When `secondsTillShutdown` is 0, the Endpoint doesn't wait and stops immediately.

To connect to the OPC UA Endpoint, you need an OPC UA Client. Here are some examples of OPC UA Clients:

- Unified Automation UaExpert UaExpert is a free test client for OPC UA. A registration for the website is required.
- Prosys OPC UA Browser Free Java-based OPC UA Client. A registration for the website is required.
- Official Samples from the OPC Foundation C#-based sample code from the OPC Foundation.
- Eclipse Milo Java-based Open Source SDK for Java.

Here you can see a sample Screenshot with UaExpert.



2.7.1 Supported Functions

- Operations (OPC UA method calls). Exception: Inoutput-Variables are not supported in OPC UA.
- Write Values
  - Property
  - Range
  - Blob
  - MultiLanguageProperty
  - ReferenceElement
  - RelationshipElement
  - Entity

2.7.2 Not (yet) Supported Functions

- Events
- Write Values
  - DataSpecifications
  - Qualifier
  - Category
  - ModelingKind
- AASValueTypeDataType
  - ByteString
  - Byte
  - UInt16
  - UInt32
  - UInt64
  - DateTime
  - LocalizedText
  - UtcTime

2.8 Asset Connections

`AssetConnection` implementations allows connecting/synchronizing elements of your AAS to/with assets via different protocol. This functionality is further divided into 3 so-called provider, namely

- ValueProvider: supporting reading and writing values from/to the asset, i.e. each time a value is read or written via an endpoint the request is forwarded to the asset
- OperationProvider: supporting the execution of operations, i.e. forwards operation invocation requests to the asset and returning the result value,
- SubscriptionProvider: supporting synchronizing the AAS with pub/sub-based assets, i.e. subscribes to the assets and updates the AAS with new values over time.

An implementation does not have to implement all providers, 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).

Each provider is connected to exactly one element of the AAS. Each asset connection can have multiples of each provider type. Each FA³ST Service can have multiple asset connections. Accordingly, each asset connection configuration supports at least this minimum structure

```
{
        "@class": "...",
        "valueProviders":
        {
                "{serialized Reference of AAS element}":
                {
                        // value provider configuration
```

(continues on next page)

(continued from previous page)

```
                }
        },
        "operationProviders":
        {
                "{serialized Reference of AAS element}":
                {
                        // operation provider configuration
                }
        },
        "subscriptionProviders":
        {
                "{serialized Reference of AAS element}":
                {
                        // subscription provider configuration
                }
        }
}
```

A concrete example for OPC UA asset connection could look like this

```
{
        "@class": "de.fraunhofer.iosb.ilt.faaast.service.assetconnection.opcua.
↪OpcUaAssetConnection",
        "host": "opc.tcp://localhost:4840",
        "valueProviders":
        {
                "(Submodel)[IRI]urn:aas:id:example:submodel:1,(Property)[ID_
↪SHORT]Property1":
                {
                        "nodeId": "some.node.id.property.1"
                },
                "(Submodel)[IRI]urn:aas:id:example:submodel:1,(Property)[ID_
↪SHORT]Property2":
                {
                        "nodeId": "some.node.id.property.2"
                }
        },
        "operationProviders":
        {
                "(Submodel)[IRI]urn:aas:id:example:submodel:1,(Operation)[ID_
↪SHORT]Operation1":
                {
                        "nodeId": "some.node.id.operation.1"
                }
        },
        "subscriptionProviders":
        {
                "(Submodel)[IRI]urn:aas:id:example:submodel:1,(Property)[ID_
↪SHORT]Property3":
                {
                        "nodeId": "some.node.id.property.3",
                        "interval": 1000
```

(continues on next page)

(continued from previous page)

```
                }
        }
}
```

2.9 HTTP Asset Connection

2.9.1 Supported Providers

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

2.9.2 Configuration Parameters

Asset Connection

Value Provider

Example

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

Operation Provider

Example

Operation with input parameters `in1` and `in2` and output parameters `out1` and `out2`

```
{
        "format": "JSON",
        "path": "/foo/execute",
        "headers": {
                "foo": "bar"
```

(continues on next page)

(continued from previous page)

```
        },
        "method": "POST",
        "template": "{\"input1\" : \"${in1}\", \"input2\" : \"${in2}\"}",
        "queries": {
                "out1": "$.output1",
                "out2": "$.output2"
        }
}
```

Subscription Provider

Example

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

2.10 MQTT Asset Connection

2.10.1 Supported Providers

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

2.10.2 Configuration Parameters

Asset Connection

Value Provider

Example

```
{
        "format": "JSON",
        "topic": "example/myTopic",
```

(continues on next page)

(continued from previous page)

```
        "template": "{\"foo\" : \"${value}\"}"
}
```

Subscription Provider

Example

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

2.11 OPC UA Asset Connection

2.11.1 Supported Providers

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

2.11.2 Configuration Parameters

Asset Connection

Value Provider

All NodeIds (also below) are specified in the ExpandedNodeId format (see OPC UA Reference, Part 6, Section ExpandedNodeId). In the following you can see two examples.

Example

```
{
        "nodeId": "nsu=com:example;s=foo"
}
```

or

```
{
        "nodeId": "ns=2;s=foo"
}
```

Operation Provider

Example

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

Subscription Provider

Example

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

2.11.3 Complete Example

A complete example for OPC UA asset connection could look like this

```
{
        "@class": "de.fraunhofer.iosb.ilt.faaast.service.assetconnection.opcua.
↪OpcUaAssetConnection",
        "host": "opc.tcp://localhost:4840",
        "valueProviders":
        {
                "(Submodel)[IRI]urn:aas:id:example:submodel:1,(Property)[ID_
↪SHORT]Property1":
                {
                        "nodeId": "some.node.id.property.1"
                },
                "(Submodel)[IRI]urn:aas:id:example:submodel:1,(Property)[ID_
↪SHORT]Property2":
```

(continues on next page)

(continued from previous page)

```
                {
                        "nodeId": "some.node.id.property.2"
                }
        },
        "operationProviders":
        {
                "(Submodel)[IRI]urn:aas:id:example:submodel:1,(Operation)[ID_
↪SHORT]Operation1":
                {
                        "nodeId": "some.node.id.operation.1"
                }
        },
        "subscriptionProviders":
        {
                "(Submodel)[IRI]urn:aas:id:example:submodel:1,(Property)[ID_
↪SHORT]Property3":
                {
                        "nodeId": "some.node.id.property.3",
                        "interval": 1000
                }
        }
}
```

2.12 Persistence

Each persistence configuration supports at least the following configuration parameters:

- `initialModel` (optional, can be overriden by CLI parameter or environment variable): Path to the AAS Environment model file
- `decoupleEnvironment` (optional, default: `true`): Only applicable if the AAS Environment is given as Java Object. If set to true, the persistence makes a deep copy of the AAS Environment and decouples the internal AAS Environment from the AAS Environment parsed on startup. If set to false, the same object instance is used in the FA³ST Service, which may have unexpected side effects.

Example of a persistence configuration:

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

2.13 In-Memory Persistence

The In-Memory Persistence keeps the AAS environment model parsed at startup in the local memory. Any change request, such as changing the value of a property, results in a change to the AAS environment model in the local memory. When the FA³ST Service is stopped, the changes to the AAS environment are lost.

The In Memory Persistence has no additional configuration parameters.

Not yet implemented:

- AASX Packages
- Package Descriptors
- SubmodelElementStructs

2.14 File-based Persistence

The file-based persistence keeps the entire AAS Environment in a model file which is stored at the local machine. Any change request, such as changing the value of a property, results in a change to the AAS environment model file. Thus, changes are stored permanently.

File Persistence configuration supports the following configuration parameters:

- `dataDir` (optional, default: `/`): Path under which the model file created by the persistence is to be saved
- `keepInitial` (optional, default: `true`): If false the model file parsed on startup will be overriden with changes. If true a copy of the model file will be created by the persistence which keeps the changes.
- `dataformat` (optional, default: same data format as input file): Determines the data format of the created file by file persistence. Ignored if the `keepInitial` parameter is set to false. Supported data formats are `JSON`, `XML`, `AML`, `RDF`, `AASX`, `JSONLD`, `UANODESET`.

Example configuration for the file persistence:

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

Not yet implemented:

- AASX Packages
- Package Descriptors
- SubmodelElementStructs

2.15 Example: Custom Asset Connection

You can find the full source code for this example at https://github.com/FraunhoferIOSB/FAAAST-Service/examples/assetcommectin-custom .

This page will be updated with a detailed description in the future.

2.16 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 Industie 4.0 defines an information metamodel for the AAS covering all important aspects as type/instance concept, events, redefined 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 easy and fast creation and management of AAS-compliant Digital Twins.

2.16.1 Roadmap

Next milestone is to release version 1.0.0 to Maven Central and DockerHub.

Some of the features we are working on include

- improve stability/robustness
- improve usability
- additional implementations of the persistence interface
  - file-based (✓)
  - database-backed
- support for additional APIs
  - Administration Shell Serialization Interface (✓)
  - AASX Server Interface

2.16.2 Contact

faaast@iosb.fraunhofer.de

2.16.3 License

2.17 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

2.17.1 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.

2.17.2 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`.

2.17.3 Contributors

2.18 Recommended Documents/Links

- Asset Administration Shell Specifications
  Quicklinks 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)

2.19 Changelog

2.19.1 Release version 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

2.19.2 Release version 0.2.1

Bugfixes

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

2.19.3 Release version 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

2.19.4 Release version 0.1.0

First release!