LIA-Python ASS description

Version 0.1, August 2022

LIA

from

Universität Magdeburg

Institut für Automatisierungstechnik (IFAT)

Lehrstuhl Integrierte Automation  
Universitätsplatz 2  
39016 Magdeburg

Table of content

[1 Purpose of the AAS 1](#_Toc111649251)

[2 AAS software description 1](#_Toc111649252)

[2.1 AAS functionality 1](#_Toc111649253)

[2.1.1 Architecture 1](#_Toc111649254)

[2.1.2 Interactions between modules 1](#_Toc111649255)

[2.1.3 Module interfaces 1](#_Toc111649256)

[2.1.4 Module description 1](#_Toc111649257)

[2.2 AAS UI 1](#_Toc111649258)

[2.3 AAS software structure 1](#_Toc111649259)

[2.3.1 AAS source code location 1](#_Toc111649260)

[2.3.2 Used tools and libraries including their versions 1](#_Toc111649261)

[3 AAS server generation 1](#_Toc111649262)

[3.1 AAS server generation inputs 1](#_Toc111649263)

[3.2 AAS server configuration 1](#_Toc111649264)

[3.3 AAS server generation outputs 2](#_Toc111649265)

[4 AAS usage 2](#_Toc111649266)

[4.1 AAS http/Rest interface 2](#_Toc111649267)

[4.2 AAS UI Visualisation 2](#_Toc111649268)

[5 Literature 2](#_Toc111649269)

Tables

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

Figures

Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.

Abbrevations

|  |  |
| --- | --- |
| AAS-ID | Asset Administration Shell Identifikation |
| C/S | Client Server |
| CfP | Call for Proposal |
| IP | Internet Protocol |
| JSON | JavaScript Object Notation |
| OPC UA | Open Plattform Communication Unified Architecture |
| Pub | Publisher |
| SP | Service Provider |
| SR | Service Requester |
| Sub | Subscriber |
| TCP | Transmission Control Protocol |
| UDP | User Datagram Protocol |
| VDE | Verband der Elektrotechnik Elektronik Informationstechnik |
| VDI | Verein Deutscher Ingenieure |
| LIA | Lehrstuhl Integrierte Automation |

# Purpose of the AAS

# AAS software description

## AAS functionality and structure

### Architecture

Class diagram

General principle:

* PyAAS class is the base module

The base module has a set of associated modules that extract the data from AASx \*.json file and provide the required functionalities.

### Environment Variables

The PythonAASServer requires some configuration variables, these should be provided in the .env file.

|  |  |
| --- | --- |
| **Variable Name** | **Description** |
| LIA\_AAS\_RESTAPI\_HOST | IP address for the rest api server |
| LIA\_AAS\_RESTAPI\_PORT | Port fort the rest api server |
| LIA\_AAS\_MQTT\_HOST | IP address of the external MQTT broker |
| LIA\_AAS\_MQTT\_PORT | Port of the external MQTT broker |
| LIA\_preferedI40EndPoint | Preferred endpoint for I4.0 communication (RESTAPI/MQTT) |
| LIA\_preferredCommunicationFormat | JSON / XML |
| LIA\_REGISTRYENDPOINT | Rest API endpoint of the Registry |
| LIA\_AAS\_PACKAGE | AASX \*.json file name |
| LIA\_PUBSUB\_LISTNER\_HOST | IP address of the PubSub Server |
| LIA\_PUBSUB\_LISTNER\_PORT | Port of the PubSub Server |

### AASx Package Data

The current version of the PythonAAS does not accept the .aasx, only the json serialization is permitted. The .aasx file should be extracted and placed into the **config** folder. The images, folder and other files should be placed under the under the directory structure config/aaasx/files.

Note : The directory structure /aasx/files should be maintained while adding the files to aasx package file the package explorer.

### Package descriptions

#### Base Module : PyAASServer

Package : src/main

File: pyaasServer.py

Github Link:

<https://github.com/harishpakala/PythonAASxServer/blob/main/src/main/pyaasServer.py>

Functions:

configure() Configures all the Link modules

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| configureLogger() | The server has its own Logging mechanism. Logs will be printed in the command line as well be appended in a separate file. The log files are stored in the folder src/main/ logs. |
| configureLogList() | Each of the skill has a separate log handler and as well a separate file. |
| configureExternalVariables() |  |
| configureInternalVariables() |  |
| configureAASConfigureParser() | The module that extracts and the parses the AASx \*.josn files is configured. |
| configureMsgHandler() | Message Handler is configured, it manages the communication between all the modules. It is the internal message bus of the server. |
| configureDataAdaptor() | Configure the Modules the creates and maintains the data strucutres required for storing the AAASX data |
| confiureDataManager() | Configures the DataManger, it synchronizes the read and write methods of the internal Data structures. |
| configureAASEndPoints() | Configure MQTT and HTTP endpoints of the Server. |
| configureAssetAccessPoints() | Configures the Asset Access Adaptors (opcua,aasServer) |
| configurePubSubManager() | Cofigures the pubsub manger of the server |
| configureScheduler() | Configures the scheduler, it is responsible for asset access related events |

start() starts all the Linked modules

* startDataManager()
* startPubSubManager()
* startAssetAccessPoints()
* startMsgHandlerThread()
* startScheduler()
* startSkills()

stop() stops all the linked modules

shutdown() shuts down the server

#### AASEndpointHandlers Module

**Package**: src/main/aasendpointhandlers

**Files**: mqtt\_endpointhandler.py (Mqtt Adaptor)

restapi\_endpointhandler.py (Rest Server)

rstapi\_endpointresources.py (Helper Classes for Rest Server)

**S ub-Package**: src/main/ aasendpointhandlers/templates

Contains all the html files responsible for server GUI

**GitHub Link** :

<https://github.com/harishpakala/PythonAASxServer/tree/main/src/main/aasendpointhandlers>

Each endpoint handler has to be inherited from the base class

<https://github.com/harishpakala/PythonAASxServer/blob/main/src/main/abstract/endpointhandler.py>

**Functions** :

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| configure() | Configures the endpoint handler. The ip is set to default ‘0.0.0.0’, it could be change to ‘localhost’. The port number should be provided in the .env file (LIA\_AAS\_RESTAPI\_PORT\_INTERN) |
| update() | Updates the configurations of the handler. |
| start() | Starts the Handler |
| stop() | Stops the Handler |
| dispatchMessage() | The handler should use this message to send the all the outbound I4.0 messages. The messageHandler module invokes this method. |
| retrieveMessage() | The handler should use this method to receive all the external I4.0 messages. These messages are to be pushed inbound queue maintained by the messageHandler module |

**Rest Server Handler**

Flask python package is used for creating the Rest Server

Example : rest namespace creation

drv\_rst\_api.add\_resource(ClassName,”NameSpace”)

For ever new namespace a separate Class is created and all the classes are created in the rstapi\_endpointresources.py file.

class ClassName(Resource):

def \_\_init\_\_(self):

pass

def get():

pass

def post():

pass

**Available Rest API’s**

|  |
| --- |
| <http://localhost:60011/shells> |
| <http://localhost:60011/concept-descriptions> |
| <http://localhost:60011/submodels> |
| [http://localhost:60011/shells/<path:aasIdentifier](http://localhost:60011/shells/%3cpath:aasIdentifier)> |
| [http://localhost:60011/concept-descriptions/<path:cdIdentifier](http://localhost:60011/concept-descriptions/%3cpath:cdIdentifier)> |
| [http://localhost:60011/submodels/<path:submodelIdentifier>/submodel](http://localhost:60011/submodels/%3cpath:submodelIdentifier%3e/submodel) |
| [http://localhost:60011/submodels/<path:submodelIdentifier>/submodel/submodel-elements/<path:idShortPath](http://localhost:60011/submodels/%3cpath:submodelIdentifier%3e/submodel/submodel-elements/%3cpath:idShortPath)> |
| [http://localhost:60011/shells/<path:aasIdentifier>/aas/submodels](http://localhost:60011/shells/%3cpath:aasIdentifier%3e/aas/submodels) |
| [http://localhost:60011/submodels/<path:submodelIdentifier>/submodel/submodel-elements](http://localhost:60011/submodels/%3cpath:submodelIdentifier%3e/submodel/submodel-elements) |
| [http://localhost:60011/shells/<path:aasIdentifier>/submodels/<path:submodelIdentifier>/submodel/submodel-elements/<path:idShortPath](http://localhost:60011/shells/%3cpath:aasIdentifier%3e/submodels/%3cpath:submodelIdentifier%3e/submodel/submodel-elements/%3cpath:idShortPath)> |
| [http://localhost:60011/shells/<path:aasIdentifier>/aas](http://localhost:60011/shells/%3cpath:aasIdentifier%3e/aas) |
| [http://localhost:60011/shells/<path:aasIdentifier>/aas/asset-information](http://localhost:60011/shells/%3cpath:aasIdentifier%3e/aas/asset-information) |

The JWT authentication-based source code should be included in the file.

In case of retrieving the I4.0 messages, a separate class RetrieveMessage is created.

**MQTT Handler**

Paho-mqtt python module is used for creating MQTT adaptor (client).

The server itself does not hosts an MQTT server, rather it uses an external MQTT broker.

The adaptor subscribes to the identification/id’s of all the AAS specified the AASx \*.josn file over the MQTT server. The MQTT server configurations should be provided in the .env file (LIA\_AAS\_MQTT\_HOST, LIA\_AAS\_MQTT\_PORT).

#### MessageHandler Module

**Package**: src/main/handlers

**Files**: messagehandler.py

**GitHub Link** :

https://github.com/harishpakala/PythonAASxServer/tree/main/src/main/handlers

**Class Name**: MessageHandler

* Message Handler is the message bus of the PythonAAS server, it maintains two queues and event listeners are attached both of them.
* The EndPoint Handlers need to getIbMessage method to push an inbound I4.0 messages into the message bus.
* The eventHandler sendOutBoundMessage invokes the dispatch method from the appropriate endpointhandler to send the outbound I4.0 messages.

|  |  |  |  |
| --- | --- | --- | --- |
| QueueName | Put | Get | Event Handler |
| **inBoundQueue** | putIbMessage | getIbMessage | receiveMessage |
| **outBoundQueue** | putObMessage | getIbMessage | sendOutBoundMessage |

#### DataManger Module

**Package**: src/main/datastore

**Files**: datamanager.py

**GitHub Link** :

<https://github.com/harishpakala/PythonAASxServer/blob/main/src/main/datastore/datamanager.py>

**Class Name :** DataManager

* Python AAS Server maintains internal data structures for representing AASX package data. Data Manger acts as an interface layer for this internal data structures.
* The Data structures have associated methods for read and write access.
* All the other modules of the server need to publish the respective read/write requester to DataManager.
* The Data Manager sequentially executes all the requests, for this it uses queue. All the requests are to be pushed to this queue.
* Each request is associated with a unique instance id and the responses are added to a dictionary.

|  |  |  |  |
| --- | --- | --- | --- |
| QueueName | Put | Get | Event Handler |
| **InBoundProcessingQueue** | pushInboundMessage | NA | requestHandler |

|  |  |  |
| --- | --- | --- |
| Dictionary | Key | Value |
| **outBoundProcessingDict** | InstanceID | Request response |

**Request :** {"data": {"\_shellId": aasId,"requestData":data},"method":"putAASShell"} [Write request]

{"data": aasId"method":"getAASShell"} [Read request]

Two utility classes ExecuteDBRetriever and ExecuteDBModifier are provided in the utils packages. The internal modules can utilize these classes for the access to the DataManger.

#### DataBaseServer module

**Package**: src/main/datastore

**Files**: databaseserver.py

**GitHub Link** :

https://github.com/harishpakala/PythonAASxServer/blob/main/src/main/datastore/databaseserver.py

**Class Name :** AAS\_Database\_Server

This class actually host all the data structures that capture the AASx package data.

It also provides required methods to operate upon these structures.

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| parseAssetAdministrationShells()  parseSubmodels()  processCollectionElements()  parseAssets()  parseConceptDescription() | Extract the all the relevant elements and creates an internal data structure representation |
| getSubmodel()  getAASShell()  getAssetInformation()  getSubmodelElement() | Returns the specific AAS meta model. The request should contain the specific the relevant identifier. For the sumodel element it should be idShortPath |
|  |  |
|  |  |
|  |  |
|  |  |

#### Utils Module

#### Config Package

<https://github.com/harishpakala/PythonAASxServer/blob/main/src/main/config/aasxconfig.py>

Input

Functions

#### Utils Package

## AAS UI

The http server is in

* <https://github.com/harishpakala/PythonAASxServer/blob/main/src/main/aasendpointhandlers/restapi_endpointhandler.py>

The html files are located in:

* <https://github.com/harishpakala/PythonAASxServer/tree/main/src/main/aasendpointhandlers/templates>

## AAS software structure

### AAS source code location

### Used tools and libraries including their versions

# AAS server generation

## AAS server generation inputs

Package Explorer higher than Version 3.0, AASX file

* One has to unzip the AASX file and place in <https://github.com/harishpakala/PythonAASxServer/tree/main/config/aasx/files> using the same structure as it is in AASX file

## AAS server configuration

## AAS server generation outputs

# AAS usage

## AAS http/Rest interface

## Storage during runtime

* All I4.0 language messages are stored in the following file:
  + <https://github.com/harishpakala/PythonAASxServer/blob/main/data/database.json>

## AAS UI Visualisation

# Literature

[1] blablabla