**RIC - Reference Implementation**

**The VWS vernetzt-Testbeds**

V1.0 RC01

Technical Report

15. July 2021

Universität Magdeburg

Fakultät für Elektrotechnik und Informationstechnik

Institut für Automatisierungstechnik

Postfach 4120, D-39016 Magdeburg

Germany

Index

[1 Abbreviation 2](#_Toc77245288)

[2 Relevant Documentation 2](#_Toc77245289)

[3 Introduction 3](#_Toc77245290)

[4 Registry Infrastructure Component (RIC) 3](#_Toc77245291)

[5 Interactions with RIC using the I4.0 language 5](#_Toc77245292)

[5.1 Registration 5](#_Toc77245293)

[5.2 HeartBeat 6](#_Toc77245294)

[6 RIC interface access 7](#_Toc77245295)

[6.1.1 RIC MQTT Server 7](#_Toc77245296)

[6.1.2 RIC HTTP Server 8](#_Toc77245297)

[6.1.3 RIC COAP Server 8](#_Toc77245298)

[7 RIC as Registry according to AASiD part 2 and http communication protocol 8](#_Toc77245299)

[8 Conclusion 11](#_Toc77245300)

# Abbreviation

AAS Asset Administration Shell

I4.0 Industrie 4.0

RIC Registry Infrastructure Component

AAS Asset Administration Shell

ALP Application Layer Protocol

RICSS RIC Smart Space

# Relevant Documentation

[1] VDI 2193 Blatt 1: Sprache für I4.0-Komponenten - Struktur von Nachrichten

[2] FIPA ACL Message Structure Specification (http://www.fipa.org/specs/fipa00061/SC00061G.html)

[3] AASiD Part 1: The exchange of information between partners in the value chain of Industrie 4.0 (Version 2.0.1)

[4] AASiD Part 2: Interoperability at Runtime – Exchanging Information via Application Programming Interfaces (V1.0RC01 for Review)

# Introduction

AASiD Part 1 [3] firstly introduces the term Asset Administration Shell as digital representation of a real-world physical asset. Secondly it presents a meta model for structuring the information about an asset in terms of Submodels and Submodel Elements (submodel collections, properties, operations, events, reference elements, files, and capability’s). The entire data structure representing an AAS can be packaged either in AASx container format or as plain JSON or XML formats and can be exchanged between the trading partners (Type 1 AAS).

An AAS being a digital representation of an asset, AASiD part 2 [4] specifies the standards for platform independent programming interfaces that AAS needs to support for making information about itself be accessible or available in the digital world (Type 2 AAS). The AASiD part 2 also introduces the concept of the registry that accepts, stores and provides information about any AAS. Along with the registry, AASiD part 2 defines the endpoint descriptor data structure for representing the information about the accessibility of an AAS in the digital world. An AAS would need to create its descriptor information and publish it to the registry, so that it could be made accessible to any other AAS or applications.

AASiD part 2 also mentions about Type 3 AAS that are expected to communicate with each other, probably to solve a real-world problem or to perform a specific task. A standard message format for exchange of information would reduce the complexity of interpreting it. VDI/VDE 2193 Part 1 and Part 2 provide guidelines for structuring the information and introduces a new standard that structures every message in-terms of frame and interactionElements. The frame part represents the sender and receiver information, message type, and interaction protocol information. Having a common message format, addresses the concerns of syntactic interoperability.

This document defines interaction protocol for registration of an AAS with the registry. An AAS could utilize this protocol for registering with the registry. In addition, this document defines the heartbeat interaction protocol so that registry can maintain active status of all the AAS that are associated with it.

# Registry Infrastructure Component (RIC)

This specification utilizes standards and concepts introduced in the AASiD part 1, part 2 and proposes an architecture for the infrastructure component termed as Registry Infrastructure Component (RIC). The RIC implements the registry concept (both AAS Registry and submodel registry) and also acts as a message broker (described in the part 2 of the RIC documentation). The RIC is a complex software component associated with a separate database and multiple application layer protocol (ALP) interfaces. Each interface (HTTP / COAP / MQTT) is implemented as a separate plugin that is comprised of a server and the related client. The RIC server implementation actively listens to all the inbound messages (only the I4.0 messages) over the three different interfaces, it categorizes them and internally assigns them a unique thread. Figure 1 shows an overview about the ALP interfaces, and relevant endpoints supported by the registry.

The Figure 2 presents the interaction between RIC and a set of two AAS, the AAS 1 registers itself with the RIC and AAS 2 requests the descriptor information of an AAS that is already registered with the RIC. The heartbeat interaction should happen only after the registration of the AAS is successfully completed.

The subsections 6.1 and 6.2 present in detail about these interaction protocols and also the composition of the relevant I4.0 messages.



Figure 1 Registry Overview



Figure 2 Interaction between AAS and the RIC

The chapter 5 details about the services provided the RIC as depicted in AASiD part 2 [3]. This services are available only over the http protocol and in JSON format. Chapter 6 describes the interaction between RIC and AAS using I4.0 message as described in VDI/VDE 2193. In this version of the document two semantic protocols are described.

# Interactions with RIC using the I4.0 language

RIC introduces two new interaction protocols registration and heartbeat. Each of the protocol consists of two message types, the message and its acknowledgement. The Heart Beat interaction should happen only after the registration of the AAS is successfully completed with the RIC. The subsections 6.1 and 6.2 presents in detail about these interaction protocols and also the composition of the relevant I4.0 messages.

## Registration

Every AAS firstly needs to get itself registered with the RIC. It needs to embed its shell descriptor (as defined in the AASiD detail part2) information within interactionELements part of the I4.0 message packet of type “register” and send it to the RIC. RIC firstly validates the I4.0 message, extracts the descriptor information and assigns it to an internal handler. The internal handler firstly validates the descriptor against the standard descriptor schema and returns a negative acknowledgement in case of malformed data. Otherwise, the handler inserts the descriptor into the database and returns a positive acknowledgment. In case of any other internal errors, the RIC would return a negative acknowledgment. Note: Currently, the registration using I4.0 message is possible only for the AAS Shell Descriptors.

Acknowledgment from the RIC will be an I4.0 message packet of type “registerack”. RIC embeds a submodel **StatusResponse** into the interactionElements part of the registerack message. StatusResponse submodel is a collection of three properties **Status**, **Code** and **Message,** values for these properties vary depending on the type of response. Table 1 lists down values for the submodel properties under different acknowledgements by the RIC for a register message.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Property Type** | **Descriptor**  **Invalid** | **Internal Error** | **Registration Success** | **Registration Success (Updated)** |
| Status | E | E | S | S |
| Code | 400 | 500 | 200 | 200 |
| Message | The syntax of the passed Asset Administration Shell descriptor is not valid or malformed request | Unexpected Internal Server Error | The Asset Administration Shell's registration was successful | The Asset Administration Shell's registration was successfully renewed |

Table 1 Values for StatusResponse Submodel Properties in case of registerack

Table 2 presents relevant values for the I4.0 frame properties in case of register, registerack. A point to note, **replyBy** and **replyTo** fields have one of the ALP RESTAPI(HTTP), MQTT or COAP. The replyTo indicates **“**the receiver AAS preferred ALP**”** and the replyBy property indicates **“**the ALP of the AAS that has created the message**”** (it indicates that the receiver AAS needs to send back the relevant message type as part of the interaction protocol involved over this ALP).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **I4.0 Element** | **Register I4.0 packet** | **RegisterAck I4.0 packet** | **HeartBeat I4.0 packet** | **HeartBeatAck I4.0 packet** |
| semanticProtocol/keys/type | GlobalReference | GlobalReference | GlobalReference | GlobalReference |
| semanticProtocol/keys/local | local | local | local | local |
| semanticProtocol/keys/value | www.admin-shell.io/interaction/registration | www.admin-shell.io/interaction/registration | www.admin-shell.io/interaction/heartbeat | www.admin-shell.io/interaction/heartbeat |
| semanticProtocol/keys/ idType | IRI | IRI | IRI | IRI |
| type | register | registerack | HeartBeat | HeartBeatAck |
| messageId | Set by the AAS | Set by the RIC | Set by the AAS | Set by the RIC |
| sender/identification/id | Global unique ID | VWS\_RIC | Global unique ID | VWS\_RIC |
| sender/identification/idType | Set by the AAS | idShort | Set by the AAS | idShort |
| sender/role/name | Register | RegistryHandler | AASHeartBeatHandler | HeartBeatHandler |
| receiver/identification/id | VWS\_RIC | Global unique ID | VWS\_RIC | Global unique ID |
| receiver/identification/idType | idShort | Set by the AAS | idShort | Set by the AAS |
| receiver/role/name | RegistryHandler | Register | HeartBeatHandler | AASHeartBeatHandler |
| replyBy | RESTAPI / MQTT / COAP | RESTAPI / MQTT / COAP | RESTAPI / MQTT / COAP | RESTAPI / MQTT / COAP |
| replyTo | RESTAPI / MQTT / COAP | RESTAPI / MQTT / COAP | RESTAPI / MQTT / COAP | RESTAPI / MQTT / COAP |
| conversationId | Set by the AAS | Same as register packet | Set by the AAS | Same as in the HeartBeat packet |
| interactionElements | AASiD part 2 Descriptor  See JSON example | Empty List | Empty List | Empty List / Status Submodel |

Table 2I4.0 message details for register, registerack, HeartBeat, HeartBeatAck

The **“**receiver/identification/id**”** and the **“**receiver/role/name**”** for register and registerack messages are to be adhered as specified in the Table 2. An AAS can choose one of the MQTT, RESTAPI or COAP protocols for publishing the message to the RIC, relevant technical details are provided in the RIC interface access.

## HeartBeat

An AAS can send an I4.0 message of type HeartBeat at regular intervals to RIC, however this is not mandated. Whenever the RIC receives a HeartBeat message, firstly it looks-up into the internal database to check whether the AAS is already registered or not.

If the AAS is not registered, it would inform the AAS to get its descriptors registered. In case the AAS is registered, its active status is updated with latest timestamp. The active status of all the AAS can be retrieved from the RIC. The Table 2 presents the list of relevant values for heartbeat, heartbeatack messages. Table 3 presents the values for StatusResponse submodel property elements in case the AAS is not registered.

|  |  |
| --- | --- |
| **Property Type** | **Registration Success (Updated)** |
| Status | S |
| Code | 200 |
| Message | The AAS is not registered, please provide descriptors |

Table 3 Values for StatusResponse submodel properties in case of HeartBeatAck

The Figure 3 presents the state machine of the RIC from the perspective of registry and heartbeat interactions. The complex internal mechanism is summarized into a simpler system so that it would be easy for the reader of this document to better understand the internal process. The registry message is firstly validated, next the descriptor within it is validated, late the descriptor it is stored into the internal database and lastly returning an appropriate acknowledgement.



Figure 3 State Machine from the point of view of RIC

# RIC interface access

RIC hosts different ALP plugins, it operates them parallelly and synchronizes their behaviour. A reference implementation of the RIC is hosted on the “**liabroker.ddns.net**” cloud, in this documentation the cloud space is termed as RIC smart space. The RIC smart space hosts the mosquito MQTT server and the RIC implementation itself hosts COAP and HTTP servers. The following sub-sections 6.3.1 to 6.3.3 present the technical details about the related servers.

### RIC MQTT Server

Server Host: liabroker.ddns.net

Server Port: 1883

The client application that is part of the RIC MQTT plugin subscribes to the Topic “**VWS\_RIC**”. An AAS that prefers MQTT as the ALP should subscribe to the topic name same as it’s identification/id (part of the AAS meta model), all the messages that are inbound to this AAS should be published to this topic.

An AAS that just supports MQTT as the ALP, would need to publish the register I4.0 message to the topic “**VWS\_RIC**”. The client application (as the part of the MQTT plugin), when it receives a new register message it allocates an internal handler over a separate thread to it. The Internal handler prepares the appropriate registerack message and submits to the mqtt client. The MQTT client then publishes the registerack message to the topic of the topic of the relevant AAS. Similarly, as AAS can publish its HeartBeat messages at regular intervals to the “**VWS\_RIC**” topic.

### RIC HTTP Server

Server Host: liabroker.ddns.net

Server Port: 9021

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | S.No | HTTP URI | Type | Description |
| Status | 1 | <http://liabroker.ddns.net:9021/status> | GET | Retrieve status of all the connected AAS |
| Communication | 2 | <http://liabroker.ddns.net:9021/i40commu> | POST | Post an I4.0 message packet to RIC |

The reference implementation of RIC hosts the HTTP server over the port 9021, this HTTP server provides multiple namespaces. The Table 6 present the list of URI along with the relevant rest operation. The status URI provides the active status of all the AAS agent at a particular instant in time for a GET request.

Table 4 RIC HTTP URI’s

An AAS can post its register I4.0 message packet over the communication URI. The server application part of the RIC HTTP plugin, accepts the inbound message and allocates it an internal handler. Once the internal handler prepares and submits the registerack message, the server application sends back the synchronous response. Similarly the AAS can post the heartbeat messages over the communication URI.

### RIC COAP Server

Server Host: liabroker.ddns.net

Server Port: 50683

Similar to HTTP server, reference implementation of RIC hosts the coap server on port 50683. An AAS can post the register and heartbeat messages to the communication URI.

|  |  |  |  |
| --- | --- | --- | --- |
| Name | HTTP URI | Type | Description |
| Communication | [coap://liabroker.ddns.net:50683/i40commu](http://liabroker.ddns.net:9021/i40commu) | POST | Post an I4.0 message packet to RIC |

Table 5 RIC COAP URI’s

# RIC as Registry according to AASiD part 2 and http communication protocol

The RIC implements a registry service (AAS Registry Service and the Submodel Registry Service) as specified in the AASiD part 2. The Table 6 list all the HTTP URI’s that the reference implementation of RIC provides. The table also lists the type of rest services attributed with each of the URI. The Table 7, Table 8, Table 9, Table 10, presents the appropriate responses for each of the service associated with all the HTTP URI’s.

Note 1:

{aas-identifier} = idShort or global unique identifier of AAS  
{submodel-identifier} = idShort or global unique identifier of Submodel

|  |  |  |  |
| --- | --- | --- | --- |
| Operation Name | HTTP URI | Type | Description |
| GetAllAssetAdministrationShellDescriptors | <http://liabroker.ddns.net:9021/registry/shellDescriptors> | GET | Returns all Asset Administration Shell Descriptors |
| GetAssetAdministrationShellDescriptorById | [http://liabroker.ddns.net:9021/registry/shellDescriptors/{aas-identifier](http://liabroker.ddns.net:9021/registry/shellDescriptors/%7baas-identifier)} | GET | Returns a specific Asset Administration Shell Descriptor |
| PutAssetAdministrationShellDescriptorById | [http://liabroker.ddns.net:9021/registry/shellDescriptors/{aas-identifier](http://liabroker.ddns.net:9021/registry/shellDescriptors/%7baas-identifier)} | PUT | Creates a new or updates an existing Asset Administration Shell Descriptor |
| DeleteAssetAdministrationShellDescriptorById | [http://liabroker.ddns.net:9021/registry/shellDescriptors/{aas-identifier](http://liabroker.ddns.net:9021/registry/shellDescriptors/%7baas-identifier)} | DELETE | Deletes an Asset Administration Shell Descriptor |
| GetAllSubmodelDescriptorsByAASid | [http://liabroker.ddns.net:9021/registry/shellDescriptors/{aas-identifier](http://liabroker.ddns.net:9021/registry/shellDescriptors/%7baas-identifier)}/submodelDescriptors | GET | Returns all submodel descriptors of a specific AAS Shell Descriptor |
|  |  |  |  |
| GetAllSubmodelDescriptors | <http://liabroker.ddns.net:9021/registry/submodelDescriptors> | GET | Returns all submodel descriptors |
| GetSubmodelDescriptorById | <http://liabroker.ddns.net:9021/registry/submodelDescriptors>/{submodel-identifier} | GET | Returns a specific submodel descriptor |
| PutSubmodelDescriptorById | [http://liabroker.ddns.net:9021/registry/submodelDescriptors](http://liabroker.ddns.net:9021/api/v1/registry/%7baasId%7d/submodels)/{submodel-identifier} | PUT | Creates a new or updates an existing submodel descriptor |
| DeleteSubmodelDescriptorById | [http://liabroker.ddns.net:9021/registry/submodelDescriptors](http://liabroker.ddns.net:9021/api/v1/registry/%7baasId%7d/submodels)/{submodelIdentifier} | DELETE | Creates a new or updates an existing submodel descriptor |
|  |  |  |  |
| GetShellDescriptorSchema | http://liabroker.ddns.net:9021/descriptor/shellDescriptor | GET | Returns the JSON schema for AAS Shell Descriptor |
| GetSubmodelDescriptorSchema | http://liabroker.ddns.net:9021/descriptor/submodelDescriptor | GET | Returns the JSON schema for Submodel Shell Descriptor |

Table 6 RIC namespaces list for AAS Descriptor Information as per AASiD part 2

HTTP responses for different REST interfaces

1. [http://liabroker.ddns.net:9021/registry/shellDescriptors](http://liabroker.ddns.net:9021/api/v1/registry)

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Condition | Response Message | Status Code |
| GET | Registry contains at least one entry | Returns a json object containing all the AAS shell descriptors. | 200 |
| GET | No entries | No Asset Administration Shell descriptors are yet registered | 200 |
| GET | Internal Error | Unexpected Internal Server Error | 500 |

Table 7 HTTP Responses for listing the entire Registry

1. [http://liabroker.ddns.net:9021/registry/shellDescriptors/{aas-identifier](http://liabroker.ddns.net:9021/registry/shellDescriptors/%7baas-identifier)}

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Condition | Response Message | Status Code |
| GET | Relevant descriptor is present | Returns a json object of the relevant AAS descriptor | 200 |
| GET | Relevant descriptor is not present | No Asset Administration Shell with passed identifier found | 200 |
| PUT | Relevant descriptor is not present | The Asset Administration Shell's descriptor registration is successfull | 200 |
| PUT | Relevant descriptor is already present | The Asset Administration Shell's descriptor registration is successfully renewed | 200 |
| PUT | The descriptor not well formed | The syntax of the Shell descriptor is not valid or malformed request | 500 |
| PUT | AAS ID present in the descriptor and URI do not match | The aas-identifier in the uri and in descriptor do not match | 500 |
| DELETE | Relevant AAS descriptor is not present | No Asset Administration Shell with passed identifier found | 200 |
| DELETE | Relevant AAS descriptor is already present | The Asset Administration Shell Descriptor was deleted successfully | 200 |
| GET / PUT /  DELETE | Internal Error | Unexpected Internal Server Error | 500 |

Table 8 HTTP Responses for different REST services pertaining to one AAS

1. [http://liabroker.ddns.net:9021/registry/shellDescriptors/{aas-identifier}/submodelDescriptors](http://liabroker.ddns.net:9021/registry/shellDescriptors/%7baas-identifier%7d/submodelDescriptors)

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Condition | Response Message | Status Code |
| GET | Registry contains the relevant entry | Returns a json object consisting submodel descriptors of the relevant AAS descriptor | 200 |
| GET | Relevant AAS descriptor is not present | No Asset Administration Shell with passed identifier found | 200 |
| GET | Internal Error | Unexpected Internal Server Error | 500 |

Table 9 HTTP Responses for listing Submodel Descriptors of a specific AAS

1. [http://liabroker.ddns.net:9021/registry/submodelDescriptors](http://liabroker.ddns.net:9021/api/v1/registry/%7baasId%7d/submodels)

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Condition | Response Message | Status Code |
| GET | Registry contains the relevant entry | Returns a json object containing all the AAS submodel descriptors. | 200 |
| GET | Relevant AAS descriptor is not present | No submodel shell descriptors are yet registered | 200 |
| GET | Internal Error | Unexpected Internal Server Error | 500 |

Table 10 Responses for list all the Submodel Descriptors

1. [http://liabroker.ddns.net:9021/registry/submodelDescriptors](http://liabroker.ddns.net:9021/api/v1/registry/%7baasId%7d/submodels)/{submodelIdentifier}

|  |  |  |  |
| --- | --- | --- | --- |
| Operation | Condition | Response Message | Status Code |
| GET | Relevant submodel descriptor is found | Returns a json object consisting the relevant submodel descriptor | 200 |
| GET | Relevant submodel descriptor is not present | No submodel descriptor with passed identifier found | 200 |
| PUT | Relevant submodel descriptor is already present | The submodel descriptor is successfully renewed | 200 |
| PUT | Relevant submodel descriptor is not present | The submodel descriptor is successfully registered | 200 |
| PUT | Submodel descriptor not well formed | The syntax of the passed submodel descriptor is not valid or malformed request | 500 |
| PUT | SubmodelId present in the descriptor and the id passed in the uri does not match | The submodel-identifier in the uri and in descriptor do not match | 500 |
| DELETE | Relevant submodel is not present | The submodel descriptor with the passed identifier not found | 200 |
| DELETE | Relevant submodel descriptor is present | The submodel Descriptor was successfully unregistered | 200 |
| GET / PUT /  DELETE | Internal Error | Unexpected Internal Server Error | 500 |

Table 11 Responses related different rest operations associated with one submodel descriptor

# Conclusion

This document outlines technical specifications of the reference implementation of RIC and it is only from perspective of RIC as a registry. An RIC also acts a medium for exchange of I4.0 message packets between any two AASs, another document is published that details how an AAS can utilize the services of RIC for delivering I4.0 messages to another AAS implement a different application layer protocol.