

Interface Design Description (IDD) for DataManager over REST



**Abstract**

This document describes for the Interface Design Description (IDD) of the Arrowhead DataManager service’s interfaces.

An Interface Design Description provides a detailed description of how the service is implemented/realized by using the Communication Profile and the chosen technologies.

This document outlines interfaces, message formats, metadata, and other important information to be able to use the DataManager system’s interfaces.

1. Interface Design Description Overview 3

2. Service Interfaces 3

2.1. Interface 1: Echo 3

2.1.1. Information Model 4

2.1.2. Error handling 4

2.1.3. Interaction with consumers 4

2.2. Interface 2: Orchestration 4

2.2.1. Information Model 5

2.2.2. Error handling 6

2.2.3. Interaction with consumers 7

2.3. Interface 3: Start store Orchestration by ID 7

2.3.1. Information Model 8

2.3.2. Error handling 9

3. Security 9

3.1. Certificates 9

3.2. Payload protection 9

4. References 9

5. Revision history 10

5.1. Amendments 10

5.2. Quality Assurance 10

6. Appendixes 10

## Interface Design Description Overview

This section contains pointers to Service Description (SD) documents.

Table 1 Pointers to SD documents

|  |  |
| --- | --- |
| **Service description** | **Path** |
| Echo service | tbd |
| Historian service | tbd |
| Proxy service | tbd |

This document describes how to utilize the Orchestrator system’s Orchestration service. Other services, such as the Management service, is not provided over MQTT.

* Protocol: HTTP
* Encoding: JSON
* Compression: none
* Security: Optionally using TLS and X.509 certificates (server and client)

## Services

## Service 1: **Echo**

Below are the specifics of this interface:

* The data model is plain text.
* No ontologies are in use.
* No schemas are currently defined.
* No payload encryption is used.

Table 4 Function description

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Service** | **Method** | **Input** | **Output** |
| Echo | Echo | GET | - | String |

## Echo: Information Model

The information for Echo is very basic. There is no input, and only plain text output, the string “Got it”.

## Echo: Parameters

This interface does not take any query path parameters.

## Echo: Response codes

|  |  |  |
| --- | --- | --- |
| Code | **Meaning** | **Comment** |
| 200 | Successful request |  |
| 401 | Unauthorized |  |
| 500 | Internal server error | In case of database errors etc. |

## Error handling

There is no error handling for the Echo interface, except the different response codes.

## Interaction with consumers

Echo only supports read operations, where the response is always a string “Got it”. This can be used to test if a system is actually running. No authorization is needed.



Figure 1: Echo interface

## Service 2: **Historian**

Below are the specifics of this interface:

* The data model is JSON.
* No ontologies are in use.
* No schemas are currently defined.
* No payload encryption is used.

Table 4 Function description

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Service** | **Method** | **Input** | **Output** |
| ListSystems | Historian | POST |  | DataManagerSystems |
| GetData | Historian | GET | systemName and serviceName | Sensor data |
| StoreData | Historian | PUT | systemName , serviceName, plus Sensor data | Result code |

## ListSystems: Information Model

In order to get a list of endpoints, a GET request must be sent to the corresponding URI. to the /orchestration endpoint. The response upon success is a Orchestration Response.

### Output: Example DataManagerSystems response

{

"systems": [“temperatureSys1”, “humiditySys2”, “humiditySys3”],

}

## ListSystems: Parameters

This interface does not take any query path parameters.

## ListSystems: Response codes

|  |  |  |
| --- | --- | --- |
| Code | **Meaning** | **Comment** |
| 200 | Successful request |  |
|  |  |  |

## ListSystems: Error handling

If the request was successful, a Orchestration Response is returned with a response code of 200. If an error occurs, for example due to an incorrectly formatted request, an error message is returned with the reason.

## ListSystems: Interaction with consumers

Figure 2 shows how a client must perform an orchestration operation.



Figure 2: Historian LiStSystems operation

## GetData: Information Model

In order to get data from an endpoint, a GET request must be sent to the corresponding URI. to the /historian/<systemName>/<serviceName> endpoint. The response upon success is a SenML formatted Response.

### Output: Example DataManagerSystems response

{

"systems": [“temperatureSys1”, “humiditySys2”, “humiditySys3”],

}

## GetData: Parameters

This interface does not take any query path parameters.

## GetData: Response codes

|  |  |
| --- | --- |
| Code | **Meaning** |
| 200 | Successful request |
|  |  |

## GetData: Error handling

If the request was successful, a Orchestration Response is returned with a response code of 200. If an error occurs, for example due to an incorrectly formatted request, an error message is returned with the reason.

## GetData: Interaction with consumers

Figure 2 shows how a client must perform an orchestration operation.



Figure 4: Historian GETDATA operation

## PutData: Information Model

In order to store data at an endpoint, a PUT request must be sent to the corresponding URI, to the /historian/<systemName>/<serviceName> endpoint. If the SenML encoded payload is OK, a 200 status code is returned. If the SenML message contains errors, an error is returned. For Example: to store two ball bearing temperatures (outer and inner) to the temperature service of the system ballBearingMonitor-342, perform a PUT to https://10.0.0.46:8461/datamanager/ballBearingMonitor-342/temperature with the payload below. Content-type must be set to “application/json”.

### Input: Example PutData request

[

{"bn": "temperature”, "bt": 1593759331, "bu": "Cel"}

{"n"; "bearingTempInner”, "v": 42.1},

{"n"; "bearingTempOuter”, "v": 34.5}  
]

## PutData: Parameters

This interface does not take any query path parameters.

## PutData: Response codes

|  |  |
| --- | --- |
| Code | **Meaning** |
| 200 | Successful request |
|  |  |

## PutData: Error handling

If the request was successful, a Orchestration Response is returned with a response code of 200. If an error occurs, for example due to an incorrectly formatted request, an error message is returned with the reason.

## PutData: Interaction with consumers

Figure 2 shows how a client can store data at a service endpoint.



Figure 4: Historian PUTDATA operation

## Service 3: **Proxy**

Below are the specifics of this interface:

* The data model is JSON.
* Supported semantics is SenML
* No ontologies are in use.
* No schemas are currently defined.
* No payload encryption is used.

Table 5 Function description

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Function** | **Service** | **Method** | **Input** | **Output** |
| Start store Orchestration by ID | Service Discovery | GET | id | Orchestration Response |

## Information Model

In order to start an orchestration with a known id, a client can perform a GET to /orchestrator/orchestration/{id}. The id is sent as a query parameter The response upon success is an updated ServiceRegistryEntry message with all fields filled in.

### Output: Orchestration Response message

{

"response": [

{

"provider": {

"id": 0,

"systemName": "string",

"address": "string",

"port": 0,

"authenticationInfo": "string",

"createdAt": "string",

"updatedAt": "string"

},

"service": {

"id": 0,

"serviceDefinition": "string",

"createdAt": "string",

"updatedAt": "string"

},

"serviceUri": "string",

"secure": "TOKEN",

"metadata": {

"additionalProp1": "string",

"additionalProp2": "string",

"additionalProp3": "string"

},

"interfaces": [

{

"id": 0,

"createdAt": "string",

"interfaceName": "string",

"updatedAt": "string"

}

],

"version": 0,

"authorizationTokens": {

"interfaceName1": "token1",

"interfaceName2": "token2"

},

"warnings": [

"FROM\_OTHER\_CLOUD", "TTL\_UNKNOWN"

]

}

]

}

## Error handling

If the request was successful, an Orchestration Response message is returned inside a REST-over-MQTT encapsulation message, with a response code of 200. If an error occurs, for example due to an incorrectly formatted request, an error message is returned with the reason.

## Security

This service can either run unencrypted over HTTP, or using TLS plus server and client side X509 certificates.

## Certificates

This IDD is using the same certificates as provided by the Java Spring versions.

## Payload protection

Currently, no separate payload protection is supported.

## References

1. …

## Revision history

## Amendments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Date | Version | Subject of Amendments | Author |
| 1 | 2015-02-15 | 1.0 | Revision of text | Michele Albano / Luis Ferreira |
| 2 | 2015-09-30 | 1.1 | Refinement of the structure | Michele Albano / Luis Ferreira |
| 3 | 2020-06-07 | 2.0 | Major update | Jerker Delsing |
| 4 | 2020-06-29 | 2.1 | Added DataManager text | Jens Eliasson |
| 5 | 2020-07-01 | 2.2 | Added text, errors etc | Jens Eliasson |
| 6 |  |  |  |  |

## Quality Assurance

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Date | Version | Approved by |
| 1 |  |  |  |
| 2 |  |  |  |

## Appendixes

Appendix A: REST Communication profile (CP)