# 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

This document describes how to utilize the DataManager system's Echo, Historian and Proxy services.  Generic settings, unless otherwise noted per service, are;

• Protocol:  HTTP(S)

• Encoding: JSON

• Compression: none

• Security: Optionally using TLS and X.509 certificates (server and client)

• Base path: The path /datamanager must prepended before the individual service's own paths. For example, the Echo service is accessed at http(s)://<IP>:<PORT>/datamanager/echo

# 2. Services

The DataManager, being a part of the Arrowhead Framework [1], provides three services; **Echo**, **Historian**, and **Proxy**. The Echo service is used to check the availability of the DataManager system. The Historian service provides methods for storage and retrieval of SenML encoded sensor data for a database. The Proxy service provides a mail box like feature for sleepy devices to store one message per endpoint so that consumers can access the data even if the sleepy device is offline.

## 2.1. 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 1 Function description

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

### 2.1.1. Echo: Information Model

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

### 1.1.1.Echo: Parameters

This interface does not take any query path parameters.

### 1.1.2. Echo: Response codes

| Code | Meaning | Comment |
|------|---------|---------|
| 200 | Successful request | Success |
| 401 | Unauthorized | Access denied |

### 1.1.3. Error handling

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

### 1.1.4. Interaction with consumers

Echo only supports read operations, where the response is always a string "*Got it*". This can be used to test if the DataManager system is actually running. No authorization is needed.
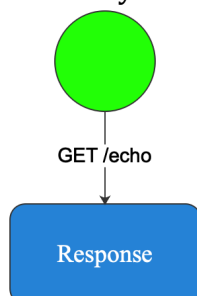
**FIGURE 1: ECHO INTERFACE**

## 2.2.  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 2 Function description

| Function | Service | Method | Input | Output |
|----------|---------|--------|-------|--------|
|  |  |  |  |  |

| ListSystems | Historian | GET | - | DataManager Systems |
|---|---|---|---|---|
| ListServices | Historian | GET | systemName | DataManager Services |
| GetData | Historian | GET | systemName, serviceName | Sensor data |
| PutData | Historian | PUT | systemName, serviceName, plus Sensor data | Result code |

# 2.2.1. ListSystems: Information Model

In order to get a list of endpoints, a GET request must be sent to the /historian endpoint. The response upon success is a list of systems using the Historian service.

Output: Example DataManagerSystems response

```
{
 "systems": ["temperatureSys1", "humiditySys2", "humiditySys3"]
}
```

# 1.1.5. ListSystems: Parameters

This interface does not take any query path parameters.

# 1.1.6. ListSystems: Response codes

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

# 1.1.7. ListSystems: Error handling

All errors are handled using HTTP response codes, see above. An error message is also added in the response payload.

# 1.1.8. ListSystems: Interaction with consumers

Figure 2 shows how a client can perform a ListSystems operation.
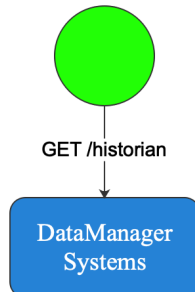


**FIGURE 2: HISTORIAN LISTSYSTEMS OPERATION**

# 2.2.2. ListServices: Information Model

In order to get a list of service endpoints, a GET request must be sent to the corresponding URI. to the /historian/<systemName> endpoint. The response upon success is an list of all service endpoints.

Output: Example DataManagerServices response
```
{
 "services": ["temperature", "humidity"]
}
```

# 1.1.9. ListServices: Parameters

This interface does not take any query path parameters.

# 1.1.10. ListServices: Response codes

| Code | Meaning | Comment |
|------|---------|---------|
| 200 | Successful request | Success |
| 400 | Bad request | If an incorrect parameter is used |
| 401 | Unauthorized | Access denied |
| 500 | Internal server error | In case of database errors etc. |

# 1.1.11. ListServices: Error handling

All errors are handled using HTTP response codes, see above. An error message is also added in the response payload.

# 1.1.12. ListServices:  Interaction with consumers

Figure 2 shows how a client must perform a ListServices operation.



**FIGURE 4: HISTORIAN LISTSERVICES OPERATION**

# 2.2.3. 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 SenML response

```
[
   {"bn": "temperature", "bt": 1593759331, "bu": "Cel"},
   {"n"; "bearingTempInner", "v": 42.1},
   {"n"; "bearingTempOuter", "v": 34.5}
]
```

# 1.1.13. GetData: Parameters

This interface takes the following query path parameters:

| Parameter | Usage | Example |
|---|---|---|
| count | To limit the number of returned values | count=10 will return the 10 newest values. |
| sigX | Is used to select only certain signals. First signal is indicated with sig0, the second with sig1, etc. | sig0=temperature&sig1=humidty will only return two signals named humidity and humidity. |
| sigXcount | Is used to limit the number of returned values per signal. | sig0=temperature&sig0count=10 will return the 10 latest values for the signal temperature. |

# 1.1.14. GetData: Response codes

| Code | Meaning | Comment |
|------|---------|---------|
| 200 | Successful request | Success |
| 400 | Bad request | If the request contains incorrect parameters |
| 401 | Unauthorized | Access denied |
| 404 | Not found | If the requested system and service combination does not exist |
| 500 | Internal server error | If a database error occurs |

# 1.1.15. GetData: Error handling

If the request was successful, a SenML message 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.

# 1.1.16. GetData:  Interaction with consumers

Figure 2 shows how a client must perform a GetData operation.

GET /historian/<systemName>/<serviceName>

SensorData

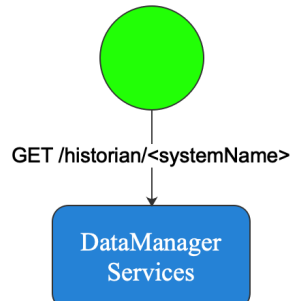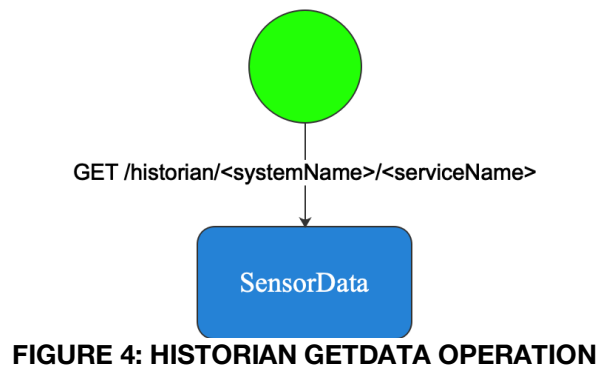**FIGURE 4: HISTORIAN GETDATA OPERATION**

# 2.2.4. 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 *ballBearingSensor-342*, perform a PUT to *https://10.0.0.46:8461/datamanager/historian/ballBearingSensor-342/temperature* with the payload below. Content-type must be set to "application/json".

Input: Example PutData SenML request

```
[
  {"bn": "temperature", "bt": 1593759331, "bu": "Cel"},
  {"n": "bearingTempInner", "v": 42.1},
  {"n": "bearingTempOuter", "v": 34.5}
]
```

# 1.1.17. PutData: Parameters

This interface does not take any query path parameters.

# 1.1.18. PutData: Response codes

| Code | Meaning | Comment |
|------|---------|---------|
| 200 | Successful request | Success |
| 400 | Bad request | Illegal or incorrect payload |
| 401 | Unauthorized | Access denied |
| 500 | Internal server error | If a database error occurs |

# 1.1.19. PutData: Error handling

If the request was successful, 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.

# 1.1.20. PutData: Interaction with consumers

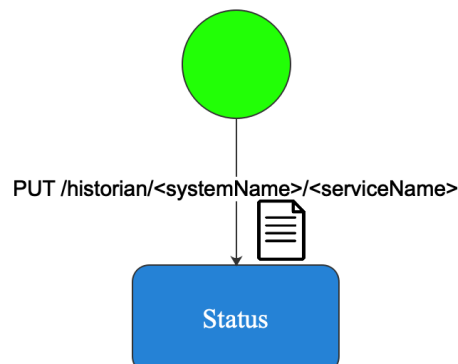Figure 2 shows how a client can store data at a service endpoint.



PUT /historian/<systemName>/<serviceName>

Status

**FIGURE 4: HISTORIAN PUTDATA OPERATION**

11

# 2.3.  Service 3: **Proxy**

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 3 Function description**

| Function | Service | Method | Input | Output |
|---|---|---|---|---|
| ListSystems | Proxy | POST | - | DataManagerSystems |
| ListServices | Proxy | GET | systemName | DataManagerServices |
| GetData | Proxy | GET | systemName, serviceName | Sensor data |
| PutData | Proxy | PUT | systemName, serviceName, plus Sensor data | Result code |

# 2.3.1. ListSystems: Information Model

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

Output: Example DataManagerSystems response
```
{
 "systems": ["temperatureSys1", "humiditySys2"]
}
```

# 1.1.21. ListSystems: Parameters

This interface does not take any query path parameters.

# 1.1.22. ListSystems: Response codes

| Code | Meaning | Comment |
|---|---|---|

| 200 | Successful request | Success |
|-----|---------------------|---------|
| 401 | Unauthorized | Access denied |
| 500 | Internal server error | In case of errors etc. |

# 1.1.23. ListSystems: Error handling

All errors are handled using HTTP response codes, see above. An error message is also added in the response payload.

# 1.1.24. ListSystems: Interaction with consumers

Figure 2 shows how a client can perform a ListSystems operation.



GET /proxy

DataManager
Systems
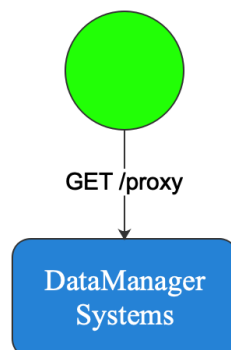
**FIGURE 3: PROXY LISTSYSTEMS OPERATION**

# 2.3.2. ListServices: Information Model

In order to get a list of service endpoints, a GET request must be sent to the corresponding URI. to the /proxy/<systemName> endpoint. The response upon success is a list of all service endpoints.

Output: Example DataManagerServices response
```
{
 "services": ["temperature", "humidity"]
}
```

# 1.1.25. ListServices: Parameters

This interface does not take any query path parameters.

# 1.1.26. ListServices: Response codes

| Code | Meaning | Comment |
| --- | --- | --- |
| 200 | Successful request | Success |
| 400 | Bad request | If an incorrect parameter is used |
| 401 | Unauthorized | Access denied |
| 500 | Internal server error | In case of errors etc. |

# 1.1.27. ListServices: Error handling

All errors are handled using HTTP response codes, see above. An error message is also added in the response payload.

# 1.1.28. ListServices:  Interaction with consumers

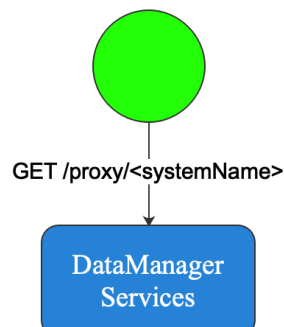Figure 2 shows how a client must perform a ListServices operation.

GET /proxy/<systemName>

DataManager
Services

**FIGURE 4: PROXY LISTSERVICES OPERATION**

# 2.3.3. GetData: Information Model

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

Output: Example SenML response

```
[
  {"bn": "temperature", "bt": 1593759331, "bu": "Cel"},
  {"n"; "bearingTempInner", "v": 42.1},
  {"n"; "bearingTempOuter", "v": 34.5}
]
```

# 1.1.29. GetData: Parameters

Unlike the more advanced Historian service, the Proxy service does not take any query path parameters.

# 1.1.30. GetData: Response codes

| Code | Meaning | Comment |
|------|---------|---------|
| 200 | Successful request | Success |
| 400 | Bad request | If the request contains incorrect parameters |
| 401 | Unauthorized | Access denied |
| 404 | Not found | If the requested system and service combination does not exist |
| 500 | Internal server error | If an error occurs |

# 1.1.31. GetData: Error handling

If the request was successful, a SenML message 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.

# 1.1.32. GetData: Interaction with consumers

Figure 2 shows how a client must perform a GetData operation.

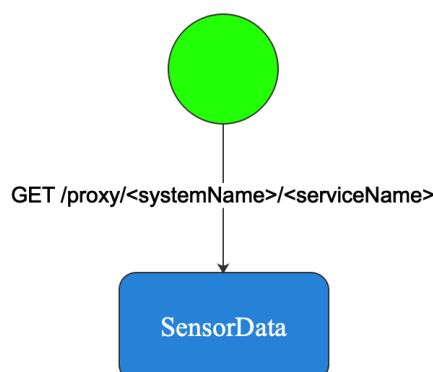GET /proxy/<systemName>/<serviceName>

SensorData

**FIGURE 4: PROXY GETDATA OPERATION**

# 2.3.4. PutData: Information Model

In order to store data at an endpoint, a PUT request must be sent to the corresponding URI, to the /proxy/<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/proxy/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}
]
```

# 1.1.33. PutData: Parameters

This interface does not take any query path parameters.

# 1.1.34. PutData: Response codes

| Code | Meaning | Comment |
| --- | --- | --- |
| 200 | Successful request | Success |
| 400 | Bad request | If the request contains incorrect parameters |
| 401 | Unauthorized | Access denied |
| 500 | Internal server error | If an error occurs |

# 1.1.35. PutData: Error handling

If the request was successful, 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.

# 1.1.36. PutData: Interaction with consumers

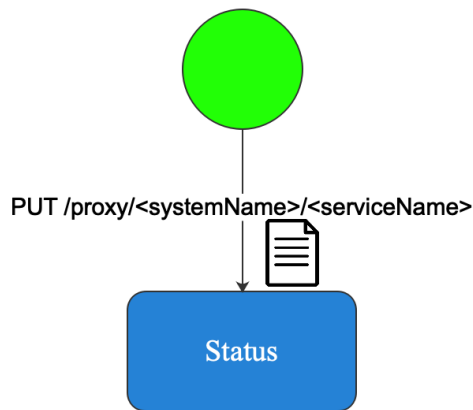Figure 2 shows how a client can store data at a service endpoint.

**FIGURE 4: PROXY PUTDATA OPERATION**

# 3.     Security

This system can either run unencrypted over HTTP, or using TLS plus server and client side X.509 certificates.

## 3.1.  Certificates

This IDD is using the same certificates as other core systems in the Java Spring versions.

## 3.2.  Payload protection

Currently, no separate payload protection is supported.

# 4.     References

[1] Arrowhead Framework repository: https://github.com/arrowhead-f/core-java-spring

[2]

# 5.     Revision history

## 5.1.  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 | 2020-07-03 | 2.3 | Finalized text and figures | Jens Eliasson |
| 7 | | | | |

## 5.2. Quality Assurance

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

# 6. Appendixes

Appendix A: REST Communication profile (CP)