The DataManager System

System Design Document

**Abstract**

This document describes the DataManager support Core System.

1. Overview of the DataManager System

Table 1 System Information

|  |  |
| --- | --- |
| **Name** | DataManager System |
| **Owner** | Jens Eliasson, jens.eliasson@thingwave.eu |

This System provides two Services:

* Historian (HS)
* Proxy (PS)

The first service is the Historian Service (HS), which provides features for storing and retrieving sensor data and generic files. The second service is the Proxy Service (PS), which provides features for mailbox-like behaviour where low-power, e.g. sleepy devices, systems can push messages to during short periods of online time. Clients can afterwards fetch these messages at any time. More information is available in [1].

1. Use-cases

|  |
| --- |
| **Name of the Use-case** |
| **ID**: PS-1 |
| **Brief description**:  The Proxy service is used to cache data for a low-power IoT device during times of offline. |
| **Primary actors**:  Low-power (sleepy) device (System), a Proxy service and a client. |
| **Secondary actors**:  n/a |
| **Preconditions**:  Arrowhead Core services such as ServiceRegistry, Authorization and Orchestration must be started |
| **Main flow**:   1. The sleepy client creates a dynamic endpoint at the Proxy service. 2. The sleepy client then pushes messages to the newly created endpoint. 3. Consuming systems can fetch the last message from the dynamic endpoint 4. When the sleepy device is done, it can delete the endpoint to inform consumers that no more data will be available. |
| **Postconditions**:  n/a |
| **Alternative flows**:  The Historian service can also be used for the same purpose, albeit with higher overhead and latency. |

|  |
| --- |
| **Name of the Use-case** |
| **ID**: HS-1 |
| **Brief description**:  The Historian service is used to store data from an Arrowhead sensor system (producer). |
| **Primary actors**:  An Arrowhead sensor, the DataManager |
| **Secondary actors**:  Sensor data consuming system |
| **Preconditions**:  The Core services are available for security, service registry and orchestration. |
| **Main flow**:   1. The sensor system pushes data to the DataManager using the Historian service at regular or intervals, or when events occur 2. The consuming system fetches data using the database capabilities of the Historian the guarantee that no messages are lost. |
| **Postconditions**:  n/a |
| **Alternative flows**:  The Proxy could be used, however since the proxy only stores one message, data can be lost if the consumer(s) do to fetch fast enough. |

1. Application services

# Produced Services

|  |  |
| --- | --- |
| Service | Description |
| Historian | To be used in a local cloud |
| Proxy | To be used in a local cloud |

# Consumed Services

|  |  |
| --- | --- |
| Service | Description |
| ServiceRegistry | To publish the Historian and Proxy services |
| Orchestration | To know which other services to interact with (currently none) |
| Authorisation | To handle access control |

1. Security

The DataManager can run in secure or insecure mode. In insecure mode, any system can publish and fetch data to any resource. In secure mode, the communication is encrypted, only a system with a valid certificate can publish to the corresponding resource. Details about the security is available in the documentation [2] and [3].

1. Internal Structure

This module is a simple Java jar executable. It uses the config folder contents, where the configuration files are. The code includes the following classes:

* **DataManagerMain**: starts the HTTP or the HTTPS server based on the configuration files and command line arguments, registers into the SR
* **DataManagerResource**: provides resources for the Proxy and Historian
* **DataManagerService**: provides functionality for the Historian service
* **ProxyService**: provides functionality for the Proxy service

# Usage

Start the module as a Java executable. The following command line arguments are available:

* “-tls”: starts the Core System in secure (HTTPS) mode, using the certificates which were set in the app.properties file.
* “-daemon” (Linux only!): starts the module in daemon mode, kill signals will prompt a normal shutdown, and the core system will de-register its services from the Service Registry.
* “-d”: starts the module in debug mode, which means every incoming REST request (URL + payload) and the corresponding response will be printed to the console output.

1. References
2. IoT Automation: Arrowhead Framework, CRC Press, ISBN 9781498756754
3. Arrowhead Framework Github: <https://github.com/arrowhead-f/>
4. Arrowhead Wiki: <http://www.arrowhead.eu/arrowhead-wiki/>
5. Revision history

# Amendments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No. | Date | Version | Subject of Amendments | Author |
| 1 | 2018-09-21 | G4.0 | First draft | Jens Eliasson |
| 2 | 2018-10-29 | G4.0 | Use cases updated | Jens Eliasson |
| 3 | 2019-03-28 | G4.0 | Text improvements | Jens Eliasson |
| 4 | 2020-11-17 | G4.1.3 | Updated to 4.1.3 | Jens Eliasson |
| 5 |  |  |  |  |

# Quality Assurance

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