Proxy HTTP/TLS/JSON

Interface Design Description

Service ID: *"proxy"*

Abstract
This document describes the Proxy service IDD for HTTP/TLS/JSON.

ARTEMIS Innovation Pilot Project: Arrowhead
THEME [SP1-JTI-ARTEMIS-2012-AIPP4 SP1-JTI-ARTEMIS-2012-AIPP6]
[Production and Energy System Automation Intelligent-Built environment and urban infrastructure for sustainable and friendly cities]

Contents

1 Overview **3**

2 Service Interfaces **4**
2.1 function GetSystems . . . 4
2.2 function GetSystemServices . . . 4
2.3 function Push . . . 4
2.4 function Fetch . . . 5

3 Information Model **6**
3.1 struct SensorData . . . 6

4 References **7**

5 Revision History **8**
5.1 Amendments . . . 8
5.2 Quality Assurance . . . 8

1 Overview

This document describes the HTTP/TLS/JSON variant of the Proxy Eclipse Arrowhead service. The Proxy service is used to exchange messages between a data producing system and a data consuming system. One of th design goals is to allow battery-powered (sleepy) devices to make their data available at all times for consumers to use.

This document exists as a complement to the *Proxy – Service Description* (ProxySD) document. For further details about how this service is meant to be used, please consult that document. The rest of this document describes how to realize the Proxy service using HTTP [1], TLS [2] and JSON [3], both in terms of its interfaces (Section 2) and its information model (Section 3).

2 Service Interfaces

This section lists the interfaces that must be exposed by Proxy services in alphabetical order. In particular, each subsection first names the HTTP method and path used to call the interface, after which it names an abstract interface from the Proxy service's SD document, as well as input and output types. All interfaces in this section respond with the HTTP status code `202 OK` if called successfully, unless otherwise is stated.

2.1 GET /datamanager/proxy

Interface: GetSystems

Called to list of all active Arrowhead systems, as exemplified in Listing 1.

```
GET /datamanager/proxy HTTP/1.1

[
  "examplesystem1", "examplesystem2", "examplesystemN"
]
```

Listing 1: A SystemList list.

Code	Type	Description
200	OK	No error
401	UNAUTHORIZED	No valid authorization
500	INTERNAL SERVER ERROR	Database error etc

Table 1: GetSystems responses

2.2 GET /datamanager/proxy/$systemName

Interface: GetSystemServices

Called to get a list of all serrvice endpoints that a specific systame has pushed data to using Push An example is given in Listing 2.

```
GET /datamanager/proxy/examplesystem1 HTTP/1.1

[
  "service1", "service1", "service2","serviceN"
]
```

Listing 2: A GetSystemServices invocation.

Code	Type	Description
200	OK	No error
400	BAD REQUEST	Bad input
401	UNAUTHORIZED	No valid authorization
500	INTERNAL SERVER ERROR	Database error etc

Table 2: GetSystems responses

2.3 PUT /datamanager/proxy/$systemName/$serviceName

Interface: Push
Input: SensorData

Called to store sensor data in a service endpoint. An example is given in Listing 3.

```
PUT /datamanager/proxy/examplesystem1/service1 HTTP/1.1

[
  {"bn": "examplesystem1","bt":1.276020076001e+09, "bu":"Cel", "bver":5,
   {"n": "temperature ","t": 0,"v":22.2},
   {"n": "temperature ","t":-15,"v":21.9}
]
```

Listing 3: A Push invocation with a SenML-encoded message.

Code	Type	Description
200	OK	No error
400	BAD REQUEST	Bad input
401	UNAUTHORIZED	No valid authorization
500	INTERNAL SERVER ERROR	Database error etc

Table 3: Push responses

2.4 GET /datamanager/proxy/$systemName/$serviceName

Interface: Fetch
Output: SensorData

Called to retrive already stored sensor data at a service endpoint. An example is given in Listing 4.

```
GET /datamanager/proxy/examplesystem1/service1 HTTP/1.1

[
  {"bn": "examplesystem1","bt":1.276020076001e+09, "bu":"Cel", "bver":5,
   {"n": "temperature ","t": 0,"v":22.2},
   {"n": "temperature ","t":-15,"v":21.9}
]
```

Listing 4: A Fetch Response with a SenML-encoded message.

Code	Type	Description
200	OK	No error
400	BAD REQUEST	Bad input
401	UNAUTHORIZED	No valid authorization
404	NOT FOUND	No such system/service combination
500	INTERNAL SERVER ERROR	Database error etc

Table 4: Fetch responses

3 Information Model

The SensorData request payload contains generic sensor data. SensorData normally contains information about the unit, source system, timestamp and metadata. The default payload type is JSON-encoded SenML (RFC 8428). The response to a Store/ Retrieve request is a simple HTTP status code (Created/OK – request was a success, No Content – request had no effect). For the Push interface, the content-type must be set to 'application/json'. For Fetch, the response content-type is 'application/json'.

3.1 struct SensorData

All messages must be encoded using SenML [4].

Object Field	Value Type	Description
”bn”	Base name	Base name of a message.
”bt”	Base time	Base time.
”bu”	Base unit	Base unit.
”bver”	Base version	Base version (not used).
”n”	Signal name	Name of sensor signal measurement.
”t”	Time	Time of measurement.
”u”	Unit	Unit of measurement.

4 References

[1] R. Fielding and J. Reschke, “Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing,” RFC 7230, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7230

[2] E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.3,” RFC 8446, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC8446

[3] T. Bray, “The JavaScript Object Notation (JSON) Data Interchange Format,” RFC 7159, 2014, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC7159

[4] C. Jennings and Z. Shelby, “Sensor Measurement Lists (SenML),” RFC 8428, 2018, RFC Editor. [Online]. Available: https://doi.org/10.17487/RFC8428

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2018-09-17	4.0	Initial	Jens Eliasson
2	2018-10-30	4.0	Text update	Jens Eliasson
3	2019-03-20	4.0	Updated data model to RFc8428	Jens Eliasson
4	2020-05-07	4.1.3	Updated text	Jens Eliasson
5	2020-11-30	4.1.3	New template	Jens Eliasson

5.2 Quality Assurance

No.	Date	Version	Approved by
1	2020-12-08	5	Jerker Delsing