Translator HTTP/TLS/JSON

Interface Design Description

Service ID: *"translator"*

Abstract
This document describes the Translator 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

This document describes the HTTP/TLS/JSON variant of the Translator Eclipse Arrowhead service. The Translator service is used to exchange messages between a data producing systems and a data consuming systems with different protocols. Tranlator service includes three components a FIWARE interface, a Translator Hub between HTTP and CoAP, and a Plugin Service to generate dinamically services.

This document exists as a complement to the *Translator – Service Translator* (Translator SD) 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 Translator 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 Translator services. 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 Translator service's SD document, as well as input and output types. All interfaces in this section respond with the HTTP status code `200 OK` if called successfully, unless otherwise is stated.

2.1 POST /

Interface: PostTranslator
Input: TranslatorSetup
Output: TranslatorHubAccess

Called to post a new Translator Hub, as exemplified in Listing 1.

```
POST / HTTP/1.1

{
  "id": "abcd",
  "ip": "0.0.0.0",
  "port": 0
}
```

Listing 1: A TranslatorHubAccess.

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

Table 1: PostTranslator responses

2.2 GET /all

Interface: GetTranslatorList
Output: TranslatorHubAccess

Called to request all available Translator Hubs, as exemplified in Listing 2.

```
GET /all HTTP/1.1

[
  {
    "id": "abcd",
    "ip": "0.0.0.0",
    "port": 0
  }
]
```

Listing 2: A TranslatorHubAccess List.

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

Table 2: GetTranslatorList responses

2.3 GET /$id

Interface: GetTranslator
Input: translatorId
Output: TranslatorHubAccess

Called to request an specific Translator Hub, as exemplified in Listing 3.

```
GET /$id HTTP/1.1

{
  "id": "abcd",
  "ip": "0.0.0.0",
  "port": 0
}
```

Listing 3: A TranslatorHubAccess.

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

Table 3: GetTranslator responses

2.4 GET /plugin/service/$entityId/$serviceName

Interface: PluginGetEntityValue
Output: Object

Called to request a service value with Plugin Service, as exemplified in Listing 4, 5 and 6.

```
GET /plugin/service/$entityId/$serviceName HTTP/1.1 ACCEPT=TEXT_PLAIN_VALUE

unknown
```

Listing 4: Response in Text Plain.

```
GET /plugin/service/$entityId/$serviceName HTTP/1.1 ACCEPT=APPLICATION_SENML

{
  "bt": 0,
  "v": 0,
  ...
}
```

Listing 5: Response in SenML.

```
GET /plugin/service/$entityId/$serviceName HTTP/1.1 ACCEPT=APPLICATION_JSON

{
  "time": 0,
  "value": 0,
  ...
}
```

Listing 6: Response in JSON.

Code	Type	Description
200	OK	No error
400	BAD REQUEST	Bad request
401	UNAUTHORIZED	No valid authorization
404	NOT FOUND	Resource not found
415	UNSUPPORTED MEDIA TYPE	Wrong requested Media Type
500	INTERNAL SERVER ERROR	Server error, etc

Table 4: PluginGetEntityValue responses

2.5 GET /v2

Interface: FiwareGetIt
Output: FiwareUrlServices

Called to request the URLs of the FIWARE services supported, as exemplified in Listing 7.

```
GET /v2 HTTP/1.1

{
  "entities_url": "/v2/entities",
  "types_url": "/v2/types",
  "subscriptions_url": "/v2/subscriptions",
  "registrations_url": "/v2/registrations"
}
```

Listing 7: A FiwareUrlServices.

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

Table 5: FiwareGetIt responses

2.6 GET /v2/entities

Interface: FiwareListEntities
Input: id,type,idPattern,typePattern,q,mq,georel,geometry,coords,limit,offset,attrs,metadata,orderBy,options
Output: FiwareEntity

Called to request the URLs of the FIWARE services supported, as exemplified in Listing 8.

```
GET /v2/entities HTTP/1.1

[
  {
    "id": "Example Id",
    "type": "Example Type",
    "other#1": "example",
    "other#2": [
      "example"
    ]
  }
]
```

Listing 8: A FiwareEntity List.

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

Table 6: FiwareListEntities responses

2.7 POST /v2/entities

Interface: FiwareCreateEntity
Input: FiwareEntity

Called to create a new FIWARE Entity, as exemplified in Listing 9.

```
POST /v2/entities HTTP/1.1
```

Listing 9: A FiwareEntity creation.

Code	Type	Description
200	OK	No error
401	UNAUTHORIZED	No valid authorization
422	UNPROCESSABLE ENTITY	Impossible to process the entity
500	INTERNAL SERVER ERROR	Server error, etc

Table 7: FiwareCreateEntity responses

2.8 GET /v2/entities

Interface: FiwareRetrieveEntity
Input: id,type,attrs,metadata,options
Output: FiwareEntity

Called to request an specific FIWARE Entity, as exemplified in Listing 10.

```
GET /v2/entities HTTP/1.1

{
  "id": "Example Id",
  "type": "Example Type",
  "other#1": "example",
  "other#2": [
    "example"
  ]
}
```

Listing 10: A FiwareEntity.

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

Table 8: FiwareRetrieveEntity responses

2.9 GET /v2/entities/$id/attrs

Interface: FiwareRetrieveEntityAttributes
Input: id,type,attrs,metadata,options
Output: JSON Object

Called to request the attributes of an specific FIWARE Entity, as exemplified in Listing 11.

```
GET /v2/entities/$id/attrs HTTP/1.1

{
  "value": 21.7,
  "type": "Number",
  "metadata": {}
}
```

Listing 11: A JSON Object attirbutes example.

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

Table 9: FiwareRetrieveEntityAttributes responses

2.10 POST /v2/entities/$id/attrs

Interface: FiwareUpdateAppendEntityAttributes
Input: id,type,attrs,metadata,options,JSON Object

Called to add or update the attributes of an specific FIWARE Entity, as exemplified in Listing 12.

```
POST /v2/entities/$id/attrs HTTP/1.1
```

Listing 12: A JSON Object attirbutes example.

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

Table 10: FiwareUpdateAppendEntityAttributes responses

2.11 DELETE /v2/entities/$id

Interface: FiwareRemoveEntity
Input: type

Called to remove an specific FIWARE Entity, as exemplified in Listing 13.

```
DELETE /v2/entities/$id HTTP/1.1
```

Listing 13: A FiwareRemoveEntity request example.

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

Table 11: FiwareRemoveEntity responses

2.12 GET /v2/types

Interface: FiwareListEntityTypes
Input: limit,offset,options
Output: JSON Object

Called to request a list of registered FIWARE types, as exemplified in Listing 14.

```
GET /v2/types HTTP/1.1

[
  {
    "type": "Car",
    "attrs": {
      "speed": {
        "types": [
          "Number"
        ]
      },
      "fuel": {
        "types": [
          "gasoline",
          "diesel"
        ]
      },
      "temperature": {
        "types": [
          "urn:phenomenum:temperature"
        ]
      }
    },
    "count": 12
  }
]
```

Listing 14: A JSON Object types list example.

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

Table 12: FiwareListEntityTypes responses

2.13 GET /v2/types/$type

Interface: FiwareRetrieveEntityType
Output: JSON Object

Called to request a specific registered FIWARE types, as exemplified in Listing 15.

```
GET /v2/types/$type HTTP/1.1
```

```
2
3  {
4    "attrs": {
5      "pressure": {
6        "types": [
7          "Number"
8        ]
9      },
10     "humidity": {
11       "types": [
12         "percentage"
13       ]
14     },
15     "temperature": {
16       "types": [
17         "urn:phenomenum:temperature"
18       ]
19     }
20   },
21   "count": 7
22 }
```

Listing 15: A JSON Object type example.

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

Table 13: FiwareRetrieveEntityType responses

3 Information Model

The default payload type is JSON-encoded. The response to a GET 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 TranslatorHubAccess

All messages must be encoded using JSON.

Object Field	Value Type	Description
”id”	Id	Hub Id.
”ip”	IP address	IP Address of the Hub.
”port”	IP port	IP port of the Hub.

3.2 struct TranslatorSetup

All messages must be encoded using JSON.

3.3 struct FiwareUrlServices

All messages must be encoded using JSON.

Object Field	Value Type	Description
”producerName”	Name	Producer Name.
”producerAddress”	Address	Producer Address.
”consumerName”	Name	Consumer Name.
”consumerAddress”	Address	Consumer Address.

Object Field	Value Type	Description
”entities_url”	URL path	Entities URL path.
”types_url”	URL path	Types URL path.
”subscriptions_url”	URL path	Subscriptions URL path.
”registrations_url”	URL path	Registrations URL path.

3.4 struct FiwareEntity

All messages must be encoded using JSON.

Object Field	Value Type	Description
”id”	Entity Id	Id of the Entity.
”type”	Entity Type	Type of the Entity.
”others”	Any	Any other parameters.

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

5 Revision History

5.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1	2019-03-27	1.0	Initial	Pablo Puñal Pereira
2	2019-05-02	1.1	Models and Interfaces update	Pablo Puñal Pereira
3	2020-12-11	2.0	Template Update	Pablo Puñal Pereira

5.2 Quality Assurance

No.	Date	Version	Approved by
1			