fiware-epcis-gateway Documentation

Release 1.0.0

Yalew Kidane

Feb 07, 2022

Contents

Oliot-MG is a mediation gateway which translates information from NGSI based IoT platform to EPCIS based IoT platform. This enables capturing state change in FIWARE context broker in the form of EPCIS Event.

CHAPTER 1

Background

To solve the issue of interoperability, multiple companies, organizations, and consortia have started to join and create standards. Currently, the two of the major standards that are widely being considered in the IoT sector are EPCIS and NGSI. Nevertheless, the two standards differ both in data encoding and service interface which create fragmentation from the point of view of data consumers application. Moreover, the two platforms differ in the underlying philosophy of representing and storing IoT data; namely, NGSI is entity-based and EPCIS is event-based. This creates an overhead to analyze and process data coming from the two platforms.

FIWARE - EPCIS mediation gateway is developed to solve the interoperability between NGSI and EPCIS. It translates the entity based data from Orion context broker to EPCIS event. Moreover, enables traceability by capturing state change in FIWARE context broker in the form of EPCIS Event.

1.1 Install

1.1.1 Install FIWARE server

Use the following page to install FIWARE: FIWARE Install

1.1.2 Install EPCIS server

- **Option 1:**
  - Make sure you have installed *mongodb/mysql* whichever you are using
  - Download epcis war file from mediation github
  - Download apache tomcat 8 from apchage page
  - After extracting the apache tomcat file put the epcis war file in to *path/to/your_tomcat_download/apache-tomact-8.x.xx/webapps*
  - On terminal go *path/to/your_tomcat_download/apache-tomact-8.x.xx/bin/*
  - [for Linux] *sh ./catalina.sh run*

  - [for Window] *use .bat file*

- **Option 2:**
  - Follow the instruction the original EPCIS GitHub

1.1.3 Mediation gateway

A jar file is included in the github. To run the mediation gateway the following command can be used

```
java -jar path/to/the_jar_file/fiware_oiliot_mediation-1.0.0.jar
```

A 'Dockerfile' is also availble in the github jar file. The following code can be used build and run the mediation gateway

```
docker build -t fiware_oiliot_mediation .
docker run -p 8081:8081 fiware_oiliot_mediation
```

Provide the following information after you run the mediation gateway:

```
Enter FIWARE server URL (e.g localhost:2016) :
        #localhost:1026
Enter FIWARE server URL (e.g localhost:8080) :
        #localhost:8080
Mediation Gateway Port (e.g 8083):
        #8081
```

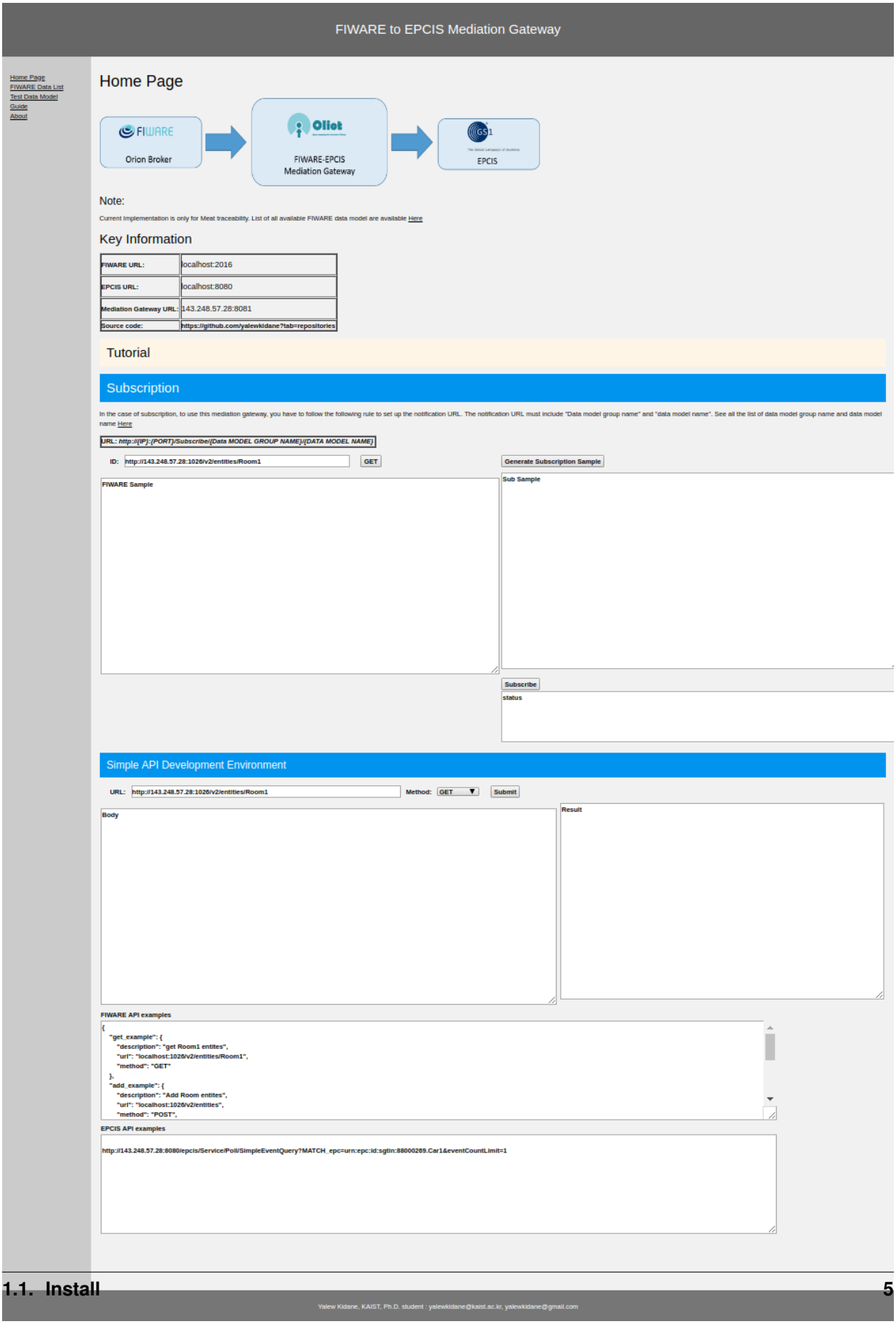
Note:

After that, the mediation gateway will run and you can access the interface through any browser
localhost:Mediation_Gateway_Port/home

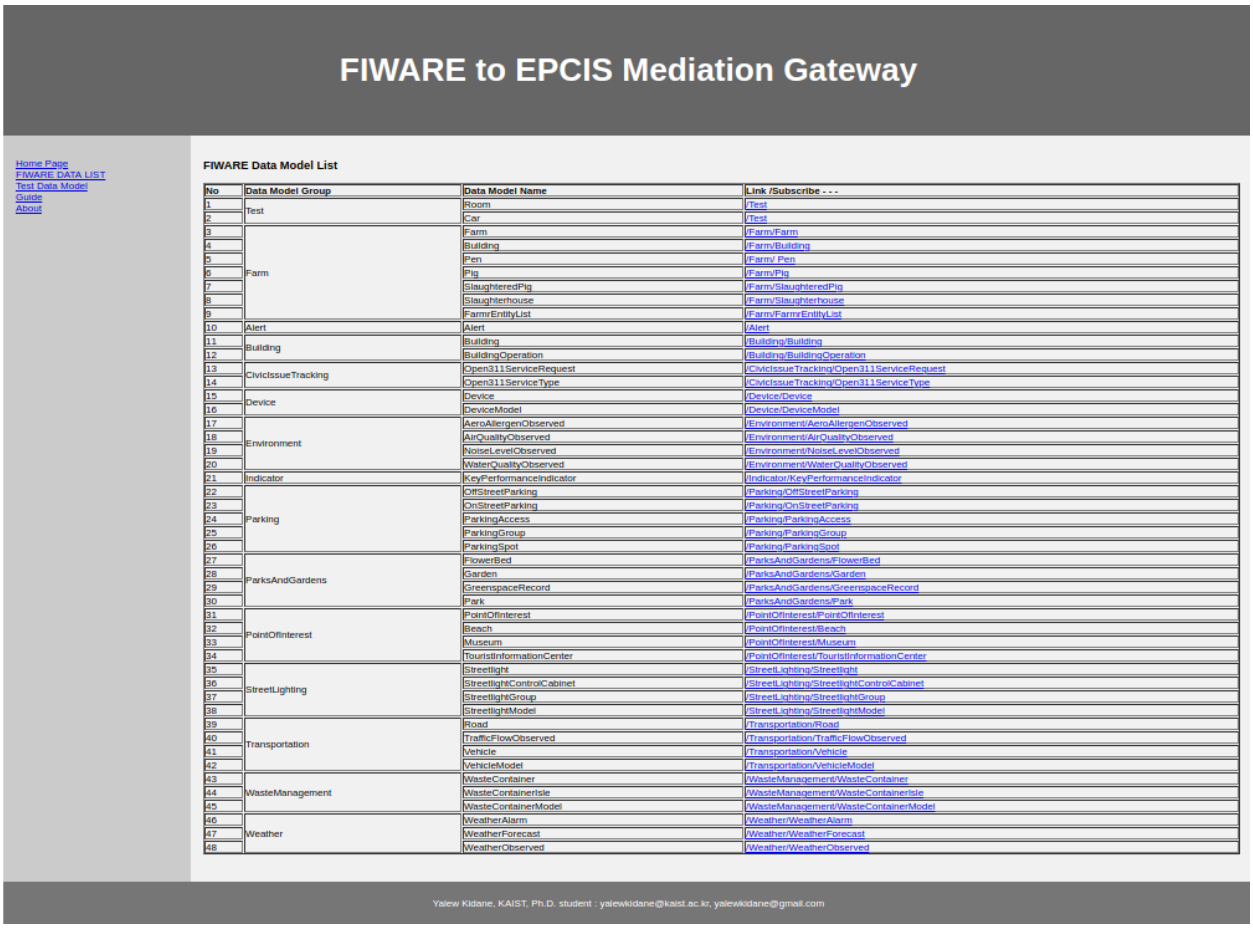
From the above example the url should be: *localhost:8081/home*

After that you will see the interfaces presented below

Main page *{IP}:{PORT} /home*


FIWARE to EPCIS Mediation Gateway
Home Page
FIWARE Data List
Test Data Model
Guide
About
Home Page
Orion Broker
FIWARE-EPCIS
Mediation Gateway
EPCIS
Note:
Current Implementation is only for Meat traceability. List of all available FIWARE data model are available Here
Key Information
FIWARE URL: localhost:2016
EPCIS URL: localhost:8080
Mediation Gateway URL: 143.248.57.28:8081
Source code: https://github.com/yalewkidane?tab=repositories
Tutorial
Subscription
In the case of subscription, to use this mediation gateway, you have to follow the following rule to set up the notification URL. The notification URL must include "Data model group name" and "data model name". See all the list of data model group name and data model name Here
URL: http://{IP}:{PORT}/Subscribe/{Data MODEL GROUP NAME}/{DATA MODEL NAME}
ID: http://143.248.57.28:1026/v2/entities/Room1
GET
Generate Subscription Sample
FIWARE Sample
Sub Sample
Subscribe
status
Simple API Development Environment
URL: http://143.248.57.28:1026/v2/entities/Room1
Method: GET
Submit
Body
Result
FIWARE API examples
{
"get_example": {
"description": "get Room1 entites",
"url": "localhost:1026/v2/entities/Room1",
"method": "GET"
},
"add_example": {
"description": "Add Room entites",
"url": "localhost:1026/v2/entities",
"method": "POST",
EPCIS API examples
http://143.248.57.28:8080/epcis/Service/Poll/SimpleEventQuery?MATCH_epc=urn:epc:id:sgtin:88000269.Car1&eventCountLimit=1
Yalew Kidane, KAIST, Ph.D. student : yalewkidane@kaist.ac.kr, yalewkidane@gmail.com

Fiware Data List *{IP}:{PORT} /FiwareDataModel*



FIWARE to EPCIS Mediation Gateway

Home Page
FIWARE DATA LIST
Test Data Model
Guide
About

FIWARE Data Model List

No	Data Model Group	Data Model Name	Link /Subscribe - - -
1	Test	Room	/Test
2		Car	/Test
3	Farm	Farm	/Farm/Farm
4		Building	/Farm/Building
5		Pen	/Farm/ Pen
6		Pig	/Farm/Pig
7		SlaughteredPig	/Farm/SlaughteredPig
8		Slaughterhouse	/Farm/Slaughterhouse
9		FarmrEntityList	/Farm/FarmrEntityList
10	Alert	Alert	/Alert
11	Building	Building	/Building/Building
12		BuildingOperation	/Building/BuildingOperation
13	CivicIssueTracking	Open311ServiceRequest	/CivicIssueTracking/Open311ServiceRequest
14		Open311ServiceType	/CivicIssueTracking/Open311ServiceType
15	Device	Device	/Device/Device
16		DeviceModel	/Device/DeviceModel
17	Environment	AeroAllergenObserved	/Environment/AeroAllergenObserved
18		AirQualityObserved	/Environment/AirQualityObserved
19		NoiseLevelObserved	/Environment/NoiseLevelObserved
20		WaterQualityObserved	/Environment/WaterQualityObserved
21	Indicator	KeyPerformanceIndicator	/Indicator/KeyPerformanceIndicator
22	Parking	OffStreetParking	/Parking/OffStreetParking
23		OnStreetParking	/Parking/OnStreetParking
24		ParkingAccess	/Parking/ParkingAccess
25		ParkingGroup	/Parking/ParkingGroup
26		ParkingSpot	/Parking/ParkingSpot
27	ParksAndGardens	FlowerBed	/ParksAndGardens/FlowerBed
28		Garden	/ParksAndGardens/Garden
29		GreenspaceRecord	/ParksAndGardens/GreenspaceRecord
30		Park	/ParksAndGardens/Park
31	PointOfInterest	PointOfInterest	/PointOfInterest/PointOfInterest
32		Beach	/PointOfInterest/Beach
33		Museum	/PointOfInterest/Museum
34		TouristInformationCenter	/PointOfInterest/TouristInformationCenter
35	StreetLighting	Streetlight	/StreetLighting/Streetlight
36		StreetlightControlCabinet	/StreetLighting/StreetlightControlCabinet
37		StreetlightGroup	/StreetLighting/StreetlightGroup
38		StreetlightModel	/StreetLighting/StreetlightModel
39	Transportation	Road	/Transportation/Road
40		TrafficFlowObserved	/Transportation/TrafficFlowObserved
41		Vehicle	/Transportation/Vehicle
42		VehicleModel	/Transportation/VehicleModel
43	WasteManagement	WasteContainer	/WasteManagement/WasteContainer
44		WasteContainerIsle	/WasteManagement/WasteContainerIsle
45		WasteContainerModel	/WasteManagement/WasteContainerModel
46	Weather	WeatherAlarm	/Weather/WeatherAlarm
47		WeatherForecast	/Weather/WeatherForecast
48		WeatherObserved	/Weather/WeatherObserved

Yalew Kidane, KAIST, Ph.D. student : yalewkidane@kaist.ac.kr, yalewkidane@gmail.com

1.2 Usage

1.2.1 Simple Subscription example

1. Check for a specific entities on FIWARE before subscription (eg. Room8)

Server	[FIWARE] localhost:1026/v2
Method	GET
URL	localhost:1026/v2/entities/Room8
Headers	Content-Type: application/json
Status	404 Not Found
Response	{ “error”: “NotFound”, “description”: “The requested entity has not been found. Check type and id” }
Comment	Entity Room8 doesn’t exist in FIWARE so we need to create it first

2. Add a Room entity to FIWARE before subscription (eg. Room8)

Server	[FIWARE] localhost:1026/v2
Method	POST
URL	localhost:1026/v2/entities
Headers	Content-Type: application/json
Status	404 Not Found

Body	{
	“id”: “Room8”,
	“type”: “Room”,
	“pressure”: {
	“type”: “Integer”,
	“value”: 123,
	“metadata”: {}
	},
	“temperature”: {
	“type”: “Float”,
	“value”: 28,
	“metadata”: {}
	}
	}

Response	{ }
Comment	Entity Room8 doesn’t exist in FIWARE so we need to create it first

3. Check the created entity

Server	[FIWARE] localhost:1026/v2
Method	GET
URL	localhost:1026/v2/entities/Room8
Headers	Content-Type: application/json
Status	200 OK

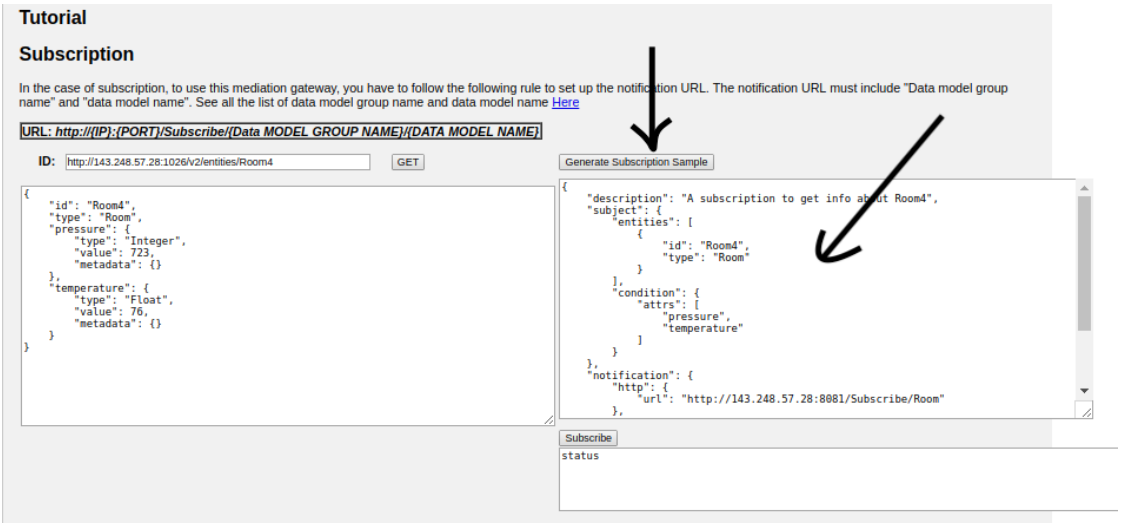
Response	{
	“id”: “Room8”,
	“type”: “Room”,
	“pressure”: {
	“type”: “Integer”,
	“value”: 123,
	“metadata”: {}
	},
	“temperature”: {
	“type”: “Float”,
	“value”: 28,
	“metadata”: {}
	}
	}

Comment	Entity Room8 created in step 2 is returned

4. Generate sample subscription.

How to make subscription body	Refer FIWARE
How to make subscription URL	To use this mediation gateway, you have to follow the following rule to set up the notification URL. The notification URL must include “Data group name” and “data model name” and it should looks like *http://{IP}:{PORT}/Subscribe/{DATA MODEL GROUP NAME}/{DATA MODEL NAME}* **IP:** IP address of the mediation gateway **Port:** port address of the mediation gateway running **DATA MODEL GROUP NAME}/DATA MODEL NAME :** check *http://{IP}:{PORT}//FiwareDataModel* example: • http://localhost:8081/Subscribe/Test/Room • http://localhost:8081/Subscribe/Test/Car • http://localhost:8081/Subscribe/Farm/Building
Sample subscription body	{ “description”: “A subscription to get info about Room8”, “subject”: { “entities”: [{ “id”: “Room8”, “type”: “Room” }], “condition”: { “attrs”: [“pressure”, “temperature”] } }, “notification”: { “http”:

You can use the mediation gateway to generate sample



5. Check if there is epcis event related to Room 8

Note: During translation sample key is generated as follows *urn:epc:id:sgtin:88000269.[entityID]*

EPCIS Query

Server	[EPCIS] localhost:8080
Method	GET
URL	http://localhost:8080/epcis/Service/Poll/SimpleEventQuery?MATCH_epc=urn:epc:id:sgtin:88000269.Room8
Status	200 OK
Response	<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <EPCISQueryDocumentType xmlns:ns2="http://www.unece.org/cefact/namespaces/StandardBusinessDocumentHeader" xmlns:ns4="urn:epcglobal:epcis:xsd:1" xmlns:ns3="urn:epcglobal:epcis-query:xsd:1"> <EPCISBody> <ns3:QueryResults> <queryName>SimpleEventQuery</queryName> <resultsBody> <EventList/> </resultsBody> </ns3:QueryResults> </EPCISBody> </EPCISQueryDocumentType>
Comment	It returns empty event list

6. Subscribe

Server	[FIWARE] localhost:1026/v2
Method	POST
URL	localhost:1026/v2/ subscriptions
Headers	Content-Type: application/json

Sample subscription body	{
	“description”: “A sub-scrip-tion to get info about Room8”,
	“subject”: {
	“entities”: [
	{
	“id”: “Room8”,
	“type”: “Room”
	}
	],
	“condition”: {
	“attrs”: [
	“pressure”,
	“temperature”
	]
	}
	},
	“notification”: {
	“http”: {
	“url”: “http: // 143. 248. 57.

7. Check if there is epcis event related to Room 8 after the subscription

Server	[EPCIS] localhost:8080
Method	GET
URL	http://localhost:8080/epcis/Service/Poll/SimpleEventQuery?MATCH_epc=urn:epc:id:sgtin:88000269.Room8
Status	200 OK
Response	<?xml version="1.0" encoding="UTF-8" standalone="yes"?> <EPCISQueryDocumentType xmlns:ns2="http://www.unece.org/cefact/namespaces/StandardBusinessDo xmlns:ns4="urn:epcglobal:epcis:xsd:1" xmlns:ns3="urn:epcglobal:epcis-query:xsd:1"> <EPCISBody> <ns3:QueryResults> <queryName>SimpleEventQuery</queryName> <resultsBody> <EventList> <ObjectEvent> <eventTime>2018-08-28T17:22:09.363Z</eventTime> <recordTime>2018-08-28T17:22:09.417Z</recordTime> <eventTimeZoneOffset>-05:00</eventTimeZoneOffset> <baseExtension> <eventID>4829cb2a-97a9-43fd-bf31-fb0374a7c792</eventID> </baseExtension> <epcList> <epc>urn:epc:id:sgtin:88000269.Roon </epcList> <action>OBSERVE</action> <bizStep>urn:epcglobal:cbv:bizstep:driving <disposition>urn:epcglobal:cbv:disp:on_the line</disposition> <readPoint> <id>urn:epc:id:sgln:8800026900016.R </readPoint>

8. Update any value of the Room

Server	[FIWARE] localhost:1026/v2
Method	PATCH
URL	localhost:1026/v2/entities/Room8/attrs
Headers	Accept: application/json
Content-Type:	application/json

Body	{
	“pressure”: {
	” “type”: “Integer”,
	” “value”: 123,
	” “metadata”: {}
	” },
	” “temperature”: {
	” “type”: “Float”,
	” “value”: 40,
	” “metadata”: {}
	” }
	“}

Status	204 No Content
Comment	Temperature value of Entity Room8 is updated to 40

9. Check if there are two Room8 events are created in epcis

Server	[EPCIS] localhost:8080
Method	GET
URL	http://localhost:8080/epcis/Service/Poll/SimpleEventQuery?MATCH_epc=urn:epc:id:sgtin:88000269.Room8
Status	200 OK
Response	<?xml version=”1.0” encoding=”UTF-8” standalone=”yes”?> <EPCISQueryDocumentType xmlns:ns2=”http://www.unece.org/cefact/namespaces/StandardBusin xmlns:ns4=”urn:epcglobal:epcis:xsd:1” xmlns:ns3=”urn:epcglobal:epcis-query:xsd:1”> <EPCISBody> <ns3:QueryResults> <queryName>SimpleEventQuery</queryName> <resultsBody> <EventList> <ObjectEvent> <eventTime>2018-08-28T17:22:09.363Z</eventTime> <recordTime>2018-08-28T17:22:09.417Z</recordTime> <eventTimeZoneOffset>-05:00</eventTimeZoneOffset> <baseExtension> <eventID>4829cb2a-97a9-43fd-bf31-fb0374a7c792</eventID> </baseExtension> <epcList> <epc>urn:epc:id:sgtin:880002 </epcList> <action>OBSERVE</action> <bizStep>urn:epcglobal:cbv:bizst <disposition>urn:epcglobal:cbv:dline</disposition> <readPoint>

1.2.2 Appendix

GS1 Key proposal for farming

Objects to be identified	FIRWARE Key	GS1 Key	Comment
Farm	urn:entity:farm:<farmId>	urn:epc:id:sgln:{companyPrefix}:{locationReference}:{extensionComponent}	SGLN is used here (GIAI can be used) Example: Farm→100 *urn:epc:id:sgln:88000269:100:<farmId>*
Building	urn:entity:building:<buildingId>	urn:epc:id:sgln:{companyPrefix}:{locationReference}:{extensionComponent}	SGLN is used here (GIAI can be used) Example: building →101 *urn:epc:id:sgln:88000269:101:<buildingId >*
Pen	urn:entity:pen:<penId>	urn:epc:id:sgln:{companyPrefix}:{locationReference}:{extensionComponent}	SGLN is used here (GIAI can be used) Example: building →102 *urn:epc:id:sgln:88000269:101:<penId >*
Pig	urn:entity:pig:<pigId>	urn:epc:id:sgtin:{companyPrefix}:{ItemReference}:{SerialNumber}	GTIN is used here Example: building →103 *urn:epc:id:sgtin:88000269:101:<pigId >*
slaughterhouse	urn:entity:slaughterhouse:<slaughterhouseId>	urn:epc:id:sgln:{companyPrefix}:{locationReference}:{extensionComponent}	SGLN is used here (GIAI can be used) Example: building →104 *urn:epc:id:sgln:88000269:104:<slaughterhouseId >*

FIWARE data models schema for farm

Farm Entity

```
{
  "$id": "https://resl.com/farm.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Farm",
  "type": "object",
  "properties": {
        "farmId":{
                "type": "Text",
                "description": "It represents the id of the Farm Entity (the <farmId>␣
↪contained in the EntityId)"
    },
        "type":{
                "type": "Text",
                "value": "Farm",
                "description": "Entity Type"
    },
    "address": {
      "type": "Text",
      "description": "It represents the address of the farm",
          "metadata":{}
    },
    "name": {
      "type": "Text",
      "description": "It represents the name of the farm",
          "metadata":{}
    },
    "ownerCompany": {
      "type": "Text",
      "description": "It represents the name of the company that owns the farm",
          "metadata":{}
    }
```

(continues on next page)

(continued from previous page)

```
  }
}

{
        "farmId":"urn:entity:farm:<farmID>",
        "type":"Farm",
        "address":"La Cañada 04120 Almería Spain",
        "name":"Greenhouse agriculture",
        "ownerCompany":"Maria"

}
```

Building Entity

```
{
  "$id": "https://resl.com/farm.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Building",
  "type": "object",
  "properties": {
        "buildingId":{
                "type": "Text",
                "description": "It represents the id of the Building Entity (the
↪<buildingId> contained in the EntityId attribute)"
    },
        "type":{
                "type": "Text",
                "value": "Building",
                "description": "Entity Type"
    },
    "name": {
      "type": "Text",
      "description": "It represents the name of the building",
          "metadata":{}
    },
    "lastUpdate": {
      "type": "DateTime",
      "description": "It represents the timestamp of the last update",
          "metadata":{}
    },
    "farmId": {
      "type": "Text",
      "description": "It represents the id of the Farm in which the Building is␣
↪located (the farmId)",
          "metadata":{}
    },
    "temperature": {
      "type": "Float",
      "description": "It represents the last value of the temperature registered␣
↪within the Building",
          "metadata":{
           "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#DegreeCelsius"
                }
          }
```

(continues on next page)

(continued from previous page)

```
    },
    "humidity": {
      "type": "Float",
      "description": "It represents the last value of the humidity registered within␣
↪the Building",
          "metadata":{
           "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#Humidity"
                }
          }
    },
    "luminosity": {
      "type": "Float",
      "description": "It represents the last value of the luminosity registered␣
↪within the Building",
          "metadata":{
           "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#LuminousIntensity"
                }
          }
    }
  }
}

{
        "buildingId":"urn:entity:building:<buildingId>",
        "type":"Building",
        "name":{
            "type": "Text",
            "value": "La Cañada 04120 Almería Spain",
            "metadata": {}
    },
        "lastUpdate":{
            "type": "ISO8601",
            "value": "2018-08-22T05:10:58.00Z",
            "metadata": {}
    },
        "farmId":"urn:entity:farm:<farmID>",
        "temperature":{
            "type": "Float",
            "value": 37.6,
            "metadata": {
                                "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#DegreeCelsius"
                                }
                        }
    },
        "humidity":{
            "type": "Float",
            "value": 45,
            "metadata": {
```

(continues on next page)

(continued from previous page)

```
                                    "uom": {
                        "type": "string",
                        "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#Humidity"
                                    }
                            }
    },
        "luminosity":{
            "type": "Float",
            "value": 0.6,
            "metadata": {
                                    "uom": {
                        "type": "string",
                        "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#LuminousIntensity"
                                    }
                            }
    }
}
```

Pen Entity

```
{
  "$id": "https://resl.com/farm.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Pig",
  "type": "object",
  "properties": {
        "pigId":{
                "type": "Text",
                "description": "It represents the id of the Pig Entity (the <pigId>␣
↪contained in the EntityId attribute)"
    },
        "type":{
                "type": "Text",
                "value": "Pig",
                "description": "Entity Type"
    },
    "serialNumber": {
      "type": "Text",
      "description": "If a serial number is assigned to the pig by the farm, this␣
↪field contains such a value",
          "metadata":{}
    },
    "lastUpdate": {
      "type": "DateTime",
      "description": "It represents the timestamp of the last update",
          "metadata":{}
    },
    "penId": {
      "type": "Text",
      "description": "It represents the id of the Farm in which the pen is located␣
↪(the penId)",
          "metadata":{}
    },
    "weight": {
      "type": "Float",
```

(continues on next page)

(continued from previous page)

```
      "description": "It represents the current weight of the pig (the last measured␣
↪value)",
          "metadata":{
           "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#Kilogram"
                }
          }
    },
    "totalConsumedWater": {
      "type": "Float",
      "description": "it represents the amount of water that was consumed between the␣
↪moment in
                                          which the pig started to drink and the␣
↪current moment (e.g., if the pig started
                                          to drink 2 minutes ago and is continuing to␣
↪drink, this value contains the
                                          total amount of water that the pig drunk␣
↪since 2 minutes ago)",
          "metadata":{
           "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#Litre"
                }
          }
    },
    "totalConsumedFood": {
      "type": "Float",
      "description": "it represents the amount of food that was consumed between the␣
↪moment
                                              in which the pig started to eat and␣
↪the current moment (e.g., if the
                                              pig started to eat 2 minutes ago and␣
↪is continuing to eat, this value
                                              contains the total amount of food␣
↪that the pig ate since 2 minutes ago)",
          "metadata":{
           "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#Kilogram"
                }
          }
    }
  }
}

{
        "pigId":"urn:entity:pig:<pigId>",
        "type":"Pig",
        "serialNumber":{
            "type": "Text",
            "value": "8764321000003",
            "metadata": {}
    },
```

(continues on next page)

(continued from previous page)

```
        "lastUpdate":{
            "type": "ISO8601",
            "value": "2018-08-22T05:10:58.00Z",
            "metadata": {}
    },
        "penId":"urn:entity:pen:<penId>",
        "weight":{
            "type": "Float",
            "value": 37.6,
            "metadata": {
                                "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#Kilogram"
                                }
                            }
    },
        "totalConsumedWater":{
            "type": "Float",
            "value": 20,
            "metadata": {
                                "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#Litre"
                                }
                            }
    },
        "totalConsumedFood":{
            "type": "Float",
            "value": 45,
            "metadata": {
                                "uom": {
                    "type": "string",
                    "value": "http://ontology.fiesta-iot.eu/ontologyDocs/m3-lite.owl
↪#Kilogram"
                                }
                            }
    }
}
```

The mediation gateway is licensed under `Apache 2.0`.