ETSI TR 118 565 V5.0.0 (2023-12)

TECHNICAL REPORT

oneM2M-SensorThings API interworking
(oneM2M TR-0065 version 5.0.0)

Reference

DTR/oneM2M-000065

Keywords

API, IoT, M2M, sensor

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Foreword

This Technical Report (TR) has been produced by ETSI Partnership Project oneM2M (oneM2M).

1 Scope

The present document defines an interworking of both standards oneM2M and OGC SensorThings API.

The present document is structured as followed:

- Introduction and background to OGC-SensorThings API.
- Describing interworking scenarios that are relevant but not exclusive for Smart City (there are also examples from other areas and verticals as well).
- Technical comparison of oneM2M and OGC/STA.
- Describing a technical solution for interworking of both standards; there might be interworking on different level:
  - opaque data routing;
  - data model mapping between oneM2M-SDT and OGC.
- Developing test cases for interworking between oneM2M and OGC SensorThings API.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] oneM2M Drafting Rules.

[i.2] SensorThings API website.

[i.3] OGC SensorThings API Part 1: Sensing Version 1.1.

[i.4] ETSI TS 118 133 (V3.0.0): "Interworking Framework (oneM2M TS-0033)".

[i.5] ETSI TS 118 123 (V4.7.1): "SDT based Information Model and Mapping for Vertical Industries".

[i.6] FROST®-Server - Open-Source-Implementierung der OGC SensorThings API - Fraunhofer IOSB.

NOTE: FROST® is the trade name of a product supplied by Fraunhofer IOSB. This information is given for the convenience of users of the present document and does not constitute an endorsement by oneM2M of the product named. Equivalent products may be used if they can be shown to lead to the same results.

[i.7] API documentation for the Open Geospatial Consortium (OGC) SensorThings API Documentation.

[i.8] oneM2M TS-0003: "Security Solutions".

[i.9]	OWASP Secure Coding Practices Quick Reference Guide.
[i.10]	OASIS: "MQTT Version 3.1.1", OASIS Standard. 29 October 2014. Edited by Andrew Banks and Rahul Gupta.

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

OGC	Open Geospacial Consortium
STA	SensorThing API

4 Conventions

The key words "Shall", "Shall not", "May", "Need not", "Should", "Should not" in the present document are to be interpreted as described in the oneM2M Drafting Rules [i.1].

5 Background

5.0 Introduction

The following text explains, what is SensorThings API and how does it work.

The SensorThings API (STA) is a standard of the Open Geospatial Consortium (OGC). It provides a framework for communication and exchanging data between sensors and applications. The standard is devided in two parts. SensorThings API Part 1 is dedicated to sensing and was published in 2016 [i.2]. Part 2 deals with tasking and was published in 2019 [i.2]. There is an OGC certified Open-Source SensorThings API Server [i.6] available. It supports the OGC SensorThings API Part 1: Sensing. It also includes preliminary actuation/tasking support.

For the description of entites SensorThings API uses "Sensing Entities" [i.3] and JSON as data format. The communication is REST-based and uses HTTP and also CoAP, MQTT, 6LowPAN.

5.1 SensorThings API Architecture

A typical STA-based architecture works in client/server mode. A sensor device pushes data to the SensorThings Server via HTTP POST request. A SensorThings Server may also support MQTT protocol to support publish and subscribe capabilities. An interested application can subscribe to the MQTT-Broker, in order to get notified about new sensor events.

The data in the SensorThings server are organized as "Sensing Entities" [i.3] (see Figure 5.1-1: "Sensing Entities" data model).

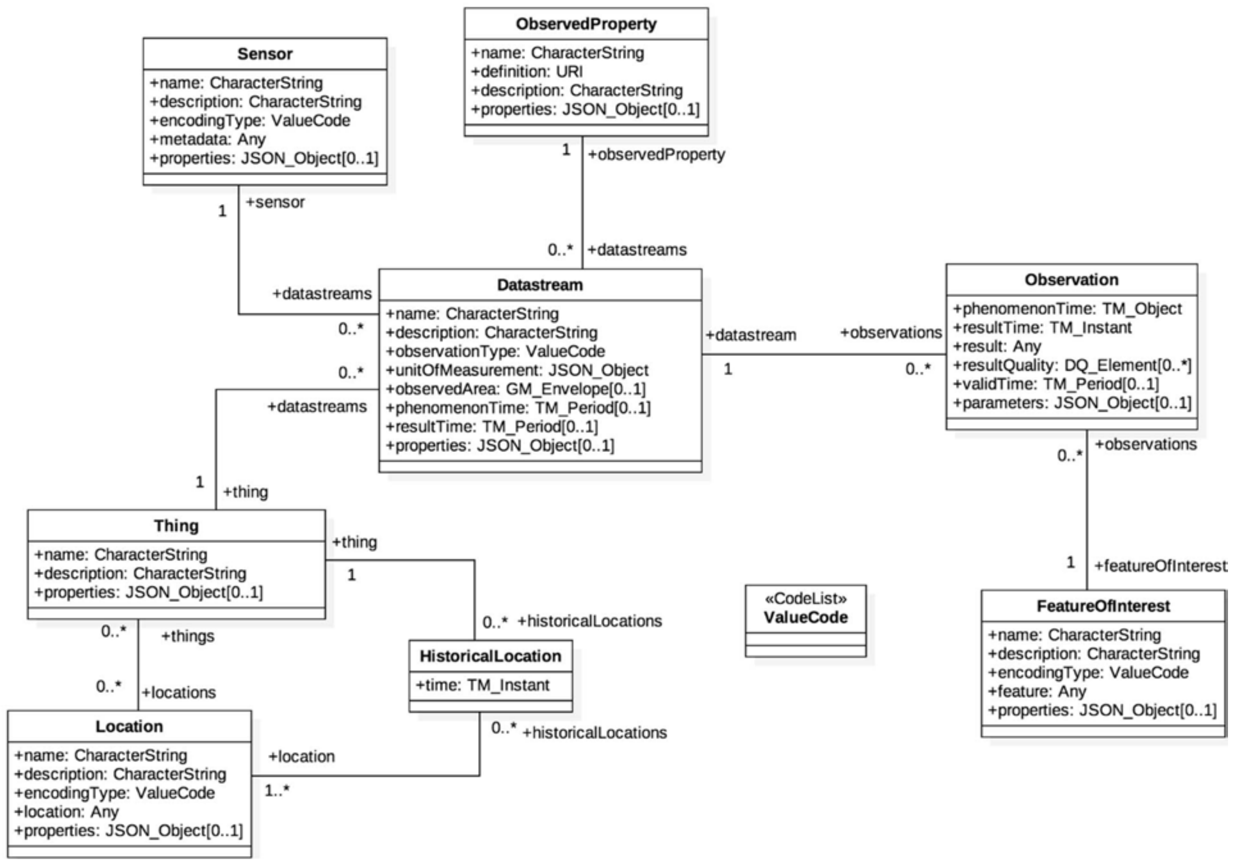

Sensor
+name: CharacterString
+description: CharacterString
+encodingType: ValueCode
+metadata: Any
+properties: JSON_Object[0..1]
ObservedProperty
+name: CharacterString
+definition: URI
+description: CharacterString
+properties: JSON_Object[0..1]
Datastream
+name: CharacterString
+description: CharacterString
+observationType: ValueCode
+unitOfMeasurement: JSON_Object
+observedArea: GM_Envelope[0..1]
+phenomenonTime: TM_Period[0..1]
+resultTime: TM_Period[0..1]
+properties: JSON_Object[0..1]
Observation
+phenomenonTime: TM_Object
+resultTime: TM_Instant
+result: Any
+resultQuality: DQ_Element[0..*]
+validTime: TM_Period[0..1]
+parameters: JSON_Object[0..1]
Thing
+name: CharacterString
+description: CharacterString
+properties: JSON_Object[0..1]
HistoricalLocation
+time: TM_Instant
Location
+name: CharacterString
+description: CharacterString
+encodingType: ValueCode
+location: Any
+properties: JSON_Object[0..1]
«CodeList»
ValueCode
FeatureOfInterest
+name: CharacterString
+description: CharacterString
+encodingType: ValueCode
+feature: Any
+properties: JSON_Object[0..1]
+sensor
+observedProperty
+datastreams
+datastream
+observations
+thing
+things
+locations
+historicalLocations
+location
+featureOfInterest
1
0..*
1..*

Figure 5.1-1: STA Sensing Entities data model

In the SensorThings data model events or sensor data are called "observations". Before a sensor is able to push an observation to the server it needs at least a 'Thing' and a 'Datastream' entity. This is created beforehand. One 'Thing' might have different 'Sensors', one 'Location' or many 'HistoricalLocations'.

Taking a closer look at the SensorThings data model and the purpose of data within the data model discloses mainly two data characteristics, associated with a 'thing':

a) Data observations originated by sensors or commands sent to interact with actuators may be seen as IoT data from oneM2M point of view.

While:

b) Data embedded in the SensorThings data model, like "historic locations" may be seen as data for documentation purposes e.g. for the city administration.

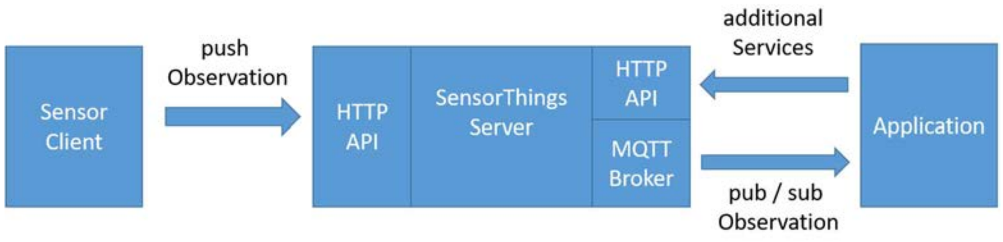


Figure 5.1-2: Example STA message flow

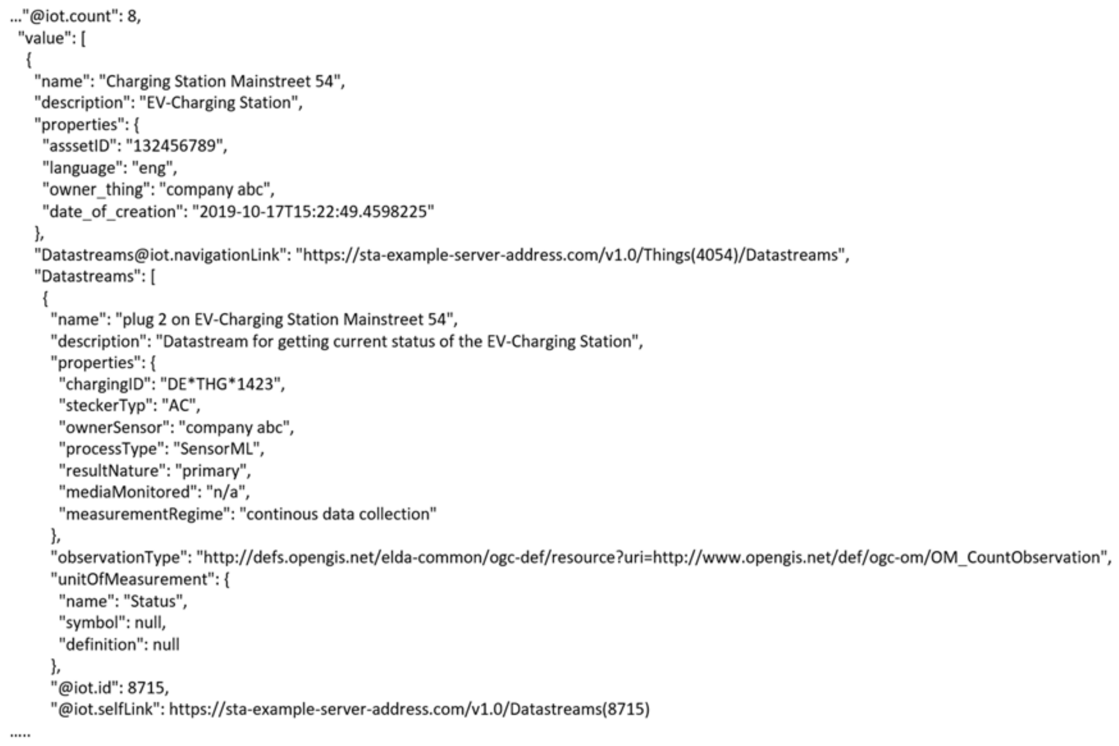
5.2 SensorThings API example use-case

The message flow in Figure 5.1-2 can be explained by using an example of an application that wants to use data of EV-Charging Stations in a city.

1) In order to get observations belonging to the cities EV-Charging Stations the application needs to know the relevant "Datastreams". Therefore the application needs to send a request with filter parameter to the HTTP-API of the server e.g.:

   https://sta-example-server-address.com/v1.0/Things?$filter=substringof("Charging",name)&$count=true&$expand=Datastreams

   As a response the server provides a list of all "Datastreams" belonging to a Charging Stations.



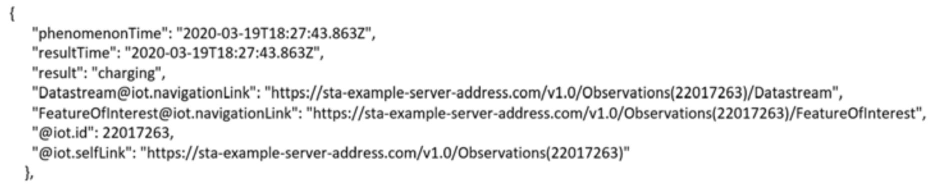
```
..."@iot.count": 8,
 "value": [
  {
   "name": "Charging Station Mainstreet 54",
   "description": "EV-Charging Station",
   "properties": {
    "asssetID": "132456789",
    "language": "eng",
    "owner_thing": "company abc",
    "date_of_creation": "2019-10-17T15:22:49.4598225"
   },
   "Datastreams@iot.navigationLink": "https://sta-example-server-address.com/v1.0/Things(4054)/Datastreams",
   "Datastreams": [
    {
     "name": "plug 2 on EV-Charging Station Mainstreet 54",
     "description": "Datastream for getting current status of the EV-Charging Station",
     "properties": {
      "chargingID": "DE*THG*1423",
      "steckerTyp": "AC",
      "ownerSensor": "company abc",
      "processType": "SensorML",
      "resultNature": "primary",
      "mediaMonitored": "n/a",
      "measurementRegime": "continous data collection"
     },
     "observationType": "http://defs.opengis.net/elda-common/ogc-def/resource?uri=http://www.opengis.net/def/ogc-om/OM_CountObservation",
     "unitOfMeasurement": {
      "name": "Status",
      "symbol": null,
      "definition": null
     },
     "@iot.id": 8715,
     "@iot.selfLink": https://sta-example-server-address.com/v1.0/Datastreams(8715)
.....
```

Figure 5.2-1: Example entry from the result list returned by filter request

2) The application can now subscribe to these "Datastreams". Figure 5.2-1 shows one entry of the result list. It represents an EV-Charging Station as a "Thing" and includes details of "Datastreams".

3) As soon as the sensor (EV-Charging Station) changes its status e.g. from "available" to "charging" it pushes an observation to the server.

4) The application gets the observation through a notification that is sent by the MQTT Broker. An example of an observation is shown in Figure 5.2-2. In the result field the current status of the EV-Charging Station is shown as "charging".



```
{
   "phenomenonTime": "2020-03-19T18:27:43.863Z",
   "resultTime": "2020-03-19T18:27:43.863Z",
   "result": "charging",
   "Datastream@iot.navigationLink": "https://sta-example-server-address.com/v1.0/Observations(22017263)/Datastream",
   "FeatureOfInterest@iot.navigationLink": "https://sta-example-server-address.com/v1.0/Observations(22017263)/FeatureOfInterest",
   "@iot.id": 22017263,
   "@iot.selfLink": "https://sta-example-server-address.com/v1.0/Observations(22017263)"
  },
```

Figure 5.2-2: Example STA observation

6 Architecture Model of OGC/STA to oneM2M interworking

6.0 Introduction

Figure 6.0-1 shows an architecture approach for an Interworking Proxy Entity (IPE) between oneM2M and the OGC SensorThings API. The IPE is located between a oneM2M CSE and an OGC/SensorThings API (STA)-Server.

The basic interworking enables applications that are connected to an oneM2M-based system to get data from sensors that are connected to an OGC/STA server. Furthermore, an application that is connected to an OGC/STA server will be able to get data from sensors that are connected to an oneM2M-based system.

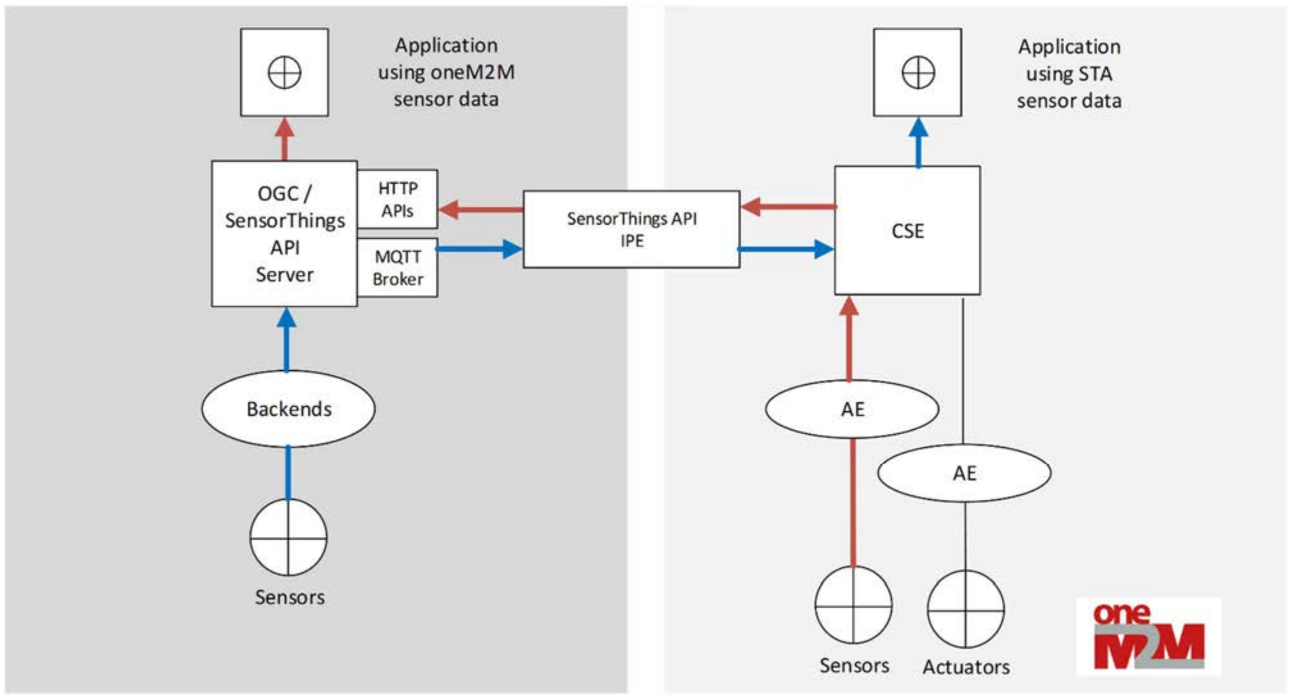

Application using oneM2M sensor data
Application using STA sensor data
OGC / SensorThings API Server
HTTP APIs
MQTT Broker
SensorThings API IPE
CSE
Backends
AE
AE
Sensors
Sensors
Actuators
oneM2M

Figure 6.0-1: IPE architecture overview with data flow

6.1 OGC/STA-to-oneM2M Data Model Mapping

According to ETSI TS 118 133 [i.4] a representation of a non-oneM2M Proximal IoT function/device in a oneM2M-specified resource instance is to be synchronized with the entity that it represents.

This means that the OGC/STA data model is represented in the hosting CSE. The data in the OGC/STA server are organized as Sensing Entities [i.3] (see Figure 5.1-1: STA Sensing Entities data model).

The oneM2M structure for data models is a tree-structure where data are organized in containers or trees of containers.

The OGC/STA data model is a relational one, as used in databases, and not hierarchical. Thus, it creates a challenge for full interworking of all data captured in the OGC/STA data model.

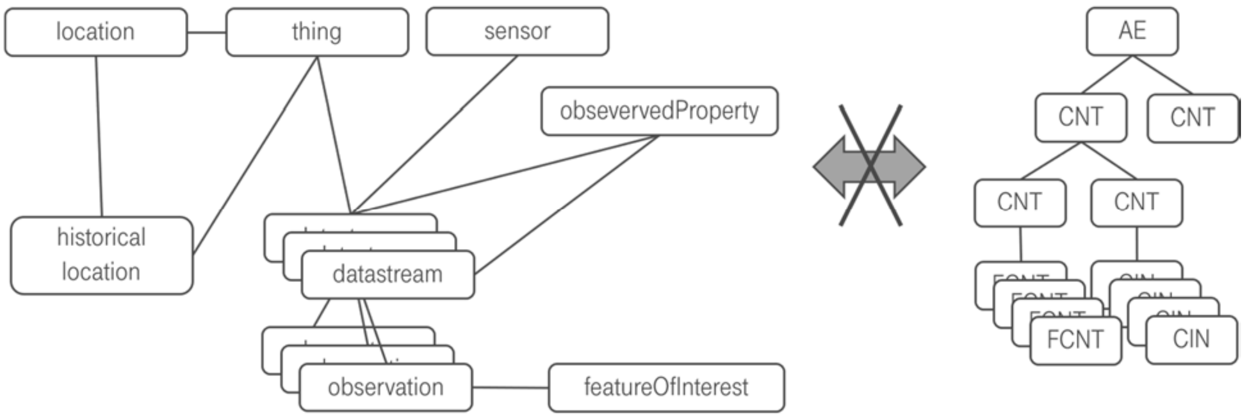


Figure 6.1-1: OGC data model cannot directly be mapped to oneM2M

One reason for this challenge is, as already outlined in clause 5.1. The SensorThings data model is comprehensive and may be regarded as a n:m relational database structure, holding both:

i. sensor (IoT-data); and

ii. administrative data (like historic locations or historic products IDs).

Aiming at a full interworking on all data being available in the OGC/STA data model, would require either:

i. additional functionality (possibly a new Common Service Function CSF) within the oneM2M CSE being able to build up, tear down and maintain such relationships between content instances and flex containers based on Mca commands; or

ii. the IPE internally keeps track about relations and inserts "helper" references in "flexContainers", pointing to related data (e.g. historical locations) in other flex containers. However, this would allow an interworking from OGC side towards oneM2M, but an AE on oneM2M side would not be able to retrieve those related data since the relation is only known to the IPE.

In the following clause the different approaches are visited in more detail.

6.2 Architecture Approaches

6.2.0 Introduction

In the following clause possible options and shortfalls in the context of a full data interworking and the consequences for the architecture of an interworking between both standards are being discussed. In the following clauses, three different approaches and their advantages and disadvantages are discussed.

6.2.1 "Flat Data Model" Approach

6.2.1.0 Overview

One approach is to create a separate oneM2M container entity for every single group of entities in the OGC data model. Figure 6.2.1.0-1 shows a oneM2M data model that could represent the OGC/STA - data in a hosting CSE.

The top of a tree may be an <*AE*> resource. Below in Figure 6.2.1.0-1, there are <*container*> resources representing the dedicated objects of the OGC and Sensing Entities [i.3] data model. They are all at the same level and represent a flat data model representation.

There may be, for example, one <*container*> resource where all incoming "Observation" objects from SensorThings API are stored by the IPE. There may be another <*container*> where the "Datastream" object for this "Observation" is stored. There may be also <*container*> resources for "Location"-, "Sensor"-, "Thing" or other objects.

The OGC/STA - "Observation" object itself may be represented as an <*flexContainer*>, under a <*container*> with a resource name "observations".

The OGC/STA -"Datastream" object itself may also represented as an <*flexContainer*> under a <*container*>with a resource name "Datastreams".

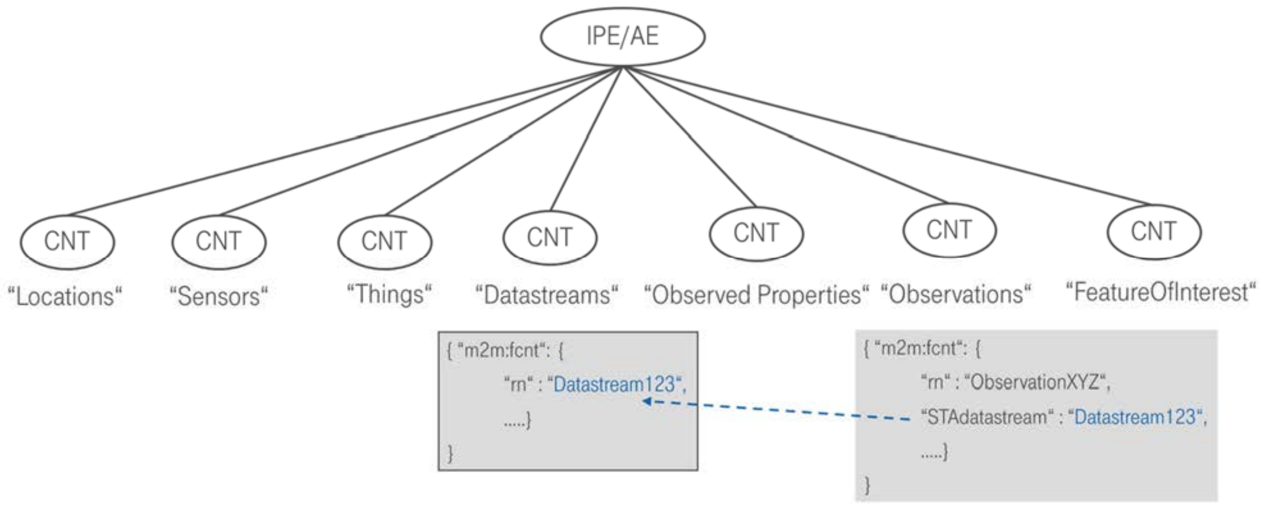
In Figure 6.2.1.0-1 these <*flexContainer*> resources are represented as grey boxes. This way a oneM2M data structure represents all the entities from the OGC and ISO 19156 data model.

The relationships between the ISO 19156 entities can be represented using dedicated custom attributes defined inside a <*flexContainer*> specialization.

Figure 6.2.1.0-1 shows the <*flexContainer*> with the resource name "ObservationXYZ", that has an attribute "STAdatastream" with the value "Datastream123". This Attribute points to a <*flexContainer*> representing the "*Datastream*" object with the resource name "Datastream123" located in the "Datastreams" <*container*>. This way the attribute represents inherent the relationships from the OGC/STA data model.

Actually, two kinds of relationships are described in this approach:

- The first kind are the oneM2M specific <*AE*>-to- <*container*>-to-<*flexContainer*> relationships (grey/continuous lines).
- The second kind of relationships are OGC specific and are expressed as attributes inside the <*flexContainer*> entities. Here for example a "Observation" -to- "Datastream" relationship is described (blue/dashed line).



```
{ "m2m:fcnt": {
        "rn" : "Datastream123",
        .....}
}
```

```
{ "m2m:fcnt": {
        "rn" : "ObservationXYZ",
        "STAdatastream" : "Datastream123",
        .....}
}
```

Figure 6.2.1.0-1: The flat data model with inherent relationship

6.2.1.1 Missing Relationship Management

As already briefly addressed in clause 6.1, the inherent relationships (dashed blue line in Figure 6.2.1.0-1) describing the OGC/STA data model are not in scope to be managed by the CSE today. Only the relationships between <*AE*> and <*container*> can be managed (grey lines) by the CSE.

For example, in case that a certain <*flexContainer*>, representing a "Datastream", is deleted, all related <*flexContainer*> resources representing an "*Observation*" need to be deleted too. This is how it is handled in an OGC/STA Server, but this would not be possible by current oneM2M CSE functionality.

NOTE: This approach is rather a theoretical one and shows the issue of a missing oneM2M relationship management in the CSE.

6.2.2 "Generic" Approach

6.2.2.0 Overview

In the "generic" approach subscriptions are used to replicate observation data between oneM2M <containers> and OGC Datastreams.

In order to transfer data from a oneM2M sensor to OGC/STA the IPE creates a <subscription> to the <container> resource with the desired data and when a new <contentInstance> is added it gets a <notification> message containing the <contentInstance> resource.

Figure 6.2.2.0-1 shows the oneM2M-to-OGC/STA direction. Based upon the creation of the <*contentInstance*> in the hosting CSE, the IPE gets a <*notification*> message including the <*contentInstance*>. The IPE constructs an "*Observation*" creation request and copies the 'content' attribute of the <*contentInstance*> to the 'result' attribute of the "*Observation*" shown in Figure 6.2.2.0-2 and sends it to the OGC/STA server.

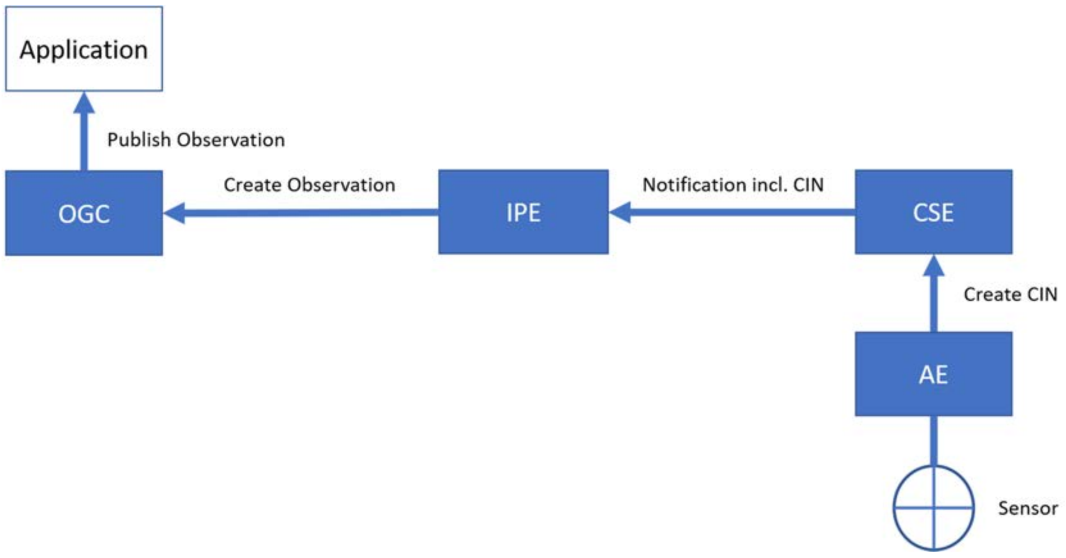
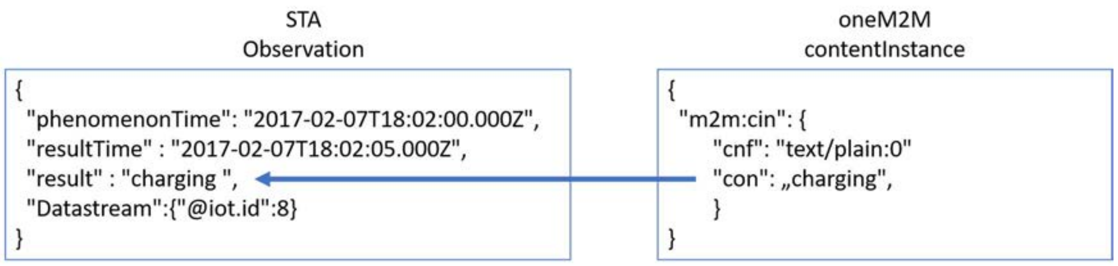


Figure 6.2.2.0-1: Gateway oneM2M-to-OGC/STA direction



```
{
  "phenomenonTime": "2017-02-07T18:02:00.000Z",
  "resultTime" : "2017-02-07T18:02:05.000Z",
  "result" : "charging ",
  "Datastream":{"@iot.id":8}
}
```

```
{
 "m2m:cin": {
     "cnf": "text/plain:0"
     "con": „charging",
     }
}
```

Figure 6.2.2.0-2: Content copying from CIN-to-Observation

Figure 6.2.2.0-3 shows the OGC/STA-to-oneM2M direction. OGC/STA does not provide a publish/subscribe mechanism on HTTP protocol level, but OGC allows an optional MQTT extension for STA services [i.2]. The IPE subscribes to the MQTT-Broker of the OGC/STA server. The OGC/STA server publishes its new "*Observation*" via the MQTT broker. The IPE creates a <*contentInstance*> using a HTTP request and copies the 'result' attribute of the "*Observation*" to the 'content' attribute of the <*contentInstance*>. The <*container*> may be created beforehand at the hosting CSE where the IPE <*contentInstance*> resources are stored. All interested applications may subscribe to this <*container*> resource.

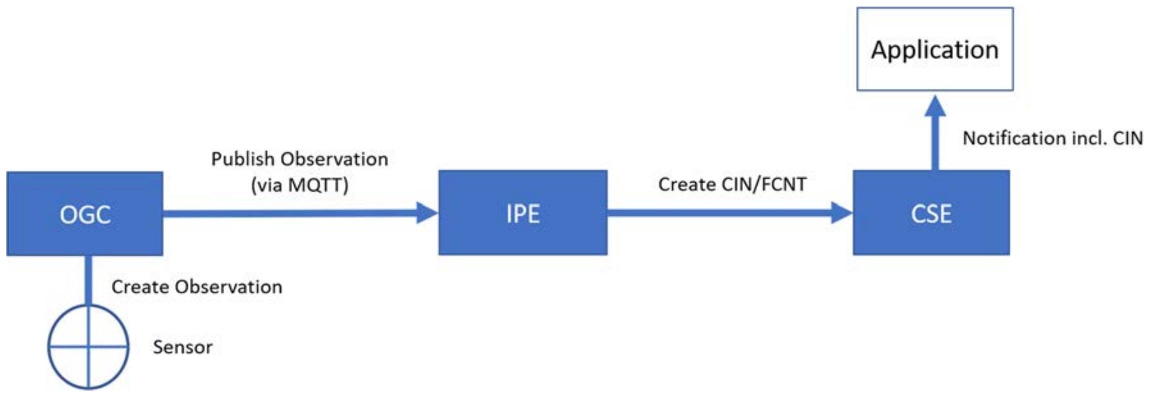


Figure 6.2.2.0-3: Gateway OGC-to-oneM2M direction

This approach is simple and sufficient in cases that only require translating "*Observation*" to <*contentInstances*> and vice-versa.

6.2.2.1 Discussion of "Generic" approach

The "generic" approach has disadvantages:

- Data are stored in the hosting CSE, but this is just a subset of the non-oneM2M proximal IoT function. These are only data that are being actually exchanged.
- The oneM2M client application is not able to gain additional information that are linked to an incoming "*Observation*" like "*Location*" or "*Sensor*". This kind of information would need to be exchanged upfront in the configuration phase described in clause 6.3.

On the other hand, the approach has also advantages:

- The IPE would not be required to copy the full OGC/STA data model into the hosting CSE.

Conclusion:

The "generic" approach would be a very flexible solution, in case only simple measurements need to be exchanged.

6.2.3 "Specific Device" Approach

6.2.3.0 Overview

Another architectural approach would be to focus the design of the IPE on a specific device type.

The data model may be described according to ETSI TS 118 123 [i.5]. ETSI TS 118 123 [i.5] describes a templating tool for describing heterogenous devices and their functionalities using a Smart Device Template (SDT). SDT offers a generic and flexible modeling structure for non-oneM2M devices.

The first step in an OGC/STA interworking scenario may be to register the IPE to the hosting CSE as an <*AE*> resource. This <*AE*> resource is a parent for dedicated <*flexContainer*> resource specializations that represent each "Thing" connected to the OGC/STA server (for example an EV-Charging station).

An OGC/STA "Thing" may be modelled as a SDT Device. Mapping of the SDT Device model to oneM2M resources is performed according to the general mapping procedure described in clause 6.2.2 of ETSI TS 118 123 [i.5]. A SDT Device component is mapped to a specialization of a <*flexContainer*> resource with an associated 'DeviceClass ID' (e.g. "org.onem2m.home.device.tv") *containerDefinition* attribute.

Figure 6.2.3.0-1 shows an example of an OGC/ STA Thing:*[deviceElectricVehicleCharger]*, which is modelled as *a* <*flexContainer*> resource specialization derived from the corresponding SDT Device component. The model of *[deviceElectricVehicleCharger]* follows the schema described in clause 5.5.18 of ETSI TS 118 123 [i.5].

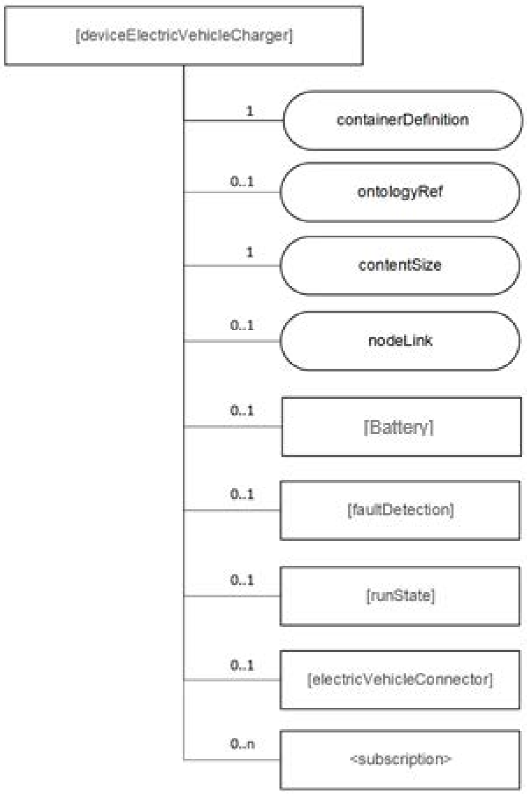


Figure 6.2.3.0-1: [deviceElectricVehicleCharger] example resource representing a OGC/STA device

The <*flexContainer*> representation based on SDTs allow the design of a desired data model for supported device types. The IPE is responsible for ensuring changes in the OGC data model are mapped to an update of the appropriate <*flexContainer*>.

6.2.3.1 Communication Schema

In this approach for the OGC-to-oneM2M direction the IPE subscribes to the MQTT message broker of the OGC/STA server, to receive all desired changes in the data model of an OGC/STA device. In addition to "*Observations*" all changes, such as "*Location*", are published to the IPE (Figure 6.2.3.1-1). The IPE may subscribe or filter out only changes affecting the <*flexContainer*> and sends respective UPDATE messages to the CSE.

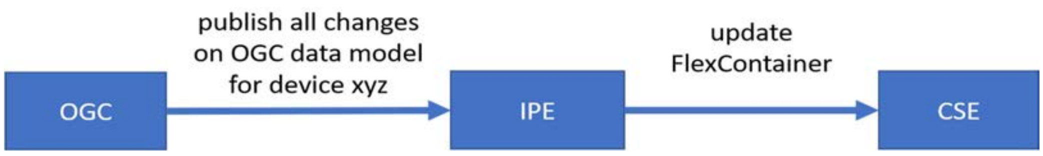


Figure 6.2.3.1-1: OGC/STA-to-oneM2M direction

For the oneM2M-to-OGC direction the IPE subscribes to the <*flexContainer*> resources in the hosting CSE. If there are changes to the <*flexContainer*> from an application the CSE will send a <*notification*> message to the IPE. The IPE assigns the appropriate messages to update the OGC data model (Figure 6.2.3.1-2).

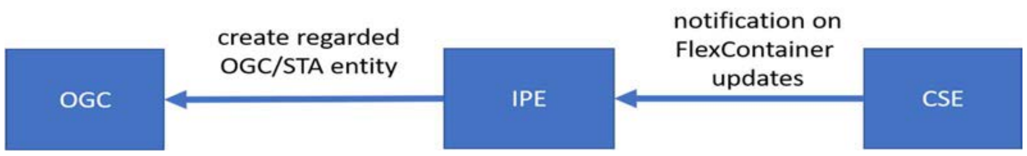

create regarded OGC/STA entity
notification on FlexContainer updates
OGC
IPE
CSE

Figure 6.2.3.1-2: oneM2M-to-OGC/STA direction

6.2.3.2 Discussion of the "Specific Device" Approach

The disadvantage of this approach would be a loss of flexibility and an of course it would involve a certain maintenance effort to keep SDT and OGC model in sync and up to date in case there are changes either on SDT or OGC side. Even if there are tools to create SDT <*flexContainer*> from a certain device automatically, the mapping to or from the OGC/STA data model may still be highly individual because the "properties" field in the OGC/STA data model can be filled with optional data in JSON-Format.

In this case it would be beneficial to have somethings similar for SDTs, a data structure for additional attributes. It is very likely that the SDT would not always describe all features, information and attributes of a complex device. To have something like 'property-extensions' in SDT could ease the process of translating foreign data model into oneM2M. In our OGC/STA example use case "EV-Charging" there is a foreign index like "chargingID" (see Figure 5.2-1) that could be defined as an attribute in a "property" structure. In SDT 3.0 "Properties" were already discussed as an addition to "Action", "Datapoint" and "Event". They could be used for non-functional data as well. An alternative approach for attributes that are not defined by a SDT is the "Label" attribute.

As a consequence, this approach enables no OGC/STA IPE for general use currently. It rather more enables an OGC/STA IPEs for specific devices e.g. "EV-Charging Stations" of "Company XYZ" in "Version 1.23".

However, this approach is also beneficial. Compared to the approaches discussed before, in this case the client application does not need to have any knowledge about the OGC data model. A client application may only rely on oneM2M specifications and is still able to read data coming from a sensor that is connected via OGC/STA.

6.2.4 Conclusion

This clause showed three architecture approaches in the context of full data interworking.

The "Flat data model" is a theoretical approach showing that it creates issues to map the meshed OGC/STA data model to a hierarchical oneM2M data model. In the process of mapping some essential relationships get lost.

In today's oneM2M specifications the maintenance of foreign data model relationships is out of scope for the CSE. This causes potential data inconsistencies. Adding a relationship management to oneM2M would be a beneficial extension.

The "generic" approach focuses mainly on dynamic parts of the OGC data model. Here only "Observations" are translated to oneM2M <*contentInstance*> or <*flexContainer*>. Other parts of the OGC data model are seen as administrative data and are not translated because their change rate is rather low in most use-cases.

A pre-condition for this kind of interworking is the exchange of administrative knowledge during the initial setup phase upfront. The IPE needs to know to which OGC "*Datastream*" or "*Sensor*" a certain "*Observation*" belongs to. Once the "*Observation*" is translated to oneM2M this administrative information is lost.

The advantage of the approach is that the IPE would not be required to copy the full OGC/STA data model into the hosting CSE. The OGC/STA data model would always be accurate, because data remain hosted at the authoritative source, the OGC/STA server. This approach reduces synchronization effort and possible errors.

The "Specific Device" approach uses ETSI TS118 123 [i.5] Smart Device Templates (SDTs) to describe the OGC/STA data model in the hosting CSE. But SDTs look different for various devices. As a consequence, the IPE has to be specific for a certain type of devices.

The disadvantage of this approach would be a loss of flexibility because it enables no OGC/STA IPE for general use currently.

The advantage of this approach is that an oneM2M client application does not need to have any knowledge about the OGC data model. A client application may only rely on oneM2M specifications and is still able to read data coming from a sensor that is connected via OGC/STA.

The "generic" approach was chosen to be investigated and described in more detail in the next chapters. Such an IPE is a "hands-on" solution and a good trade-of between ease of implementation and full data interworking. It is usable in many application fields like "Smart Home" or "Smart City".

6.3 Configuration Aspects

6.3.0 Introduction

This elaborates the initial steps to configure an Interworking Proxy Entity (IPE) between oneM2M and the OGC SensorThings API using the "Generic Approach" according to clause 6.2.2. In order to enable interworking, preparation is needed in both the oneM2M-CSE and the OGC/STA Server (Figure 6.3.0-1). These configuration steps could be initiated manually beforehand or by the IPE at startup time.

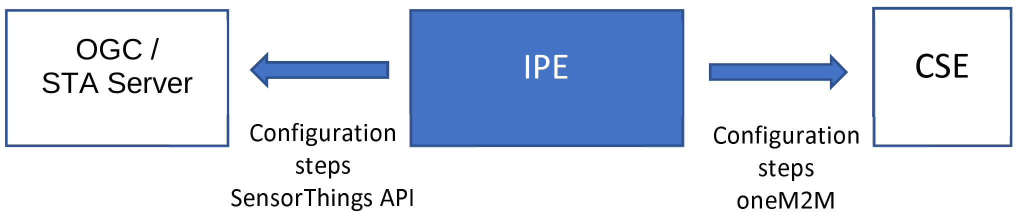

OGC / STA Server
IPE
CSE
Configuration steps SensorThings API
Configuration steps oneM2M

Figure 6.3.0-1: Both sides of the IPE configuration

6.3.1 Data Model mapping

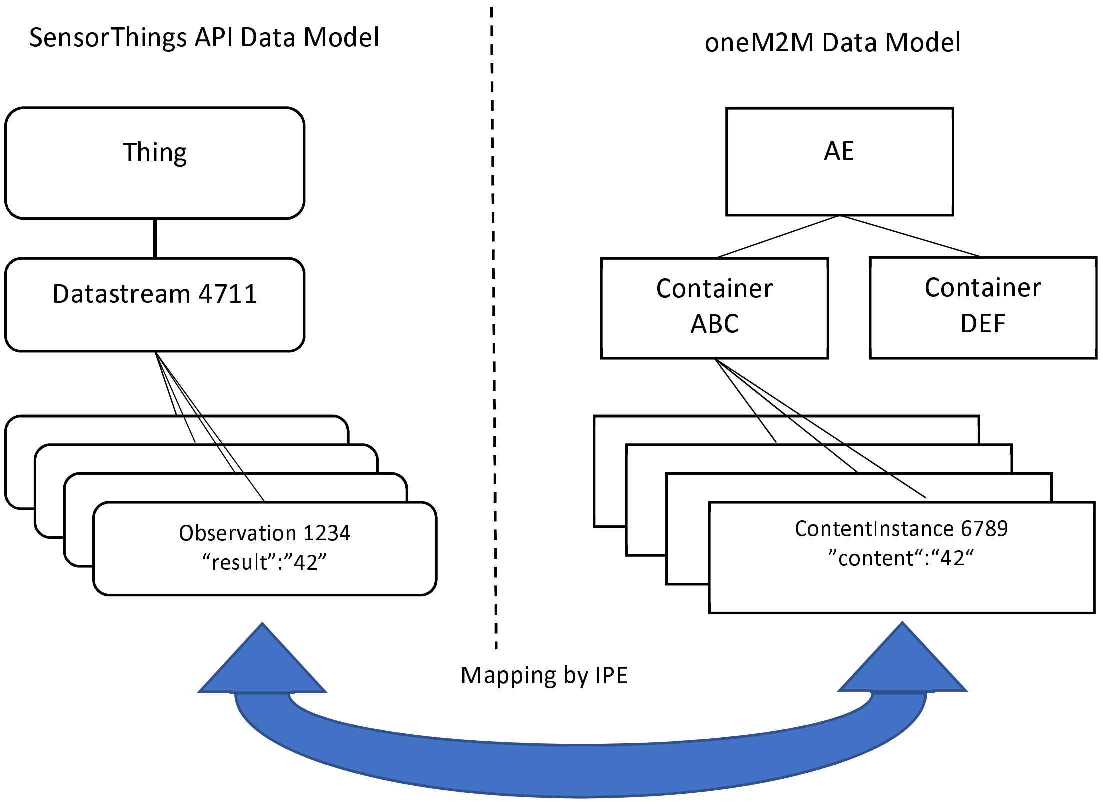

SensorThings API Data Model
oneM2M Data Model
Thing
Datastream 4711
Observation 1234 "result":"42"
AE
Container ABC
Container DEF
ContentInstance 6789 "content":"42"
Mapping by IPE

Figure 6.3.1-1: Data Model mapping in the IPE

As described in the "Generic approach" detailed in clause 6.2.2 the IPE copies the "content" field of a oneM2M <contentInstance> to the "result" field of an STA "Observation" (see Figure 6.2.2.0-2) and the other way around. Certain parts or entities of the data model are important during the mapping process (see Figure 6.3.1-1). In OGC/STA the "Datastream" is the entity containing "Observations". On the other hand, a "Datastream" alone is not defined in OGC/STA. It needs at least a "Thing" which the "Datastream" belongs to. In oneM2M a <contentInstance> is stored in a <container> belonging to another container or finally to an <AE> entity.
The information concerning which OGC/STA "Thing" and its respective "Datastream" is mapped to which <AE> and its respective <container> has to be managed and stored by the IPE e.g. in a configuration file.

6.3.2 Configuration necessary on the OGC/STA Server side

6.3.2.0 Overview

Both directions of the data flow between OGC/STA Server and the IPE need their own configuration steps. A typical OGC/STA server [i.6], has two protocol interfaces, HTTP and MQTT. The HTTP interface can be used for most operational tasks like creating and retrieving entities of the OGC data model [i.3]. The HTTP interface of OGC/STA does not support a publish/subscribe mechanism. To support a publish/subscribe mechanism, OGC/STA has foreseen an additional MQTT Broker, offering parties to subscribe to events they are interested in like e.g. incoming *"Observations"*.

6.3.2.1 Communication direction OGC/STA Server towards IPE

In an exemplary setup (Figure 6.3.2.1-1) a STA Client is connected to an OGC/SensorThings API Server and its data is forwarded to the IPE. The SensorThings Client publishes data to the SensorThings API-Server via a HTTP-Post message. The 'result' attribute of an *"Observation"* contains the sensor data (see Figure 6.2.2.0-2).

An *"Observation"* according to OGC data model [i.3] belongs to a *"DataStream"* (see Figure 5.1-1). This *"DataStream"* has an unique Id like e.g. {"@iot.id": "8715"} and topic name like e.g. {"sta-example-server-address.com/v1.0/Datastreams(8715)"}.

> **Configuration step:** The IPE needs to subscribe to the relevant *"DataStream"* at the MQTT-Broker of the OGC/STA Server using its specific URL. After doing that the IPE receives every *"Observation"* that is pushed to that *"DataStream"*.

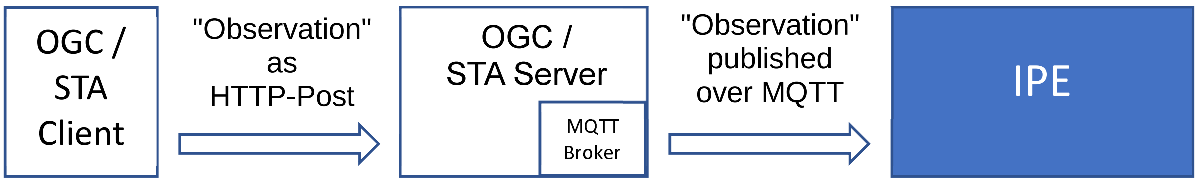


Figure 6.3.2.1-1: Message flow from OGC STA Client to OGC/STA Server to IPE

6.3.2.2 Communication direction IPE towards OGC/STA Server

As described in the "Generic approach" detailed in clause 6.2.2 the IPE receives a *<contentInstance>* included in a *<Notification>* message from the oneM2M CSE. The IPE copies the 'content' attribute of every incoming *<contentInstance>* to the 'result' attribute of a new formed *"Observation"*.

In an exemplary setup (Figure 6.3.2.2-1) the *"Observation"* is posted from the IPE to the OGC/STA Server using HTTP. But before an *"Observation"* can be sent to the OGC/STA Server, the IPE needs to know the destination *"DataStream"*. According to the OGC data model [i.3] a *"DataStream"* needs at least a *"Thing"* entity that it belongs to (see Figure 5.1-1).

> **Configuration steps:** The IPE requires a destination-*"DataStream*" in order to send an *"Observation"*. In case there is no associated *"DataStream"* on the OGC/STA Server yet, because the *"Thing"* does not yet exist, the IPE needs to create a new *"Thing"* and an associated *"DataStream"* on the OGC/STA Server. When a *"DataStream"* is created in the OGC/STA Server the IPE gets back an Id as a reference (e.g. {"@iot.id:3635353"}).
> This reference is later needed to send an *"Observation"* to the related *"DataStream"*. That's why the reference has to be stored when an entity of OGC data model has been created. The IPE might also create additional optional entities of the OGC data model like e.g. *"Location"* or *"Sensor"* when needed.
> The creation of entities like *"DataStream"* and *"Thing"* requires a number of mandatory properties that have to be known at configuration time, e.g. 'name' and 'description'. These property fields might be defined "by hand" or they could be automatically derived e.g. from the "Label" or "ResourceName" property of the regarded oneM2M <AE> or <container> during IPE configuration. The OGC/STA procedures for creating OGC entities are described in SensorThing API documentation [i.7].

When *"Thing"* and *"DataStream"* entities are created the IPE is able to send *"Observations"* to the OGC/STA Server as HTTP POST messages. The interested STA Client can now subscribe to the destination "Datastream" on the MQTT Broker of the OGC/STA Server and thus getting every *"Observation"* forwarded from the IPE.

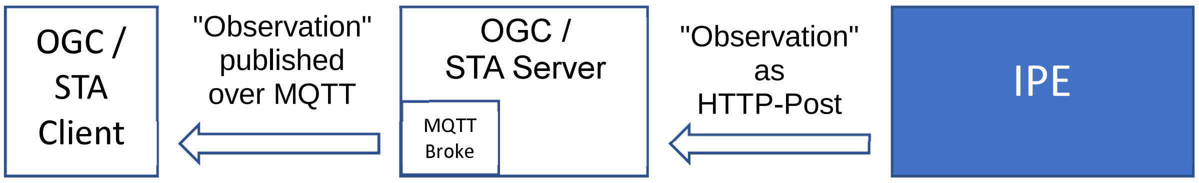


Figure 6.3.2.2-1: Message flow from IPE to OGC/STA Server to OGC Client

6.3.3 Configuration of the oneM2M CSE

6.3.3.0 Overview

The IPE needs also to perform configuration steps at the hosting CSE.

6.3.3.1 Communication direction oneM2M CSE towards IPE

In an exemplary setup (Figure 6.3.3.1-1) a oneM2M AE sends a data point to the CSE by creating a <*contentInstance*> under a certain <*container*> that belongs to a certain <*AE*>. The IPE can set a <*subscription*> to this <*container*> and gets a <*notification*> message along with a <*contentInstance*>, when a new data point arrives.

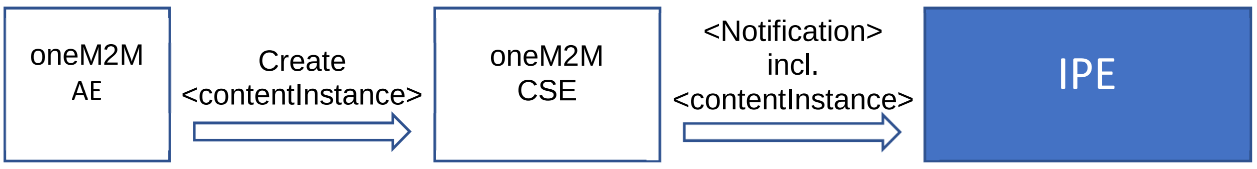


Figure 6.3.3.1-1: Data message flow from AE to CSE to IPE

Configuration step: The IPE needs to set a <*subscription*> to the <*container*> that holds data that should be forwarded to OGC/STA side.

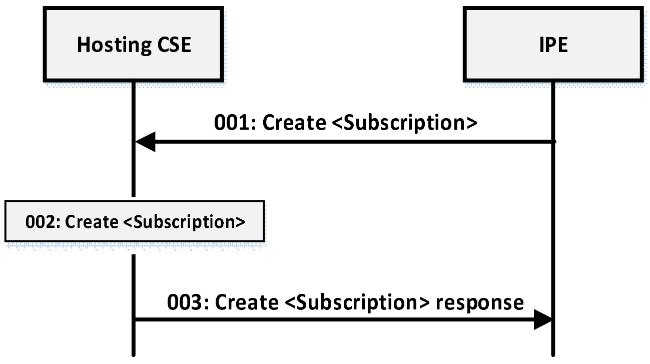

Hosting CSE
IPE
001: Create <Subscription>
002: Create <Subscription>
003: Create <Subscription> response

Figure 6.3.3.1-2: Configuration message flow for CSE-to-IPE direction

The detailed configuration messages are shown in Figure 6.3.3.1-2:

1) The IPE creates a <*subscription*> to the <*container*> that is appointed to hold data to be forwarded to the OGC/STA side.

2) The Hosting CSE evaluates the request, performs the appropriate checks, and creates the <*subscription*> resource.

3) Hosting CSE responds with the successful result of <*subscription*> resource creation, otherwise it responds with an error.

6.3.3.2 Communication direction IPE towards CSE

In an exemplary setup (Figure 6.3.3.2-1) a oneM2M AE creates a <*subscription*> to the <*container*> that is being used by the IPE. Subsequently the AE gets a <*notification*> along with data contained in a <*contentInstance*> every time the IPE creates a <*contentInstance*> in that <*container*>.

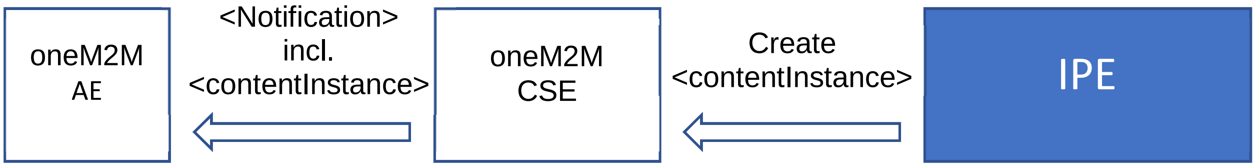


Figure 6.3.3.2-1: Data message flow from IPE to CSE to AE

As described in the "Generic approach" detailed in clause 6.2.2", the IPE copies the 'result' attribute of every incoming "*Observation*" from OGC/STA side to the '*content*' attribute of a new <*contentInstance*>. But before the IPE is able to create a <*contentInstance*> it needs to know the destination <*container*> *on their hosting CSE. If this <container> does not exist, it has to be created. The <container> might belong to an existing <AE> or a new <AE> has to be created.*

Configuration steps: In case there is no container available, the IPE needs to create a <*container*> *and if necessary, also a dedicated* <*AE*> on the hosting CSE that are appointed to be used as destination for data coming from the OGC/STA side.

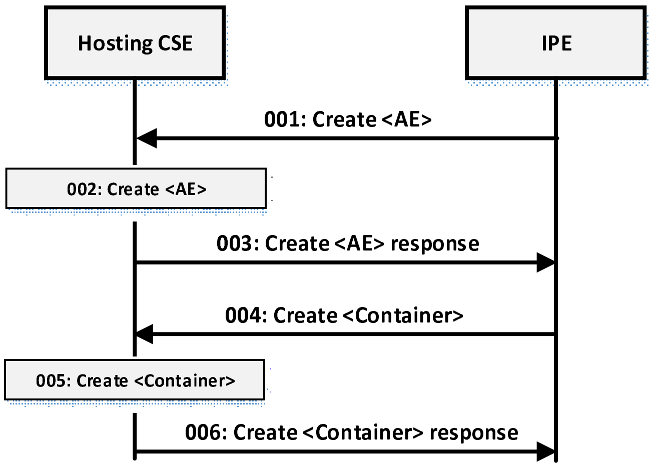


Figure 6.3.3.2-2: Configuration message flow between IPE and CSE

The detailed configuration messages are shown in Figure 6.3.3.2-2:

1) The IPE requests to create an <*AE*> resource on the Hosting CSE.

2) The Hosting CSE evaluates the request, performs the appropriate checks, and creates the <AE> resource.

3) Hosting CSE responds with the successful result of <*AE*> resource creation, otherwise it responds with an error.

4) The IPE requests to create a <container> under the <*AE*>.

5) The Hosting CSE evaluates the requests, performs the appropriate checks, and creates the <*container*> resources.

6) Hosting CSE responds with the successful result of <*container*> resource creation, otherwise it responds with an error.

6.3.4 Finding data sources

6.3.4.0 Introduction

There might be situations when just certain groups of sensor data should be transferred from STA domain to oneM2M using the IPE. Then it is helpful to use filter mechanisms in order to identify relevant data source.

6.3.4.1 Filtering using OGC/STA protocol

In a typical application field like e.g. in Smart City it might be necessary to get data from a group of sensors and send it to the IPE. In a exemplary setup (Figure 6.3.4.1-1) status data of all EV-Charging stations in town should be forwarded to the oneM2M side using the IPE. A typical OGC/STA-based Smart City platform might host data from many different sources, like environmental- and weather sensors, streetlights and many more. The challenge in this case is to identify sensors belonging to the group of EV-Charging Station in the OGC/STA server automatically.

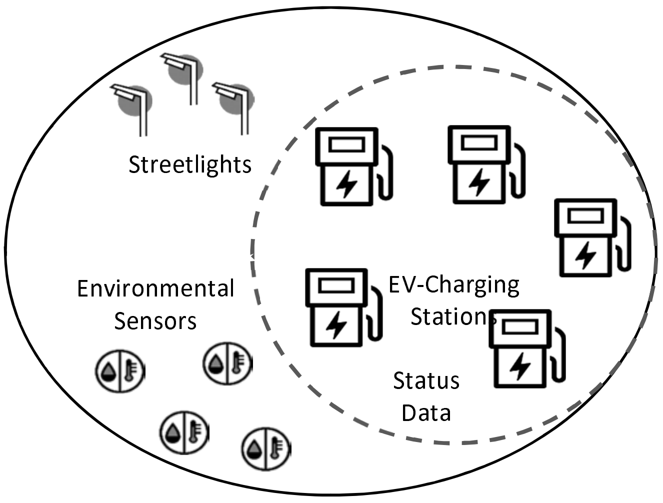


Figure 6.3.4.1-1: Identifying a group of data among others

The SensorThings API protocol defines sophisticated discovery and filter mechanisms. It has a $filter query option combined with operators and functions [i.7]. Thus the IPE could identify all "Things" e.g. having the word "Charging" in its "name" property. Also spatial requests are possible if "Things" have associated geographic locations described.

When the IPE has identified the relevant "Things" it needs also to ask for their related "Datastreams". Thanks to an "expand" option, all necessary data can be retrieved within one request. The SensorThings API $expand query enables to retrieve a "Thing" inline with related entities like "Datastream" and "Observation".

The answer contains a list of all EV-Charging "Things" and related "Datastreams" including their Ids. The Ids have to be extracted from the request answer. They are used for the subscription at the MQTT broker of the OGC/STA server. Subsequently the IPE gets every status change of an EV-Charging station as a new "Observation" that can be forwarded to the oneM2M side. An example discovery and filter request and the regarded answer is shown in clause 5.2.

NOTE: It might be useful that the IPE repeats the request after a certain time period, because there might be new "Things" added or others disappear.

6.4 Operational Aspects

6.4.1 Using the IPE for more than one data source

Earlier clauses in the present document describe the mapping of exactly one data source from oneM2M (<*AE*>/<*container*>) to one OGC/STA (*"DataStream"*) and the opposite direction. But the IPE could also be used for the mapping of several data sources at once.

To handle this the IPE might create more complex entity structures in the hosting CSE. There might be groups or trees of <AE> and <*container*> where a single <*container*> is dedicated to a certain *"DataStream"*.

It might also be necessary to create several *"Things"* and *"DataStreams"* at the OGC/STA Server in order to map and distinguish several types of sensor data received from the oneM2M side.

NOTE: The storage and management of mapping information for many data sources might lead to more complex configuration of the IPE. An alternative could be to run several IPEs in parallel with a simpler configuration.

6.4.2 Check for existing configuration

The configuration step described in clause 6.3 have to be completed before an IPE is able to operate properly. Usually this is done at start-up time. So, when an IPE is started it creates all required entities on the OGC/STA Server and the hosting CSE.

In a typical state-of-the-art cloud environment, the IPE runs as a service in a so called "container" package (not to be confused with oneM2M <*container*>). The "containerized" IPE runs among other application in the cloud. All applications are usually orchestrated and managed by container orchestration service.
There might be situations where the orchestration service restarts the IPE service. This might happen because of temporarily lack of memory in the virtual machine, short time over load, miss-configuration of the cluster or many other reasons.

As a consequence, the IPE may be restarted from time to time. When this happens some, or all, of the required entities may already exist on the OGC /STA server and Hosting CSE. An IPE, when it restarts, should tolerate this situation and reuse these required entities if they already exist, only creating new ones if they do not.

6.5 Security Considerations

6.5.0 Introduction

Every API in the open internet is prone to security attacks. Like most Web APIs the SensorThings API IPE might be designed, implemented, and operated respecting API Security Best Practice for example described in OWASP Secure Coding Practices Quick Reference Guide [i.9]. The intention of this clause is not a comprehensive security analysis. Rather more it aims to lay out the IPE setup and to show regarded security aspects.

From an architectural point of view the IPE is located between an OGC STA Server and a oneM2M CSE (see Figure 6.5.0-1).

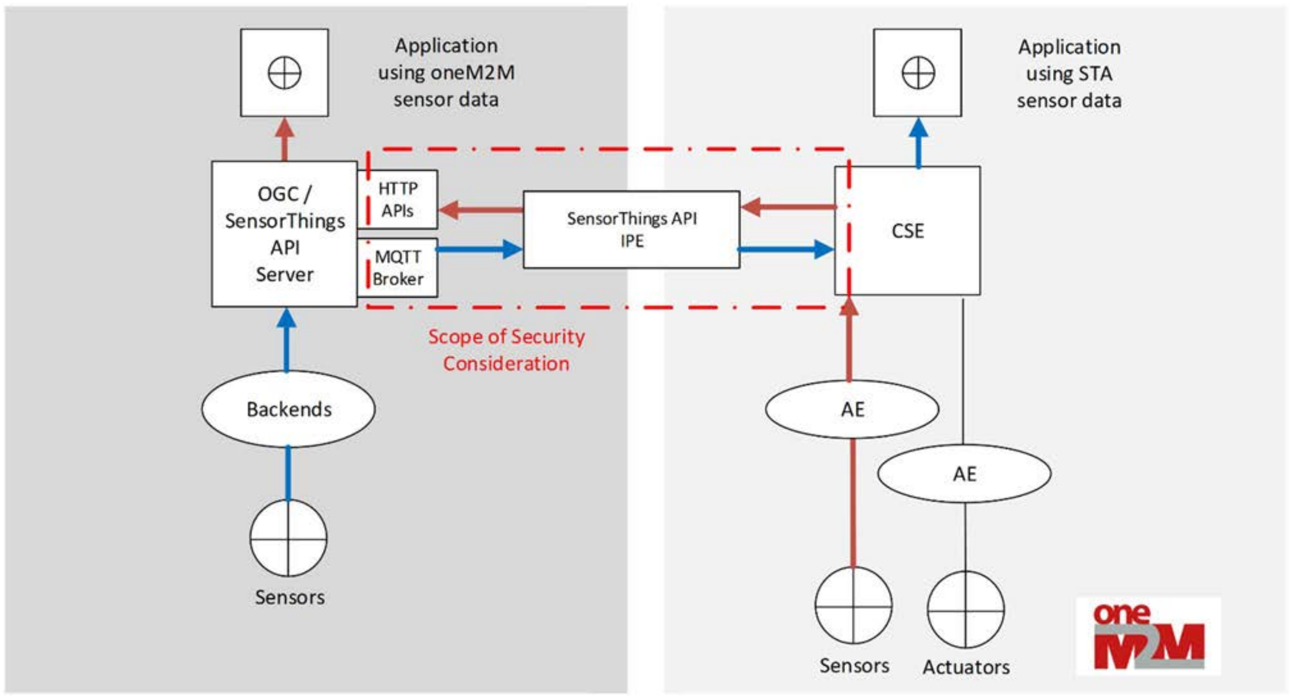

Application using oneM2M sensor data
Application using STA sensor data
OGC / SensorThings API Server
HTTP APIs
MQTT Broker
SensorThings API IPE
CSE
Scope of Security Consideration
Backends
Sensors
AE
AE
Sensors
Actuators

Figure 6.5.0-1: Scope of the Security Considerations

There might be situations where OGC/STA Server and CSE belong to different security domains (e.g. different company networks). The IPE could be either located in one of the networks or it could be a separate service between two networks operated by a third-party service provider.

The IPE has three different protocol APIs:

1) IPE and CSE using oneM2M over HTTP for both communication directions. Other protocol bindings than HTTP are specified in oneM2M but are not discussed in this clause.

2) Communication between IPE and OGC/STA Server is based on two different APIs:

   a) Sensor Things API over HTTP is used to push data to the OGC/STA Server described in clause 6.3.2.2.

   b) The IPE subscribes to changes in the data model of the OGC/STA Server via MQTT as described in clause 6.3.2.1.

Communication relationships between OGC/STA Server, IPE and CSE are static. Each instance knows exactly its communication partner(s) beforehand. There is no dynamic change of communication endpoints.

The level of security measurements depends on the concrete operational situation. The following measurements are recommended:

- Mutual authentication of endpoints makes sure that only legitimate resources communicate with each other.
- Encryption ensures confidentiality of communication between endpoints.
- Authorization ensures that OGC/STA Server, IPE and CSE have the appropriate permissions to access a particular resource and not any other even if one of the three entities is under control of an attacker.
- Throttling limits, the number of API requests in a certain period. This protects an API from "denial of service attacks" where an API is overwhelmed with requests.
- Input validation is preventing malformed requests or data from persisting in the database and triggering malfunction in one of the entities.

While most of the recommendations require additional services or tools there are protocol inherent functions for authentication and authorization available.

6.5.1 Mutual authentication for different protocol interfaces

Most cloud environments allow for configuration of IP address filter. This way only preconfigured instances with pre-known IP addresses are entitled to communicate with each other. A filter table might be created that allows only data traffic between IP addresses of the OGC/STA Server, IPE and CSE.

Direct VPN connections might be established between the endpoints to protect the communication.

If these kind of "additional" security measurements are not applicable there are protocol specific ways for authentication in HTTP, oneM2M and MQTT.

Mutual Authentication for HTTP APIs

One possibility for mutual authentication between OGC/STA Server, IPE and CSE is to share tokens, certificates, passwords or keys beforehand. Appropriate artifacts might be placed in the "Authorization header" of every HTTP-Request for authentication during operation of the IPE.

Mutual Authentication in oneM2M

oneM2M TS-0003 Security Solutions offers a Security Association Establishment Framework (SAEF) [i.8] and describes possibilities for mutual authentication between the CSE and the IPE. There are two methods described in detail to exchange credentials using "Provisioned Symmetric Key"- or "Certificate-Based Security Association".

Mutual Authentication in MQTT

Basic functionalities like authentication are described in MQTT standard [i.10]. Mutual authentication is described at section "enhanced Authentication". MQTT enables client authentication. In the architectural setup the IPE is the client that has to be authenticated at the MQTT Broker of the SensorThings API Server.

6.5.2 Authorization

oneM2M has the concept of access policies described in the authorization clause of oneM2M TS-0003 Security Solutions [i.8]. An access policy might be attached to a resource (e.g. a container) and privileges define what action an authenticated requester is entitled to perform.

oneM2M authorization might be used to restrict the IPE privileges in the CSE. This makes sure that the IPE cannot be abused to get unwanted access to data in the CSE database.

SensorThings API does not specify any authorization functionalities.

NOTE: External services like API gateways might also provide authorization services by defining a set of operations that is allowed to certain resources.

6.5.3 Additional Security Services and Tools

Beside protocol inherent security measurements additional services might be used extensively to protect all involved APIs. These services could be for example API Gateways, API Filters, Intrusion Detection Systems, Firewalls and VPN Gateways that are able to provide input validation, rate limiting, authentication, authorization, encryption of communication and more.

History

Document history		
V5.0.0	December 2023	Publication