|  |
| --- |
|  |

|  |  |
| --- | --- |
| CHANGE REQUEST | |
| Meeting ID:\* | SDS 54 |
| Source:\* | Andreas Neubacher, Deutsche Telekom, [andreas.neubacher@magenta.at](mailto:andreas.neubacher@magenta.at)  Ingo Friese, Deutsche Telekom, [Ingo.Friese@telekom.de](mailto:Ingo.Friese@telekom.de)  Andreas Kraft, Deutsche Telekom, a.kraft@telekom.de |
| Date:\* | 2022-04-04 |
| Reason for Change/s:\* | This CR adds content w.r.t. configuration aspects of an IPE using the “Generic Approach” according to Clause 6.2.2 in TR-0065 |
| CR against: Release\* | Rel 5 |
| CR against: WI\* | Active <Work Item number>  MNT maintenance / < Work Item number(optional)>  Is this a mirror CR? Yes  No  mirror CR number: (Note to Rapporteur - use latest agreed revision)  STE Small Technical Enhancements / < Work Item number (optional)>  Only ONE of the above shall be ticked |
| CR against: TS/TR\* | TR-065 and V0.1.0 |
| Clauses \* |  |
| Type of change: \* | Editorial change  Bug Fix or Correction  Change to existing feature or functionality  New feature or functionality  Only ONE of the above shall be ticked |
| Other TS/TR(s) impacted | none |
| Post Freeze checking:\* | This CR contains only essential changes and corrections? YES  NO  This CR may break backwards compatibility with the last approved version of the TS? YES  NO |
| Template Version: January 2020 (do not modify) | |

**oneM2M Notice**

The document to which this cover statement is attached is submitted to oneM2M. Participation in, or attendance at, any activity of oneM2M, constitutes acceptance of and agreement to be bound by terms of the Working Procedures and the Partnership Agreement, including the Intellectual Property Rights (IPR) Principles Governing oneM2M Work found in Annex 1 of the Partnership Agreement.

## Introduction

During SDS#51.2, document [SDS-2021-0224](https://member.onem2m.org/Application/documentApp/documentinfo/?documentId=34122&fromList=Y) was aggreed as input to “TR-0065-oneM2M\_SensorThings\_API\_interworking-V0\_1\_0”. The aggreed next steps were to add content w.r.t to configuration aspects of the selected “Generic Approach” according to Clause 6.2.2 in TR-0065. The following changes reflect the expected contented with regards to IPE configuration.

### -----------------------Start of Changes to References Clause -------------

## 2.2 Informative references

Clause 2.2 shall only contain informative references which are cited in the document itself.

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 (<http://member.onem2m.org/Static_pages/Others/Rules_Pages/oneM2M-Drafting-Rules-V1_0.doc>)

[i.2] SensorThings API website, https://www.ogc.org/standards/sensorthings

[i.3] ISO 19156 website, <https://www.iso.org/standard/32574.html>

[i.4] oneM2M TS-0033 3.0.0 "Interworking Framework“

[i.5] oneM2M TS-0023 4.7.1 “SDT based Information Model and Mapping for Vertical Industries”

[i.6] [FROST®-Server - Open-Source-Implementierung der OGC SensorThings API - Fraunhofer IOSB](https://www.iosb.fraunhofer.de/de/projekte-produkte/frostserver.html)

[i.7] OGC and ISO 19156:2001, OGC 10-004r3 and ISO 19156:2011(E), OGC Abstract Specification: Geographic  
 information — Observations and Measurements. Available Online:  
 <https://www.ogc.org/standards/om>

[i.8] API documentation for the Open Geospatial Consortium (OGC) SensorThings international standard  
<https://developers.sensorup.com/docs/#introduction>

[i.9] Kubernets website - Production-Grade Container Orchestration; <https://kubernetes.io/>

[i.10] OASIS Website MQTT https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

### -----------------------End of Changes to References -------------

### -----------------------Start of change 1-------------------------------------------

### 6.2.4 Conclusion

This chapter 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 TS-0023 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*

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 6.3-1). These configuration steps could be initiated manually beforehand or by the IPE at startup time.

IPE

OGC /  
STA Server

CSE

Configuration  
steps  
SensorThings API

Configuration  
steps  
oneM2M

Figure 6.3-1: Both sides of the IPE configuration

6.3.1 Data Model mapping

As described in the “Generic approach” according to Clause 6.2.2 the IPE copies the “content” field of an 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 6.3.1.1-1). 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” where 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 which OGC / STA “Thing” with the regarded “Datastream” is mapped to which <AE> and its regarded <container> has to be managed and stored by the IPE e.g. in a configuration file.

Container  
ABC

SensorThings API Data Model

ContentInstance 4711

Container  
DEF

AE

oneM2M Data Model

Mapping by IPE

Thing

Datastream 4711

Observation 1234

Observation 1234

Observation 1234

Observation 1234

„result“:“42“

ContentInstance 4711

ContentInstance 4711

ContentInstance 6789

„content“:“42“

Figure 6.3.1.0-1: Data Model mapping in the IPE

### 6.3.2 Configuration necessary on the OGC / STA Server side

Both directions of the data flow between OGC / STA Server and the IPE need their own configuration steps. A typical OGC / STA server, like for example the Open Source FROST-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.7]. The HTTP interface of OGC / STA does not support publish- / subscribe mechanism. To support 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 examplary setup (Figure 6.3.2.1.-1) a STA Client is connected to an OGC/SensorThings API Server and its data shall be forwared 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.7] belongs to a *“DataStream”* (see Figure 5.1-1). This *“DataStream”* has an unique Id like e.g. {"@iot.id”: “8715”} and URL like e.g. {“sta-example-server-address.com/v1.0/Datastreams(8715)”}.

**Configuration step:** The IPE needs to subscribe to the regarded *“DataStream”* determined by its Id/URL at the MQTT-Broker of the OGC / STA Server. In doing that the IPE receives every *“Observation”* that is pushed to that *“DataStream”.*

IPE

OGC / STA  
Client

MQTT  
Broker

“Observation“  
as  
HTTP-Post

“Observation“  
published  
over MQTT

OGC /  
STA Server

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” according to 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 regarded *“DataStream”*. According to the OGC data model [i.7] 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”* doesn’t yet exists, the IPE needs to create a new *“Thing”*and an associated *“DataStream”* on the OGC / STA Server. When a *“Thing”* and 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 needed to attach a *“DataStream”* to a *“Thing”* as well as to send an *“Observation”* to a dedicated *“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 be known at configuration time, eg. ‘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.8]

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 regarded “Datastream” on the MQTT Broker of the OGC/STA Server and thus getting every *“Observation”* forwarded from the IPE.

OGC /  
STA Server

IPE

OGC / STA  
Client

MQTT  
Broker

“Observation“  
as  
HTTP-Post

“Observation“  
published  
over MQTT

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

The IPE needs also to proceed configuration steps at the hostig CSE.

#### 6.3.3.1.Communication direction oneM2M CSE towards IPE

In an examplary setup (Figure 6.3.3.1.-1) a oneM2M AE sends data 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 new data arrive.

IPE

oneM2M  
CSE

oneM2M  
AE

<Notification>  
incl.  
<contentInstance>

Create  
<contentInstance>

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.



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 *<Subscription>* by testing the existence of the notification endpoint at IPE
3. The Hosting CSE evaluates the requests, performs the appropriate checks, and creates the *<Subscription>* resources.
4. 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 examplary setup (Figure 6.3.3.2.-1) a oneM2M AE may set a *<subscription>* to the IPE regarded *<container> on the hosting CSE* . Subsequently the AE gets a *<notification>* along with data contained in a *<contentInstance>* everytime the IPE creates a  *<contentInstance>* at the CSE.

IPE

oneM2M  
CSE

oneM2M  
AE

<Notification>  
incl.  
<contentInstance>

Create  
<contentInstance>

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

As described in 6.2.2 “Generic approach” 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 send a *<contentInstance>* it needs either to know or to create the destination *<AE>* and *<container> on their hosting CSE.*

**Configuration steps:** If not exist the IPE needs to create a *<AE>* and *<container>* 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

There might be situations when just certain groups of sensor data should be transferred from STA domain to oneM2M using the IPE. Then its 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.

Status  
 Data



Streetlights



Environmental  
Sensors



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 combiend with operators and functions[i..8]. Thus the IPE could identify all “Things” e.g. having the word “Charging” in thier “name” property. Also spatial requests are possible if a “Things” has associated geographic locations described.

When the IPE has identified the relevant "Things” it needs also to ask for the regarded “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 enities like “Datastream” and “Observation”.

The answer contains a list of all EV-Charging “Things” and regarded “Datestreams” including their Ids. The Ids have do 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 5.2.

**Note:** It might be useful that the IPE repeats the request after a certain time periode because there might be new “Things” added or others disappear.

#### 6.3.4.2 Filtering using MQTT protocol

Automatic filtering of certain data sources can also be done with MQTT mechanisms. The IPE might subscribe to a certain URL at the MQTT broker e.g. {“v1.0/Observations”} or {“v1.0/Locations”} and gets all incoming updates of that kind of entity. MQTT also defines wildcard character [i..10] to filter certain topics at a MQTT Broker. The multi-level wildcard “#” might be used to identify the parent and any number of child levels in the hierarchy of MQTT topics. The IPE could subscribe to “#”. Via such a subscription the IPE does receive all topics being pushed to the MQTT broker of the OGC / STA Server, and may build up a datastructure of all available *“DataStreams”* determined by its Id.

## *6.4 Operational Aspects*

For the implementation and operation of an OGC / STA Interworking Proxy Entity it is important to be aware of operational aspects.

### 6.4.1 Using the IPE for more than one data source

This document describes in the chapters before the mapping of exact one data source from oneM2M (*<AE> / <Container>*) to OGC / STA (*“DataStream”*) and the opposite direction. But the IPE could also be used for the mapping of several data sources at once.

In preparation 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 be also necessary to create several *“Things”* and regarded *“DataStreams”* at the OGC / STA Server in order to map and distinguish several 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 more simple configuration.

### 6.4.2 Check for existing configuration

The configuration step described in 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 like e.g. Kubernets [i.9].  
There might be situations where the orchestration service restarts the IPE service. This might happen because of temporarly 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 and it should check the existence of required entites in the OGC / STA Server and in the hosting CSE before in creates new ones. Otherwise the IPE creates new data structures with every restart.

### -----------------------End of change 1---------------------------------------------

CHECK LIST

* Does this Change Request include an informative introduction containing the problem(s) being solved, and a summary list of proposals.?
* Does this CR contain changes related to only one particular issue/problem?
* Have any mirror CRs been posted?
* Does this Change Request make **all** the changes necessary to address the issue or problem? E.g. A change impacting 5 tables should not include a proposal to change only 3 tables?Does this Change Request follow the drafting rules?
* Are all pictures editable?
* Have you checked the spelling and grammar?
* Have you used change bars for all modifications?
* Does the change include the current and surrounding clauses to clearly show where a change is located and to provide technical context of the proposed change? (Additions of complete clauses need not show surrounding clauses as long as the proposed clause number clearly shows where the new clause is proposed to be located.)
* Are multiple changes in this CR clearly separated by horizontal lines with embedded text such as, start of change 1, end of change 1, start of new clause, end of new clause.?