

**ITU-T**

TELECOMMUNICATION  
STANDARDIZATION SECTOR  
OF ITU

**Y.4473**

(08/2020)

SERIES Y: GLOBAL INFORMATION  
INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS,  
NEXT-GENERATION NETWORKS, INTERNET OF  
THINGS AND SMART CITIES

Internet of things and smart cities and communities –  
Frameworks, architectures and protocols

---

## **SensorThings API – Sensing**

Recommendation ITU-T Y.4473

ITU-T Y-SERIES RECOMMENDATIONS

GLOBAL INFORMATION INFRASTRUCTURE, INTERNET PROTOCOL ASPECTS, NEXT-GENERATION NETWORKS, INTERNET OF THINGS AND SMART CITIES

GLOBAL INFORMATION INFRASTRUCTURE	
General	Y.100–Y.199
Services, applications and middleware	Y.200–Y.299
Network aspects	Y.300–Y.399
Interfaces and protocols	Y.400–Y.499
Numbering, addressing and naming	Y.500–Y.599
Operation, administration and maintenance	Y.600–Y.699
Security	Y.700–Y.799
Performances	Y.800–Y.899
INTERNET PROTOCOL ASPECTS	
General	Y.1000–Y.1099
Services and applications	Y.1100–Y.1199
Architecture, access, network capabilities and resource management	Y.1200–Y.1299
Transport	Y.1300–Y.1399
Interworking	Y.1400–Y.1499
Quality of service and network performance	Y.1500–Y.1599
Signalling	Y.1600–Y.1699
Operation, administration and maintenance	Y.1700–Y.1799
Charging	Y.1800–Y.1899
IPTV over NGN	Y.1900–Y.1999
NEXT GENERATION NETWORKS	
Frameworks and functional architecture models	Y.2000–Y.2099
Quality of Service and performance	Y.2100–Y.2199
Service aspects: Service capabilities and service architecture	Y.2200–Y.2249
Service aspects: Interoperability of services and networks in NGN	Y.2250–Y.2299
Enhancements to NGN	Y.2300–Y.2399
Network management	Y.2400–Y.2499
Network control architectures and protocols	Y.2500–Y.2599
Packet-based Networks	Y.2600–Y.2699
Security	Y.2700–Y.2799
Generalized mobility	Y.2800–Y.2899
Carrier grade open environment	Y.2900–Y.2999
FUTURE NETWORKS	Y.3000–Y.3499
CLOUD COMPUTING	Y.3500–Y.3599
BIG DATA	Y.3600–Y.3799
QUANTUM KEY DISTRIBUTION NETWORKS	Y.3800–Y.3999
INTERNET OF THINGS AND SMART CITIES AND COMMUNITIES	
General	Y.4000–Y.4049
Definitions and terminologies	Y.4050–Y.4099
Requirements and use cases	Y.4100–Y.4249
Infrastructure, connectivity and networks	Y.4250–Y.4399
<b>Frameworks, architectures and protocols</b>	<b>Y.4400–Y.4549</b>
Services, applications, computation and data processing	Y.4550–Y.4699
Management, control and performance	Y.4700–Y.4799
Identification and security	Y.4800–Y.4899
Evaluation and assessment	Y.4900–Y.4999

For further details, please refer to the list of ITU-T Recommendations.

# Recommendation ITU-T Y.4473

## SensorThings API – Sensing

### Summary

Recommendation ITU-T Y.4473 specifies the SensorThings application programming interface (API) which provides an open standard-based and geospatial-enabled framework to interconnect Internet of things (IoT) devices, data and applications over the Web.

The SensorThings API is an open standard, and that means it is non-proprietary, platform-independent. It builds on a rich set of proven-working and widely-adopted open standards, such as the web protocols and the Open Geospatial Consortium (OGC) sensor web enablement (SWE) standards, including the ISO/OGC observation and measurement data model. The SensorThings API is extensible and can be applied to not only simple but also complex use cases.

This Recommendation provides a standard way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems. The SensorThings API uses representational state transfer (REST) principles, an efficient JavaScript object notation (JSON) encoding, message queuing telemetry transport (MQTT) protocol, flexible OASIS open data protocol (OData) and uniform resource locator (URL) conventions.

### History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T Y.4473	2020-08-29	20	<a href="http://handle.itu.int/11.1002/1000/14375">11.1002/1000/14375</a>

### Keywords

Internet of things, sensor web, smart cities, smart communities.

---

\* To access the Recommendation, type the URL <http://handle.itu.int/> in the address field of your web browser, followed by the Recommendation's unique ID. For example, <http://handle.itu.int/11.1002/1000/11830-en>.

## FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

## NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

## INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at <http://www.itu.int/ITU-T/ipr/>.

© ITU 2020

All rights reserved. No part of this publication may be reproduced, by any means whatsoever, without the prior written permission of ITU.

## Table of Contents

	<b>Page</b>
1	Scope ..... 1
2	References..... 1
3	Definitions ..... 2
3.1	Terms defined elsewhere ..... 2
3.2	Terms defined in this Recommendation..... 2
4	Abbreviations and acronyms ..... 3
5	Conventions ..... 3
6	The SensorThings API entities ..... 3
6.1	Common control information ..... 3
6.2	The sensing entities ..... 4
7	SensorThings service interface ..... 18
7.1	Common control information ..... 18
7.2	Resource path ..... 19
7.3	Requesting data ..... 23
8	SensorThings sensing create-update-delete ..... 31
8.1	Overview ..... 31
8.2	Create an entity ..... 31
8.3	Update an entity ..... 33
8.4	Delete an entity ..... 34
9	Batch requests ..... 35
9.1	Introduction ..... 35
9.2	Batch-processing request..... 35
9.3	Batch-processing response ..... 38
9.4	Asynchronous batch requests ..... 39
10	SensorThings MultiDatastream extension ..... 40
11	SensorThings Data Array Extension ..... 45
11.1	Retrieve a Datastream's Observation entities in dataArray ..... 45
11.2	Create Observation entities with dataArray ..... 48
12	SensorThings Sensing MQTT Extension ..... 50
12.1	Create a SensorThings Observation with MQTT Publish..... 50
12.2	Receive updates with MQTT Subscribe..... 50
	Bibliography..... 53



# Recommendation ITU-T Y.4473

## SensorThings API – Sensing

### 1 Scope

This Recommendation specifies a SensorThings application programming interface (API) which provides an open standard-based and geospatial-enabled framework to interconnect Internet of things devices, data, and applications over the Web.

The scope of this Recommendation includes:

- SensorThings API entities and relevant properties, including thing, location, datastream, sensor, observations, etc.;
- SensorThings service interface for requesting data;
- SensorThings sensing operations, including reading, creating, updating and deleting;
- SensorThings batch processing requests and relevant responses, and extensions for multidatastream, data array and Message Queuing Telemetry Transport (MQTT).

### 2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

- [ITU-T Y.4000] Recommendation ITU-T Y.4000/Y.2060 (2012), *Overview of Internet of things*.
- [IETF RFC 2046] IETF RFC 2046, *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*.  
<https://www.ietf.org/rfc/rfc2046.txt>
- [IETF RFC 2616] IETF RFC 2616, *Hypertext Transfer Protocol -- HTTP/1.1*.  
<https://www.ietf.org/rfc/rfc2616.txt>
- [IETF RFC 5789] IETF RFC 5789, *PATCH Method for HTTP*,  
<https://www.ietf.org/rfc/rfc5789.txt>
- [IETF RFC 6902] IETF RFC 6902, *JavaScript Object Notation (JSON) Patch*.  
<https://www.ietf.org/rfc/rfc6902.txt>
- [IETF RFC 7946] IETF RFC 7946, *The GeoJSON Format*.  
<https://www.ietf.org/rfc/rfc7946.txt>
- [ISO 8601] ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times*.
- [OASIS OData ABNF] OASIS OData Version 4.0 (2017), *ABNF Construction Rules*.  
<http://docs.oasis-open.org/odata/odata/v4.01/cs01/abnf/odata-abnf-construction-rules.txt>
- [OASIS OData JSON] OASIS OData Version 4.0 (2016), *JSON Format Version 4.0 Plus Errata 03*.  
<http://docs.oasis-open.org/odata/odata-json-format/v4.0/odata-json-format-v4.0.html>
- [OASIS OData Part1] OASIS OData Version 4.0 (2016), *Part 1: Protocol Plus Errata 03*.  
<http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html>

- [OASIS OData Part2] OASIS OData Version 4.0 (2016), *Part 2: URL Conventions Plus Errata 03*.  
<http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html>
- [OGC/ISO 19156] OGC/ISO 19156 (2011), *Abstract Specification Topic 20: Geographic information – Observations and Measurements*.  
[http://portal.opengeospatial.org/files/?artifact\\_id=41579](http://portal.opengeospatial.org/files/?artifact_id=41579)

### 3 Definitions

#### 3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

**3.1.1 collection** [b-IETF RFC 5023]: Sets of resources, which can be retrieved in whole or in part.

**3.1.2 entity** [OASIS OData Part1]: Instances of entity types.

**3.1.3 entity sets** [OASIS OData Part1]: Named collections of entities (e.g., Sensors is an entity set containing Sensor entities). An entity's key uniquely identifies the entity within an entity set. Entity sets provide entry points into an OGC SensorThings API service.

**3.1.4 Internet of things (IoT)** [ITU-T Y.4000]: A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies.

NOTE 1 – Through the exploitation of identification, data capture, processing and communication capabilities, the IoT makes full use of things to offer services to all kinds of applications, whilst ensuring that security and privacy requirements are fulfilled.

NOTE 2 – From a broader perspective, the IoT can be perceived as a vision with technological and societal implications.

**3.1.5 measurement** [OGC/ISO 19156]: A set of operations having the object of determining the value of a quantity.

**3.1.6 observation** [OGC/ISO 19156]: Act of measuring or otherwise determining the value of a property.

**3.1.7 observation result** [OGC/ISO 19156]: Estimate of the value of a property determined through a known observation procedure.

**3.1.8 resource** [IETF RFC 2616]: A network-accessible data object or service identified by an URI.

**3.1.9 sensor** [b-OGC 12-000]: An entity capable of observing a phenomenon and returning an observed value. Type of observation procedure that provides the estimated value of an observed property at its output.

**3.1.10 thing** [ITU-T Y.4000]: With regard to the Internet of things, this is an object of the physical world (physical things) or the information world (virtual things), which is capable of being identified and integrated into communication networks.

#### 3.2 Terms defined in this Recommendation

This Recommendation defines the following term:

**3.2.1 representational state transfer (REST)**: An abstraction of the architectural elements within a distributed hypermedia system that focuses on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements.

NOTE 1 – REST encompasses the fundamental constraints upon components, connectors, and data that define the basis of the web architecture, and thus the essence of its behaviour as a network-based application.

NOTE 2 – An API that conforms to the REST architectural principles/constraints is called a RESTful API.

#### **4 Abbreviations and acronyms**

This Recommendation uses the following abbreviations and acronyms:

API	Application Programming Interface
CRUD	Create, Read, Update, and Delete
GML	Geography Markup Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
JSON	JavaScript Object Notation
MQTT	Message Queuing Telemetry Transport
OData	Open Data Protocol
O&M	Observations and Measurements
REST	Representational State Transfer
SensorML	Sensor Model Language
SOS	Sensor Observation Service
SWE	Sensor Web Enablement
UCUM	Unified Code for Units of Measure
UML	Unified Modelling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WiFi	Wireless-Fidelity
WKT	Well-Known Text
XML	extensible Markup Language

#### **5 Conventions**

In this Recommendation:

The word "SHALL" is used to express mandatory requirements for conformance to the Recommendation. The word "MAY" is used to refer to behaviour that is allowed, but not necessarily required. The word "SHOULD" is used to refer to behaviour that is encouraged to be followed, but is not a mandatory requirement for conformance to this Recommendation.

#### **6 The SensorThings API entities**

##### **6.1 Common control information**

In SensorThings control information is represented as annotations whose names start with `iot` followed by a dot ( `iot.` ). Annotations are name/value pairs that have a dot ( `.` ) as part of the name.

When annotating a name/value pair for which the value is represented as a JSON object, each annotation is placed within the object and represented as a single name/value pair. In SensorThings

the name always starts with the "at" sign (@), followed by the namespace iot, followed by a dot (.), followed by the name of the term (e.g., "@iot.id":1).

When annotating a name/value pair for which the value is represented as a JSON array or primitive value, each annotation that applies to this name/value pair is placed next to the annotated name/value pair and represented as a single name/value pair. The name is the same as the name of the name/value pair being annotated, followed by the "at" sign (@), followed by the namespace iot, followed by a dot (.), followed by the name of the term. (e.g., "Locations@iot.navigationLink":"http://example.org/v.1.0/Things(1)/Locations")

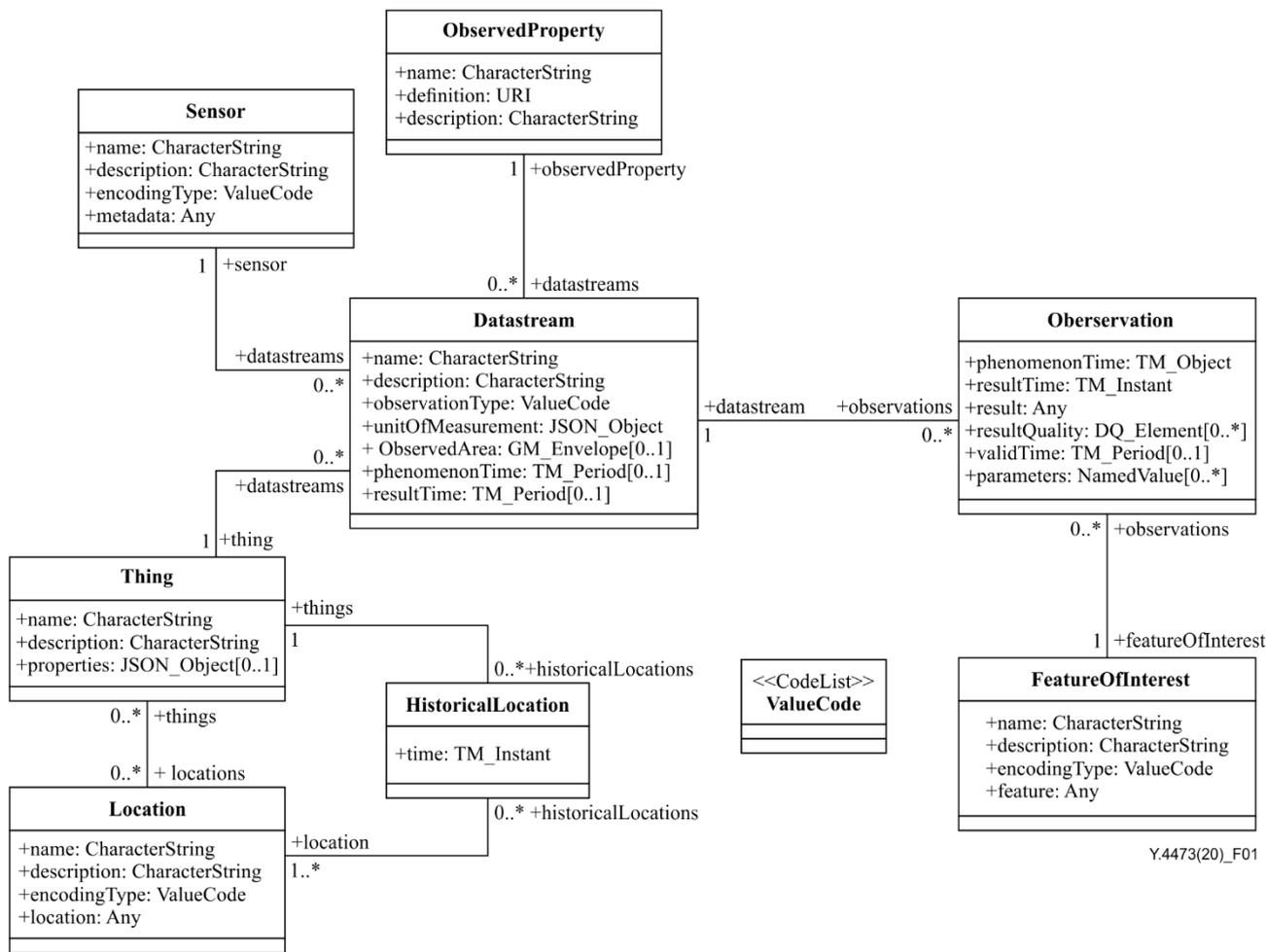
Table 1 presents common control information.

**Table 1 – Common control information**

<b>Annotation</b>	<b>Definition</b>	<b>Data type</b>	<b>Multiplicity and use</b>
<b>id</b>	id is the system-generated identifier of an entity. id is unique among the entities of the same entity type in a SensorThings service.	Any	One (mandatory)
<b>selfLink</b>	selfLink is the absolute URL of an entity that is unique among all other entities.	URL	One (mandatory)
<b>navigationLink</b>	navigationLink is the relative or absolute URL that retrieves content of related entities.	URL	One-to-many (mandatory)

## 6.2 The sensing entities

The SensorThings API Sensing part's entities are depicted in Figure 1. The sensing entities include Things, Location, HistoricalLocation, Datastream, Sensor, ObservedProperty, Observations, and FeatureOfInterest.



Y.4473(20)\_F01

**Figure 1 – Sensing entities**

In this clause, the properties in each entity type and the direct relationship to the other entity types are described. In addition, for each entity type, an example of the associated JSON encoding is presented.

### 6.2.1 Thing

The SensorThings API follows the ITU-T definition, i.e., with regard to the Internet of things, a thing is an object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication networks [ITU-T Y.4000].

Table 2 shows properties of a Thing entity and Table 3 shows the direct relation between a Thing entity and other entity types.

**Table 2 – Properties of a Thing entity**

Name	Definition	Data type	Multiplicity and use
<b>name</b>	A property provides a label for Thing entity, commonly a descriptive name.	CharacterString	One (mandatory)
<b>description</b>	This is a short description of the corresponding Thing entity.	CharacterString	One (mandatory)
<b>properties</b>	A JSON Object containing user-annotated properties as key-value pairs.	JSON Object	Zero-to-one

**Table 3 – Direct relation between a Thing entity and other entity types**

Entity type	Relation	Description
<b>Location</b>	Many optional to many optional	The Location entity locates the Thing. Multiple Things MAY be located at the same Location. A Thing MAY not have a Location. A Thing SHOULD have only one Location. However, in some complex use cases, a Thing MAY have more than one Location representations. In such case, the Thing MAY have more than one Locations. These Locations SHALL have different encodingTypes and the encodingTypes SHOULD be in different spaces (e.g., one encodingType in Geometrical space and one encodingType in Topological space).
<b>HistoricalLocation</b>	One mandatory to many optional	A Thing has zero-to-many HistoricalLocations. A HistoricalLocation has one-and-only-one Thing.
<b>Datastream</b>	One mandatory to many optional	A Thing MAY have zero-to-many Datastreams.

**Example 1 – An example of a Thing entity:**

```

{
  "@iot.id": 1,
  "@iot.selfLink": "http://example.org/v1.0/Things(1)",
  "Locations@iot.navigationLink": "Things(1)/Locations",
  "Datastreams@iot.navigationLink": "Things(1)/Datastreams",
  "HistoricalLocations@iot.navigationLink": "Things(1)/HistoricalLocations",
  "name": "Oven",
  "description": "This thing is an oven.",
  "properties": {
    "owner": "Noah Liang",
    "color": "Black"
  }
}

```

## 6.2.2 Location

The Location entity locates the Thing or the Things it is associated with. A Thing's Location entity is defined as the last known location of the Thing.

A Thing's Location may be identical to the Thing's Observations' FeatureOfInterest. In the context of the IoT, the principle location of interest is usually associated with the location of the Thing, especially for in-situ sensing applications. For example, the location of interest of a WiFi-connected thermostat should be the building or the room in which the smart thermostat is located. The FeatureOfInterest of the Observations made by the thermostat (e.g., room temperature readings) should also be the building or the room. In this case, the content of the smart thermostat's location should be the same as the content of the temperature readings' feature of interest.

However, the ultimate location of interest of a Thing is not always the location of the Thing (e.g., in the case of remote sensing). In those use cases, the content of a Thing's Location is different from the content of the FeatureOfInterest of the Thing's Observations. Section 7.1.4 of [OGC/ISO 19156] provides a detailed explanation of observation location.

Table 4 lists properties of a Location entity, Table 5 lists the direct relations between a Location entity and other entity types and Table 6 lists of some code values used for identifying types for the encodingType of the Location and FeatureOfInterest entity.

**Table 4 – Properties of a Location entity**

<b>Name</b>	<b>Definition</b>	<b>Data type</b>	<b>Multiplicity and use</b>
<b>name</b>	A property provides a label for Location entity, commonly a descriptive name.	CharacterString	One (mandatory)
<b>description</b>	The description about the Location.	CharacterString	One (mandatory)
<b>encodingType</b>	The encoding type of the Location property. Its value is one of the ValueCode enumeration.	ValueCode	One (mandatory)
<b>location</b>	The location type is defined by encodingType.	Any ( <i>i.e.</i> , the type is depending on the value of the encodingType)	One (mandatory)

**Table 5 – Direct relations between a Location entity and other entity types**

Entity type	Relation	Description
<b>Thing</b>	Many optional to many optional	Multiple Things MAY locate at the same Location. A Thing MAY not have a Location.
<b>HistoricalLocation</b>	Many mandatory to many optional	A Location can have zero-to-many HistoricalLocations. One HistoricalLocation SHALL have one or many Locations.

**Example 2** – An example of a Location entity:

```
{
  "@iot.id": 1,
  "@iot.selfLink": "http://example.org/v1.0/Locations(1)",
  "Things@iot.navigationLink": "Locations(1)/Things",
  "HistoricalLocations@iot.navigationLink": "Locations(1)/HistoricalLocations",
  "encodingType": "application/vnd.geo+json",
  "name": "CCIT",
  "description": "Calgary\n Center for Innovative Technologies",
  "location": {
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -114.06,
        51.05
      ]
    }
  }
}
```

**Table 6 – Code values used for identifying types for the encodingType of the Location and FeatureOfInterest entity**

Location encodingType	ValueCode Value
GeoJSON <sup>1</sup>	application/vnd.geo+json

A thing can be geo-referenced in different spaces. For example, for some applications it is more suitable to use a topological space model (e.g., IndoorGML) to describe an indoor things' location rather than using a geometric space model (e.g., GeoJSON). Currently GeoJSON is the only Location encodingType of the SensorThings API. In the future it is expected to extend SensorThings API's

---

<sup>1</sup> GeoJSON: Geospatial data interchange format based on JavaScript Object Notation, defined in [IETF RF 7946].

capabilities by adding additional encodingType to the code values listed in the above table. For example, one potential new Location encodingType can be a JSON encoding for IndoorGML.

### 6.2.3 HistoricalLocation

A Thing's HistoricalLocation entity set provides the times of the current (i.e., last known) and previous locations of the Thing.

The HistoricalLocation can also be created, updated and deleted. One use case is to migrate historical observation data from an existing observation data management system to a SensorThings API system.

Table 7 lists properties of a HistoricalLocation entity and Table 8 lists direct relations between an HistoricalLocation entity and other entity types.

**Table 7 – Properties of a HistoricalLocation entity**

Name	Definition	Data type	Multiplicity and use
<b>time</b>	The time when the Thing is known at the Location.	TM_Instant (ISO-8601 Time String)	One (mandatory)

**Table 8 – Direct relations between an HistoricalLocation entity and other entity types**

Entity type	Relation	Description
<b>Location</b>	Many optional to many mandatory	A Location can have zero-to-many HistoricalLocations. One HistoricalLocation SHALL have one or many Locations.
<b>Thing</b>	Many optional to one mandatory	A HistoricalLocation has one-and-only-one Thing. One Thing MAY have zero-to-many HistoricalLocations.

**Example 3** – An example of a HistoricalLocations entity set (e.g., Things(1)/HistoricalLocations):

```
{
  "value": [
    {
      "@iot.id": 1,
      "@iot.selfLink": "http://example.org/v1.0/HistoricalLocations(1)",
      "Locations@iot.navigationLink": "HistoricalLocations(1)/Locations",
      "Thing@iot.navigationLink": "HistoricalLocations(1)/Thing",
      "time": "2015-01-25T12:00:00-07:00"
    },
    {
      "@iot.id": 2,
      "@iot.selfLink": "http://example.org/v1.0/HistoricalLocations(2)",
      "Locations@iot.navigationLink": "HistoricalLocations(2)/Locations",
      "Thing@iot.navigationLink": "HistoricalLocations(2)/Thing",
      "time": "2015-01-25T13:00:00-07:00"
    }
  ],
  "@iot.nextLink": "http://example.org/v1.0/Things(1)/HistoricalLocations?$skip=2&$top=2"
}
```

## 6.2.4 Datastream

A Datastream groups a collection of Observations measuring the same ObservedProperty and produced by the same Sensor.

Table 9 lists properties of a Datastream entity and Table 10 lists direct relations between a Datastream entity and other entity types.

**Table 9 – Properties of a Datastream entity**

<b>Name</b>	<b>Definition</b>	<b>Data type</b>	<b>Multiplicity and use</b>
<b>name</b>	A property provides a label for Datastream entity, commonly a descriptive name.	CharacterString	One (mandatory)
<b>description</b>	The description of the Datastream entity.	CharacterString	One (mandatory)
<b>unitOfMeasurement</b>	A JSON Object containing three key-value pairs. The name property presents the full name of the unitOfMeasurement; the symbol property shows the textual form of the unit symbol; and the definition contains the URI defining the unitOfMeasurement. The values of these properties SHOULD follow the Unified Code for Unit of Measure (UCUM).	JSON Object	One (mandatory) NOTE – When a Datastream does not have a unit of measurement (e.g., an OM_TruthObservation type), the corresponding unitOfMeasurement properties SHALL have null values.
<b>observationType</b>	The type of Observation (with unique result type), which is used by the service to encode observations.	ValueCode see Table 14.	One (mandatory)
<b>observedArea</b>	The spatial bounding box of the spatial extent of all FeaturesOfInterest that belong to the Observations associated with this Datastream.	GM_Envelope (GeoJSON Polygon)	Zero-to-one (optional)
<b>phenomenonTime</b>	The temporal interval of the phenomenon times of all observations belonging to this Datastream.	TM_Period (ISO 8601 Time Interval)	Zero-to-one (optional)

**Table 9 – Properties of a Datastream entity**

Name	Definition	Data type	Multiplicity and use
<b>resultTime</b>	The temporal interval of the result times of all observations belonging to this Datastream.	TM_Period (ISO 8601 Time Interval)	Zero-to-one (optional)

**Table 10 – Direct relation between a Datastream entity and other entity types**

Entity type	Relation	Description
<b>Thing</b>	Many optional to one mandatory	A Thing has zero-to-many Datastreams. A Datastream entity SHALL only link to a Thing as a collection of Observations.
<b>Sensor</b>	Many optional to one mandatory	The Observations in a Datastream are performed by one-and-only-one Sensor. One Sensor MAY produce zero-to-many Observations in different Datastreams.
<b>ObservedProperty</b>	Many optional to one mandatory	The Observations of a Datastream SHALL observe the same ObservedProperty. The Observations of different Datastreams MAY observe the same ObservedProperty.
<b>Observation</b>	One mandatory to many optional	A Datastream has zero-to-many Observations. One Observation SHALL occur in one-and-only-one Datastream.

**Example 4 – A Datastream entity example:**

```

{
  "@iot.id": 1,
  "@iot.selfLink": "http://example.org/v1.0/Datastreams(1)",
  "Thing@iot.navigationLink": "HistoricalLocations(1)/Thing",
  "Sensor@iot.navigationLink": "Datastreams(1)/Sensor",
  "ObservedProperty@iot.navigationLink": "Datastreams(1)/ObservedProperty",
  "Observations@iot.navigationLink": "Datastreams(1)/Observations",
  "name": "oven temperature",
  "description": "This is a datastream measuring the air temperature in an oven.",
  "unitOfMeasurement": {
    "name": "degree Celsius",
    "symbol": "°C",
    "definition": "http://unitsofmeasure.org/ucum.html#para-30"
  },
  "observationType": "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
  "observedArea": {
    "type": "Polygon",
    "coordinates":
      [[[100,0],[101,0],[101,1],[100,1],[100,0]]]
  },
  "phenomenonTime": "2014-03-01T13:00:00Z/2015-05-11T15:30:00Z",
  "resultTime": "2014-03-01T13:00:00Z/2015-05-11T15:30:00Z"
}

```

The observationType defines the result types for specialized observations [OGC/ISO 19156]. Table 11 shows some of the valueCodes that maps the Unified modelling language (UML) classes in O&M v2.0 [OGC/ISO 19156] to observationType names and observation result types.

**Table 11 – List of some code values used for identifying types defined in the O&M conceptual model [OGC/ISO 19156]**

O&M 2.0	Code value (observationType names)	Content of result
OM_CategoryObservation	<a href="http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CategoryObservation">http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CategoryObservation</a>	URI
OM_CountObservation	<a href="http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CountObservation">http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CountObservation</a>	integer
OM_Measurement	<a href="http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement">http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement</a>	double
OM_Observation	<a href="http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Observation">http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Observation</a>	Any
OM_TruthObservation	<a href="http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_TruthObservation">http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_TruthObservation</a>	boolean

### 6.2.5 Sensor

A Sensor is an instrument that observes a property or phenomenon with the goal of producing an estimate of the value of the property<sup>2</sup>.

Table 12 lists properties of a Sensor entity, Table 13 lists direct relations between a Sensor entity and other entity types and Table 14 lists some code values used for identifying types for the encodingType of the Sensor entity.

**Table 12 – Properties of a Sensor entity**

Name	Definition	Data type	Multiplicity and use
<b>name</b>	A property provides a label for Sensor entity, commonly a descriptive name.	CharacterString	One (mandatory)
<b>description</b>	The description of the Sensor entity.	CharacterString	One (mandatory)

<sup>2</sup> In some cases, the Sensor in this data model can also be seen as the Procedure (method, algorithm, or instrument) defined in [OGC/ISO 19156].

**Table 12 – Properties of a Sensor entity**

Name	Definition	Data type	Multiplicity and use
<b>encodingType</b>	The encoding type of the metadata property. Its value is one of the ValueCode enumeration (Table 14. for the available ValueCode).	ValueCode	One (mandatory)
<b>metadata</b>	The detailed description of the Sensor or system. The metadata type is defined by encodingType.	Any (depending on the value of the encodingType)	One (mandatory)

**Table 13 – Direct relations between a Sensor entity and other entity types**

Entity type	Relation	Description
<b>Datastream</b>	One mandatory to many optional	The Observations of a Datastream are measured with the same Sensor. One Sensor MAY produce zero-to-many Observations in different Datastreams.

**Table 14 – List of some code values used for identifying types for the encodingType of the Sensor entity**

Sensor encodingType	ValueCode Value
PDF	application/pdf
SensorML	<a href="http://www.opengis.net/doc/IS/SensorML/2.0">http://www.opengis.net/doc/IS/SensorML/2.0</a>

The Sensor encodingType allows clients to know how to interpret metadata's value. Currently SensorThings API defines two common Sensor metadata encodingTypes. Most sensor manufacturers provide their sensor datasheets in a PDF format. As a result, PDF is a Sensor encodingType supported by SensorThings API. The second Sensor encodingType is SensorML.

**Example 5** – An example of a Sensor entity:

```
{
  "@iot.id": 1,
  "@iot.selfLink": "http://example.org/v1.0/Sensors(1)",
  "Datastreams@iot.navigationLink": "Sensors(1)/Datastreams",
  "name": "TMP36",
  "description": "TMP36\n – Analog Temperature sensor",
  "encodingType": "application/pdf",
  "metadata": "http://example.org/TMP35_36_37.pdf"
}
```

### 6.2.6 ObservedProperty

An ObservedProperty specifies the phenomenon of an Observation. Table 15 lists properties of an ObservedProperty entity and Table 16 lists direct relations between an ObservedProperty entity and other entity types.

**Table 15 – Properties of an ObservedProperty entity**

Name	Definition	Data type	Multiplicity and use
<b>name</b>	A property provides a label for ObservedProperty entity, commonly a descriptive name.	CharacterString	One (mandatory)
<b>definition</b>	The URI of the ObservedProperty. Dereferencing this URI SHOULD result in a representation of the definition of the ObservedProperty.	URI	One (mandatory)
<b>description</b>	A description about the ObservedProperty.	CharacterString	One (mandatory)

**Table 16 – Direct relations between an ObservedProperty entity and other entity types**

Entity type	Relation	Description
<b>Datastream</b>	One mandatory to many optional	The Observations of a Datastream observe the same ObservedProperty. The Observations of different Datastreams MAY observe the same ObservedProperty.

**Example 6** – An example ObservedProperty entity:

```
{
  "@iot.id": 1,
  "@iot.selfLink": "http://example.org/v1.0/ObservedProperties(1)",
  "Datastreams@iot.navigationLink": "ObservedProperties(1)/Datastreams",
  "description": "The dewpoint temperature is the temperature to which the air must be\n cooled, at constant pressure, for dew to form. As the grass and other objects\n near the ground cool to the dewpoint, some of the water vapor in the\n atmosphere condenses into liquid water on the objects.",
  "name": "DewPoint Temperature",
  "definition": "http://dbpedia.org/page/Dew_point"
}
```

### 6.2.7 Observations

An Observation is the act of measuring or otherwise determining the value of a property [OGC/ISO 19156]. Table 17 lists properties of an Observation entity and Table 18 lists direct relations between an Observation entity and other entity types.

**Table 17 – Properties of an Observation entity**

Name	Definition	Data type	Multiplicity and use
<b>phenomenonTime</b>	<p>The time instant or period of when the Observation happens.</p> <p>NOTE – Many resource-constrained sensing devices do not have a clock. As a result, a client may omit phenomenonTime when POST new Observations, even though phenomenonTime is a mandatory property. When a SensorThings service receives a POST Observations without phenomenonTime, the service SHALL assign the current server time to the value of the phenomenonTime.</p>	<p>TM_Object (ISO 8601 Time string or Time Interval string (e.g., 2010-12-23T10:20:00.00-07:00 or 2010-12-23T10:20:00.00-07:00/2010-12-23T12:20:00.00-07:00))</p>	<p>One (mandatory)</p>
<b>result</b>	<p>The estimated value of an ObservedProperty from the Observation.</p>	<p>Any (depends on the observationType defined in the associated Datastream)</p>	<p>One (mandatory)</p>
<b>resultTime</b>	<p>The time of the Observation's result was generated.</p> <p>NOTE – Many resource-constrained sensing devices do not have a clock. As a result, a client may omit resultTime when POST new Observations, even though resultTime is a mandatory property. When a SensorThings service receives a POST Observations without resultTime, the service SHALL assign a null value to the resultTime.</p>	<p>TM_Instant (ISO 8601 Time string)</p>	<p>One (mandatory)</p>
<b>resultQuality</b>	<p>Describes the quality of the result.</p>	<p>DQ_Element</p>	<p>Zero-to-many</p>
<b>validTime</b>	<p>The time period during which the result may be used.</p>	<p>TM_Period (ISO 8601 Time Interval string)</p>	<p>Zero-to-one</p>
<b>parameters</b>	<p>Key-value pairs showing the environmental conditions during measurement.</p>	<p>NamedValues in a JSON Array</p>	<p>Zero-to-one</p>

**Table 18 – Direct relations between an Observation entity and other entity types**

Entity type	Relation	Description
<b>Datastream</b>	Many optional to one mandatory	A Datastream can have zero-to-many Observations. One Observation SHALL occur in one-and-only-one Datastream.
<b>FeatureOfInterest</b>	Many optional to one mandatory	An Observation observes on one-and-only-one FeatureOfInterest. One FeatureOfInterest could be observed by zero-to-many Observations.

**Example 7** – An Observation entity example. The following example shows an Observation whose Datastream has an ObservationType of OM\_Measurement. A result's data type is defined by the observationType.

```
{
  "@iot.id": 1,
  "@iot.selfLink": "http://example.org/v1.0/Observations(1)",
  "FeatureOfInterest@iot.navigationLink": "Observations(1)/FeatureOfInterest",
  "Datastream@iot.navigationLink": "Observations(1)/Datastream",
  "phenomenonTime": "2014-12-31T11:59:59.00+08:00",
  "resultTime": "2014-12-31T11:59:59.00+08:00",
  "result": 70.4
}
```

### 6.2.8 FeatureOfInterest

An Observation results in a value being assigned to a phenomenon. The phenomenon is a property of a feature, the latter being the FeatureOfInterest of the Observation [OGC/ISO 19156]. In the context of the Internet of things, many Observations' FeatureOfInterest can be the Location of the Thing. For example, the FeatureOfInterest of a WiFi-connected thermostat can be the Location of the thermostat (i.e., the living room where the thermostat is located). In the case of remote sensing, the FeatureOfInterest can be the geographical area or volume that is being sensed. Table 19 lists properties of a FeatureOfInterest entity and Table 20 lists direct relations between a FeatureOfInterest entity and other entity types.

**Table 19 – Properties of a FeatureOfInterest entity**

Name	Definition	Data type	Multiplicity and use
<b>name</b>	A property provides a label for FeatureOfInterest entity, commonly a descriptive name.	CharacterString	One (mandatory)
<b>description</b>	The description about the FeatureOfInterest.	CharacterString	One (mandatory)

Name	Definition	Data type	Multiplicity and use
<b>encodingType</b>	The encoding type of the feature property. Its value is one of the ValueCode enumeration (for the available ValueCode).	ValueCode	One (mandatory)
<b>feature</b>	The detailed description of the feature. The data type is defined by encodingType.	Any	One (mandatory)

**Table 20 – Direct relation between a FeatureOfInterest entity and other entity types**

Entity type	Relation	Description
<b>Observation</b>	One mandatory to many optional	An Observation observes on one-and-only-one FeatureOfInterest. One FeatureOfInterest could be observed by zero-to-many Observations.

**Example 8 – An example of a FeatureOfInterest entity**

```
{
  "@iot.id": 1,
  "@iot.selfLink": "http://example.org/v1.0/FeaturesOfInterest(1)",
  "Observations@iot.navigationLink": "FeaturesOfInterest(1)/Observations",
  "name": "Weather Station YYC.",
  "description": "This is a weather station located at the Calgary Airport.",
  "encodingType": "application/vnd.geo+json",
  "feature": {
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -114.06,
        51.05
      ]
    }
  }
}
```

**7 SensorThings service interface**

A SensorThings API service exposes service document resources that describe its data model. The service document lists the entity sets that can be create, read, update and delete (CRUD). SensorThings API clients can use the service document to navigate the available entities in a hypermedia-driven fashion.

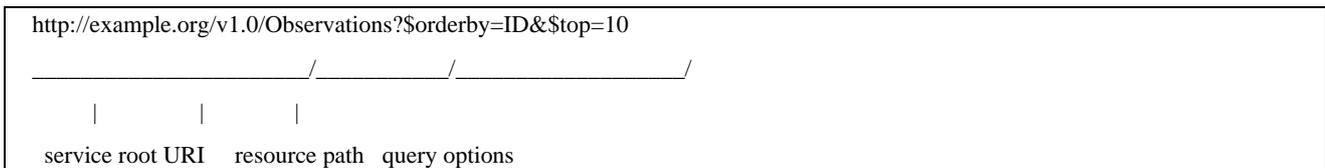
## 7.1 Common control information

The SensorThings API service groups the same types of entities into entity sets. Each entity has a unique identifier and one-to-many properties. Also, in the case of an entity holding a relationship with entities in other entity sets, this type of relationship is expressed with navigation properties (i.e., navigationLink and associationLink).

Therefore, in order to perform CRUD actions on the resources, the first step is to address to the target resource(s) through URI. There are three major URI components used here, namely (1) the service root URI, (2) the resource path, and (3) the query options. In addition, the service root URI consists of two parts: (1) the location of the SensorThings service and (2) the version number. The version number follows the format indicated below:

"v"majorversionnumber + "." + minorversionnumber

### Example 9 – Complete URI example



By attaching the resource path after the service root URI, clients can address to different types of resources such as an entity set, an entity, a property, or a navigation property. Finally, clients can apply query options after the resource path to further process the addressed resources, such as sorting by properties or filtering with criteria.

## 7.2 Resource path

The resource path comes right after the service root URI and can be used to address to different resources. Clauses 7.2.1 to 7.2.8 list the usages of the resource path.

### 7.2.1 Usage 1: No resource path

#### URI Pattern: SERVICE\_ROOT\_URI

Response: A JSON object with a property named value. The value of the property SHALL be a JSON array containing one element for each entity set of the SensorThings Service.

Each element SHALL be a JSON object with at least two name/value pairs, one with name name containing the name of the entity set (e.g., Things, Locations, Datastreams, Observations, ObservedProperties and Sensors) and one with name url containing the URL of the entity set, which may be an absolute or a relative URL [OASIS OData JSON].

#### Example 10 – A SensorThings request with no resource path

Example request:

http://example.org/v1.0/

Example response:

```
{
  "value": [
    {
      "name": "Things",
      "url": "http://example.org/v1.0/Things"
    },
    {
```

```

"name": "Locations",
"url": " http://example.org/v1.0/Locations"
},
{
"name": "Datastreams",
"url": " http://example.org/v1.0/Datastreams"
},
{
"name": "Sensors",
"url": " http://example.org/v1.0/Sensors"
},
{
"name": "Observations",
"url": " http://example.org/v1.0/Observations"
},
{
"name": "ObservedProperties",
"url": "\n http://example.org/v1.0/ObservedProperties"
},
{
"name": "FeaturesOfInterest",
"url": "\n http://example.org/v1.0/FeaturesOfInterest"
}
]
}

```

### 7.2.2 Usage 2: Address to a collection of entities

To address to an entity set, users can simply put the entity set name after the service root URI. The service returns a JSON object with a property of value. The value of the property SHALL be a list of the entities in the specified entity set.

**URI Pattern:** SERVICE\_ROOT\_URI/ENTITY\_SET\_NAME

**Response:** A list of all entities (with all the properties) in the specified entity set when there is no service-driven pagination imposed. The response is represented as a JSON object containing a name/value pair named value. The value of the value name/value pair is a JSON array where each element is representation of an entity or a representation of an entity reference. An empty collection is represented as an empty JSON array.

The count annotation represents the number of entities in the collection. If present, it comes before the value name/value pair.

When there is service-driven pagination imposed, the nextLink annotation is included in a response that represents a partial result [OASIS OData JSON].

**Example 11** – An example to address an entity set

Example request:

http://example.org/v1.0/ObservedProperties

Example response:

```
{
  "@iot.count": 84,
  "value": [
    {
      "@iot.id": 1,
      "@iot.selfLink": "http://example.org/v1.0/ObservedProperties(1)",
      "Datastreams@iot.navigationLink": "ObservedProperties(1)/Datastreams",
      "description": "The dew point is the temperature at which the water vapor in air at constant barometric pressure condenses into liquid water at the same rate at which it evaporates.",
      "name": "DewPoint Temperature",
      "definition": "http://dbpedia.org/page/Dew_point"
    },
    {
      "@iot.id": 2,
      "@iot.selfLink": "http://example.org/v1.0/ObservedProperties(2)",
      "Datastreams@iot.navigationLink": "ObservedProperties(2)/Datastreams",
      "description": "Relative humidity is the ratio of the partial pressure of water vapor in an air-water mixture to the saturated vapor pressure of water at a prescribed temperature.",
      "name": "Relative Humidity",
      "definition": "http://dbpedia.org/page/Relative_humidity"
    }
  ],
  "@iot.nextLink": "http://example.org/v1.0/ObservedProperties?$top=5&$skip=5"
}
```

### 7.2.3 Usage 3: Address to an entity in a collection

Users can address a specific entity in an entity set by place the unique identifier of the entity between brace symbol "(" and put after the entity set name. The service then returns the entity with all its properties.

**URI Pattern:** SERVICE\_ROOT\_URI/ENTITY\_SET\_NAME(ID\_OF\_THE\_ENTITY)

**Response:** A JSON object of the entity (with all its properties) that holds the specified id in the entity set.

**Example 12** – An example request that addresses to an entity in a collection

Example request:

http://example.org/v1.0/Things(1)

### 7.2.4 Usage 4: Address to a property of an entity

Users can address to a property of an entity by specifying the property name after the URI addressing to the entity. The service then returns the value of the specified property. If the property has a complex type value, properties of that value can be addressed by further property name composition.

If the property is single-valued and has the null value, the service SHALL respond with 204 No Content. If the property is not available, for example due to permissions, the service SHALL respond with 404 Not Found [OASIS OData Part1].

**URI Pattern:**

SERVICE\_ROOT\_URI/RESOURCE\_PATH\_TO\_AN\_ENTITY/PROPERTY\_NAME

**Response:** The specified property of an entity that holds the id in the entity set.

**Example 13** – An example to address to a property of an entity

Example request:

```
http://example.org/v1.0/Observations(1)/resultTime
```

Example response:

```
{
  "resultTime":
  "2010-12-23T10:20:00-07:00"
}
```

### 7.2.5 Usage 5: Address to the value of an entity's property

To address the raw value of a primitive property, clients append a path segment containing the string \$value to the property URL.

The default format for TM\_Object types is text/plain using the ISO 8601 format, such as 2014-03-01T13:00:00Z/2015-05-11T15:30:00Z for TM\_Period and 2014-03-01T13:00:00Z for TM\_Instant.

#### URI Pattern:

SERVICE\_ROOT\_URI/ENTITY\_SET\_NAME(ID\_OF\_THE\_ENTITY)/PROPERTY\_NAME/\$value

**Response:** The raw value of the specified property of an entity that holds the id in the entity set.

**Example 14** – An example of addressing to the value of an entity's property

Example:

```
http://example.org/v1.0/Observations(1)/resultTime/$value
```

Example response:

```
2015-01-12T23:00:13-07:00
```

### 7.2.6 Usage 6: Address to a navigation property (navigationLink)

As the entities in different entity sets may hold some relationships, users can request the linked entities by addressing to a navigation property of an entity. The service then returns one or many entities that hold a certain relationship with the specified entity.

#### URI Pattern:

SERVICE\_ROOT\_URI/ENTITY\_SET\_NAME(ID\_OF\_THE\_ENTITY)/LINK\_NAME

**Response:** A JSON object of one entity or a JSON array of many entities that holds a certain relationship with the specified entity.

**Example 15** – An example request addressing to a navigational property

Example:

```
http://example.org/v1.0/Datastreams(1)/Observations
```

Returns all the Observations in the Datastream that holds the id 1.

### 7.2.7 Usage 7: Address to an associationLink

As the entities in different entity sets may hold some relationships, users can request the linked entities' selfLinks by addressing to an association link of an entity. An associationLink can be used to retrieve a reference to an entity or an entity set related to the current entity. Only the selfLinks of related entities are returned when resolving associationLinks.

#### URI Pattern:

SERVICE\_ROOT\_URI/ENTITY\_SET\_NAME(KEY\_OF\_THE\_ENTITY)/LINK\_NAME/\$ref

**Response:** A JSON object with a value property. The value of the value property is a JSON array containing one element for each associationLink. Each element is a JSON object with a name/value pair. The name is url and the value is the selfLinks of the related entities.

**Example 16** – An example of addressing to an association link

Example request:

```
http://example.org/v1.0/Datastreams(1)/Observations/$ref
```

Returns all the selfLinks of the Observations of Datastream(1).

Example response:

```
{
  "value": [
    {
      "@iot.selfLinks":
      "http://example.org/v1.0/Observations(1)"
    },
    {
      "@iot.selfLinks":
      "http://example.org/v1.0/Observations(2)"
    }
  ]
}
```

### 7.2.8 Usage 8: Nested resource path

As users can use navigation properties to link from one entity set to another, users can further extend the resource path with unique identifiers, properties, or links (i.e., Usage 3, 4 and 6).

**Example 17** – Examples of nested resource path

Example request 1:

```
http://example.org/v1.0/Datastreams(1)/Observations(1)
```

Returns a specific Observation entity in the Datastream.

Example request 2:

```
http://example.org/v1.0/Datastreams(1)/Observations(1)/resultTime
```

Returns the resultTime property of the specified Observation in the Datastream.

Example Request 3:

```
http://example.org/v1.0/Datastreams(1)/Observations(1)/FeatureOfInterest
```

Returns the FeatureOfInterest entity of the specified Observation in the Datastream.

## 7.3 Requesting data

Clients issue HTTP GET requests to SensorThings API services for data. The resource path of the URL specifies the target of the request. Additional query operators can be specified through query options that are presented in the following way. The query operators are prefixed with a dollar (\$) character and specified as key-value pairs after the question symbol (?) in the request URI. Many of the SensorThings API's query options are adapted from OData's query options. OData developers should be able to pick up SensorThings API query options very quickly.

### 7.3.1 Evaluating system query options

The SensorThings API adapts many of OData's system query options and their usage. These query options allow refining the request.

The result of the service request is as if the system query options were evaluated in the following order.

Prior to applying any server-driven pagination:

- \$filter
- \$count
- \$orderby
- \$skip
- \$top

After applying any server-driven pagination:

- \$expand
- \$select

### 7.3.2 Specifying properties to return

The \$select and \$expand system query options enable the client to specify the set of properties to be included in a response.

#### 7.3.2.1 \$expand

**Example 18** – Examples of \$expand query option

Example request 1:

`http://example.org/v1.0/Things?$expand=Datastreams`

Returns the entity set of Things as well as each of the Datastreams associated with each Thing entity.

Example request 1 response:

```
{
  "values": [
    {
      "@iot.id": 1,
      "@iot.selfLink": "http://example.org/v1.0/Things(1)",
      "Locations@iot.navigationLink": "Things(1)/Locations",
      "Datastreams@iot.count": 1,
      "Datastreams": [
        {
          "@iot.id": 1,
          "@iot.selfLink": "http://example.org/v1.0/Datastreams(1)",
          "name": "oven temperature",
          "description": "This is a datastream measuring the air temperature in an oven.",
          "unitOfMeasurement": {
            "name": "degree Celsius",
            "symbol": "°C",
            "definition": "http://unitsofmeasure.org/ucum.html#para-30"
          },
          "observationType": "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
          "observedArea": {
            "type": "Polygon",
            "coordinates": [[[100,0],[101,0],[101,1],[100,1],[100,0]]]
          },
          "phenomenonTime": "2014-03-01T13:00:00Z/2015-05-11T15:30:00Z",
          "resultTime": "2014-03-01T13:00:00Z/2015-05-11T15:30:00Z"
        }
      ],
      "HistoricalLocations@iot.navigationLink": "Things(1)/HistoricalLocations",
      "description": "This thing is a convection oven.",
      "name": "Oven",
      "properties": {
        "owner": "John Doe",
        "color": "Silver"
      }
    }
  ]
}
```

Example request 2:

`http://example.org/v1.0/Things?$expand=Datastreams/ObservedProperty`

Returns the collection of Things, the Datastreams associated with each Thing, and the ObservedProperty associated with each Datastream.

Example request 3:

`http://example.org/v1.0/Datastreams(1)?$expand=Observations,ObservedProperty`

Returns the Datastream whose id is 1 as well as the Observations and ObservedProperty associated with this Datastream.

Query options can be applied to the expanded navigation property by appending a semicolon-separated list of query options, enclosed in parentheses, to the navigation property name. Allowed system query options are \$filter, \$select, \$orderby, \$skip, \$top, \$count, and \$expand [OASIS OData Part2].

Example request 4:

`http://example.org/v1.0/Datastreams(1)?$expand=Observations($filter=result eq 1)`

Returns the Datastream whose id is 1 as well as its Observations with a result equal to 1.

### 7.3.2.2 \$select

**Example 19** – Examples of \$select query option

Example request 1:

`http://example.org/v1.0/Observations?$select=result,resultTime`

Returns only the result and resultTime properties for each Observation entity.

Example request 2:

`http://example.org/v1.0/Datastreams(1)?$select=id,Observations&$expand=Observations/FeatureOfInterest`

Returns the id property of the Datastream entity, and all the properties of the entity identified by the Observations and FeatureOfInterest navigation properties.

Example request 3:

`http://example.org/v1.0/Datastreams(1)?$expand=Observations($select=result)`

Returns the Datastream whose id is 1 as well as the result property of the entity identified by the Observations navigation property.

### 7.3.3 Query entity sets

#### 7.3.3.1 \$orderby

**Example 20** – Examples of \$orderby query option

Example request 1:

`http://example.org/v1.0/Observations?$orderby=result`

Returns all Observations ordered by the result property in ascending order.

Example request 2:

`http://example.org/v1.0/Observations?$expand=Datastream&$orderby=Datastreams/id desc, phenomenonTime`

Returns all Observations ordered by the id property of the linked Datastream entry in descending order, then by the phenomenonTime property of Observations in ascending order.

#### 7.3.3.2 \$top

**Example 21** – Examples of \$top query option

Example request 1:

`http://example.org/v1.0/Things?$top=5`

Returns only the first five entities in the Things collection.

Example request 2:

```
http://example.org/v1.0/Observations?$top=5&$orderby=phenomenonTime%20desc
```

Returns the first five Observation entries after sorted by the phenomenonTime property in descending order.

### 7.3.3.3 \$skip

**Example 22** – Examples of \$skip query option

Example request 1:

```
http://example.org/v1.0/Things?$skip=5
```

Returns Thing entities starting with the sixth Thing entity in the Things collection.

Example request 2:

```
http://example.org/v1.0/Observations?$skip=2&$top=2&$orderby=resultTime
```

Returns the third and fourth Observation entities from the collection of all Observation entities when the collection is sorted by the resultTime property in ascending order.

### 7.3.3.4 \$count

**Example 23** – Examples of \$count query option

Example request 1:

```
http://example.org/v1.0/Things?$count=true
```

Return, along with the results, the total number of Things in the collection.

Example response:

```
{
  "@iot.count":2,
  "value": [
    {...},
    {...}
  ]
}
```

### 7.3.3.5 \$filter

**Example 24** – Examples of \$filter query option

Example request 1:

```
http://example.org/v1.0/Observations?$filter=result lt 10.00
```

Returns all Observations whose result is less than 10.00.

In addition, clients can choose to use the properties of linked entities in the \$filter predicate. The following are examples of the possible uses of the \$filter in the data model of the SensorThings service.

Example request 2:

```
http://example.org/v1.0/Observations?$filter=Datastream/id eq '1'
```

Returns all Observations whose Datastream's id is 1.

Example request 3:

```
http://example.org/v1.0/Things?$filter=geo.distance(Locations/location, geography'POINT(-122, 43)') gt 1
```

Returns Things that the distance between their last known locations and POINT(-122, 43) is greater than 1.

Example request 4:

```
http://example.org/v1.0/Things?$expand=
Datastreams/Observations/FeatureOfInterest&$filter=Datastreams/Observations/FeatureOfInterest/id eq 'FOI_1' and
Datastreams/Observations/resultTime ge 2010-06-01T00:00:00Z and Datastreams/Observations/resultTime le 2010-07-01T00:00:00Z
```

Returns Things that have any observations of a feature of interest with a unique identifier equals to 'FOI\_1' in June 2010.

### 7.3.3.5.1 Built-in filter operators

The SensorThings API supports a set of built-in filter operators, as described in Table 21. These built-in filter operator usages and definitions follow the [OASIS OData Part1] and [OASIS OData ABNF].

**Table 21 – Built-in filter operators**

Operator	Description	Example
<b>Comparison operators</b>		
eq	Equal	/ObservedProperties?\$filter=unitOfMeasurement/name eq 'degree Celsius'
ne	Not equal	/ObservedProperties?\$filter=unitOfMeasurement/name ne 'degree Celsius'
gt	Greater than	/Observations?\$filter=result gt 20.0
ge	Greater than or equal	/Observations?\$filter=result ge 20.0
lt	Less than	/Observations?\$filter=result lt 100
le	Less than or equal	/Observations?\$filter=result le 100
<b>Logical operators</b>		
and	Logical and	/Observations?\$filter=result le 3.5 and FeatureOfInterest/id eq '1'
or	Logical or	/Observations?\$filter=result gt 20 or result le 3.5
not	Logical negation	/Things?\$filter=not startswith(description,'test')
<b>Arithmetic operators</b>		
add	Addition	/Observations?\$filter=result add 5 gt 10
sub	Subtraction	/Observations?\$filter=result sub 5 gt 10
mul	Multiplication	/Observations?\$filter=result mul 2 gt 2000
div	Division	/Observations?\$filter=result div 2 gt 4

**Table 21 – Built-in filter operators**

Operator	Description	Example
mod	Modulo	/Observations?\$filter=result mod 2 eq 0
<b>Grouping operators</b>		
()	Precedence grouping	/Observations?\$filter=(result sub 5) gt 10

**7.3.3.5.2 Built-in query functions**

The SensorThings API supports spatial relationship functions, the SensorThings API defines nine additional geospatial functions based on the spatial relationship between two geometry objects. The spatial relationship functions are defined in the OGC simple feature access specification [OASIS oData Part1]. The names of these nine functions start with a prefix "st" following the OGC simple feature access specification [OASIS oData Part1]. In addition, the well-known text (WKT) format is the default input geometry for these nine functions. Table 22 lists built-in query functions which include spatial relationship functions and other related functions.

**Table 22 – Built-in query functions**

Function	Example
<b>String functions</b>	
bool substringof(string p0, string p1)	substringof('Sensor Things',description)
bool endswith(string p0, string p1)	endswith(description,'Things')
bool startswith(string p0, string p1)	startswith(description,'Sensor')
int length(string p0)	length(description) eq 13
int indexof(string p0, string p1)	indexof(description,'Sensor') eq 1
string substring(string p0, int p1)	substring(description,1) eq 'sensor Things'
string tolower(string p0)	tolower(description) eq 'sensor things'
string toupper(string p0)	toupper(description) eq 'SENSOR THINGS'

**Table 22 – Built-in query functions**

<b>Function</b>	<b>Example</b>
string trim(string p0)	trim(description) eq 'Sensor Things'
string concat(string p0, string p1)	concat(concat(unitOfMeasurement/symbol,', '), unitOfMeasurement/name) eq 'degree, Celsius'
<b>Date functions</b>	
int year	year(resultTime) eq 2015
int month	month(resultTime) eq 12
int day	day(resultTime) eq 8
int hour	hour(resultTime) eq 1
int minute	minute(resultTime) eq 0
int second	second(resultTime) eq 0
int fractionalseconds	second(resultTime) eq 0
int date	date(resultTime) ne date(validTime)
time	time(resultTime) le validTime
int totaloffsetminutes	totaloffsetminutes(resultTime) eq 60
now	resultTime ge now()
mindatetime	resultTime eq mindatetime()
maxdatetime	resultTime eq maxdatetime()
<b>Math functions</b>	
round	round(result) eq 32
floor	floor(result) eq 32

**Table 22 – Built-in query functions**

Function	Example
ceiling	ceiling(result) eq 33
Geospatial functions	
double geo.distance(Point p0, Point p1)	geo.distance(location, geography'POINT (30 10) ')
double geo.length(LineString p0)	geo.length(geography'LINESTRING (30 10, 10 30, 40 40) ')
bool geo.intersects(Point p0, Polygon p1)	geo.intersects(location, geography'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))')
Spatial relationship functions	
bool st_equals	st_equals(location, geography'POINT (30 10)')
bool st_disjoint	st_disjoint(location, geography'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))')
bool st_touches	st_touches(location, geography'LINESTRING (30 10, 10 30, 40 40)')
bool st_within	st_within(location, geography'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))')
bool st_overlaps	st_overlaps(location, geography'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))')
bool st_crosses	st_crosses(location, geography'LINESTRING (30 10, 10 30, 40 40)')
bool st_intersects	st_intersects(location, geography'LINESTRING (30 10, 10 30, 40 40)')
bool st_contains	st_contains(location, geography'POINT (30 10)')
bool st_relate	st_relate(location, geography'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))', 'T*****')

### 7.3.3.6 Server-driven paging

#### Example 25:

http://example.org/v1.0/Things

Returns a subset of the Thing entities of requested collection of Things. The nextLink contains a link allowing retrieving the next partial set of items.

Example response:

```
{
  "value": [
    {...},
    {...}
  ],
  "@iot.nextLink":
  "http://examples.org/v1.0/Things?$top=100&$skip=100"
}
```

## 8 SensorThings sensing create-update-delete

### 8.1 Overview

As many IoT devices are resource-constrained, the SensorThings API adopts the efficient REST web service style. That means the create, update and delete actions can be performed on the SensorThings entity types. Clauses 8.2 to 8.4 respectively explain the Create, Update and Delete protocol.

### 8.2 Create an entity

Table 23 lists integrity constraints when creating an entity.

**Table 23 – Integrity constraints when creating an entity**

Scenario	Integrity constraints
<b>Create a Thing entity</b>	–
<b>Create a Location entity</b>	–
<b>Create a Datastream entity</b>	SHALL link to a Thing entity.
	SHALL link to a Sensor entity
	SHALL link to an ObservedProperty entity.
<b>Create a Sensor entity</b>	–
<b>Create an ObservedProperty entity</b>	–
<b>Create an Observation entity</b>	SHALL link to a Datastream entity.
	SHALL link to a FeatureOfInterest entity. If no link specified, the service SHALL create a FeatureOfInterest entity from the content of the Location entities.
<b>Create a FeatureOfInterest entity</b>	–

## 8.2.1 Request

```
HTTP Method: POST
URI Pattern:SERVICE_ROOT_URI/COLLECTION_NAME
Header: Content-Type: application/json
Message Body: A single valid entity representation for the specified collection.
```

### Example 26:

```
POST /v1.0/Things HTTP/1.1
Host: example.org/
Content-Type: application/json
{
  "name":
  "thermostat",
  "description": "This is a smart thermostat with WiFi communication capabilities."
}
```

#### 8.2.1.1 Link to existing entities when creating an entity

**Example 27** – Create an Observation entity, which links to an existing Sensor entity (whose id is 1), an existing FeatureOfInterest entity (whose id is 2).

```
POST /v1.0/Observations HTTP/1.1
Host: example.org
Content-Type: application/json
{
  "Datastream":
  {
    "@iot.id": 1
  },
  "phenomenonTime": "2013-04-18T16:15:00-07:00",
  "result": 124,
  "FeatureOfInterest":
  {
    "@iot.id": 2
  }
}
```

#### 8.2.1.2 Create related entities when creating an entity

**Example 28** – Create a Thing while creating two related Sensors and one related Observation (which links to an existing FeatureOfInterest entity and an existing ObservedProperty entity).

```
POST /v1.0/Things HTTP/1.1
Host: example.org
Content-Type: application/json

{
  "description":
  "This an oven with a temperature datastream.",
  "name": "oven",
  "Locations": [
    {
      "name": "CCIT",
      "description":
      "Calgary Centre for Innovative Technologies",
      "encodingType": "application/vnd.geo+json",
      "location": {
        "type":
        "Feature",
        "geometry": {
          "type":
          "Point",
          "coordinates": [10,10]
```

```

    }
  }
},
"Datastreams": [
  {
    "name": "oven temperature",
    "description":
      "This is a datastream for an oven's internal temperature.",
    "unitOfMeasurement": {
      "name": "degree Celsius",
      "symbol": "°C",
      "definition": "http://unitsofmeasure.org/ucum.html#para-30"
    },
    "observationType": "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
    "observedArea":
      {
        "type":
          "Polygon",
        "coordinates": [[[100,0], [101,0], [101,1], [100,1], [100,0]]]
      },
    "phenomenonTime": "2009-01-11T16:22:25.00Z/2011-08-21T08:32:10.00Z",
    "Observations":
    [
      {
        "phenomenonTime": "2012-06-26T03:42:02-0600",
        "result": 70.4,
        "FeatureOfInterest": {
          "name": "CCIT #361",
          "description": "This CCIT #361, Noah's dad's office",
          "encodingType": "application/vnd.geo+json",
          "feature": {
            "type":
              "Feature",
            "geometry": {
              "type": "Polygon",
              "coordinates": [
                [[100,50], [10,9], [23,4],
                 [100,50]], [[30,20], [10,4], [4,22], [30,20]]
              ]
            }
          }
        }
      ]
    ],
    "ObservedProperty": {
      "name": "DewPoint Temperature",
      "definition": "http://sweet.jpl.nasa.gov/ontology/property.owl#DewPointTemperature",
      "description":
        "The dewpoint temperature is the temperature to which the air must be
        cooled, at constant pressure, for dew to form. As the grass and other objects
        near the ground cool to the dewpoint, some of the water vapor in the
        atmosphere condenses into liquid water on the objects."
    },
    "Sensor": {
      "name": "DS18B20",
      "description": "DS18B20 is an air temperature sensor...",
      "encodingType": "application/pdf",
      "metadata": "http://datasheets.maxim-ic.com/en/ds/DS18B20.pdf"
    }
  }
]
}

```

### 8.2.2 Response

Upon successfully creating an entity, the service response SHALL contain a Location header that contains the URL of the created entity. Upon successful completion the service SHALL respond with 201 Created. Regarding all the HTTP status code, refer to the HTTP Status Codes (section 10 of [IETF RFC 2616]).

## 8.3 Update an entity

### 8.3.1 Request

In SensorThings PATCH is the preferred means of updating an entity. PATCH provides more resiliency between clients and services by directly modifying only those values specified by the client.

The semantics of PATCH, as defined in [IETF RFC 5789], are to merge the content in the request payload with the entity's current state, applying the update only to those components specified in the request body. The properties provided in the payload corresponding to updatable properties SHALL replace the value of the corresponding property in the entity. Missing properties of the containing entity or complex property SHALL NOT be directly altered.

Services MAY additionally support PUT, but should be aware of the potential for data-loss in round-tripping properties that the client may not know about in advance, such as open or added properties, or properties not specified in metadata. Services that support PUT SHALL replace all values of structural properties with those specified in the request body. Omitting a non-nullable property with no service-generated or default value from a PUT request results in a 400 Bad Request error.

Key and other non-updatable properties that are not tied to key properties of the principal entity, can be omitted from the request. If the request contains a value for one of these properties, the service SHALL ignore that value when applying the update.

The service ignores entity id in the payload when applying the update.

The entity SHALL NOT contain related entities as inline content. It MAY contain binding information for navigation properties. For single-valued navigation properties this replaces the relationship. For collection-valued navigation properties this adds to the relationship.

On success, the response SHALL be a valid success response.

Services MAY additionally support JSON PATCH format [IETF RFC 6902] to express a sequence of operations to apply to a SensorThings entity. [OASIS OData Part1]

**HTTP Method:** PATCH or PUT

**URI Pattern:** An URI addressing to a single entity.

**Header:** Content-Type: application/json

**Message Body:** A single entity representation including a subset of properties for the specified collection.

#### Example 29 – Update the Thing whose id is 1.

```
PATCH /v1.0/Things(1) HTTP1.1
Host: example.org
Content-Type: application/json
{
  "description":"This thing is an oven."
}
```

### 8.3.2 Response

On success, the response SHALL be a valid success response. In addition, when the client sends an update request to a valid URL where an entity does not exist, the service SHALL fail the request.

Upon successful completion, the service must respond with 200 OK or 204 No Content. Regarding all the HTTP status code, please refer to the HTTP Status Code section.

## 8.4 Delete an entity

### 8.4.1 Request

A successful DELETE request to an entity's edit URL deletes the entity. The request body SHOULD be empty.

Services SHALL implicitly remove relations to and from an entity when deleting it; clients need not delete the relations explicitly.

Services MAY implicitly delete or modify related entities if required by integrity constraints. Table 24 lists SensorThings API's integrity constraints when deleting an entity.

**HTTP Method:** DELETE

**URI Pattern:** An URI addressing to a single entity.

#### Example 30 – Delete the Thing with unique identifier equals to 1

```
DELETE http://example.org/v1.0/Things(1)
```

**Table 24 – Integrity constraints when deleting an entity**

Scenario	Integrity constraints
<b>Delete a Thing entity</b>	Delete all the Datastream entities linked to the Thing entity.
<b>Delete a Location entity</b>	Delete all the HistoricalLocation entities linked to the Location entity
<b>Delete a Datastream entity</b>	Delete all the Observation entities linked to the Datastream entity.
<b>Delete a Sensor entity</b>	Delete all the Datastream entities linked to the Sensor entity.
<b>Delete an ObservedProperty entity</b>	Delete all the Datastream entities linked to the ObservedProperty entity.
<b>Delete an Observation entity</b>	–
<b>Delete a FeatureOfInterest entity</b>	Delete all the Observation entities linked to the FeatureOfInterest entity.
<b>Delete a HistoricalLocation entity entity.</b>	–

## 9 Batch requests

### 9.1 Introduction

The SensorThings service interface provides interfaces for users to perform CRUD actions on resources through different HTTP methods. However, as many IoT devices are resource-constrained, handling a large number of communications may not be practical. This clause describes how a SensorThings service can support executing multiple operations sent in a single HTTP request through the use of batch processing. This section covers both how batch operations are represented and processed. SensorThings batch request extension is adapted from [OGC OData Part1] and all

subsections. The only difference is that the OData-Version header SHOULD be omitted in SensorThings. Readers are encouraged to read the OData specification section 11.7 before reading the examples below.

## 9.2 Batch-processing request

A batch request is represented as a Multipart MIME v1.0 message [IETF RFC 2046], a standard format allowing the representation of multiple parts, each of which may have a different content type, within a single request.

The example below shows a GUID as a boundary and example.org/v1.0/ for the URI of the service.

Batch requests are submitted as a single HTTP POST request to the batch endpoint of a service, located at the URL \$batch relative to the service root (e.g., example.org/v1.0/\$batch).

NOTE – In the example, request bodies are excluded in favour of English descriptions inside '<>' brackets to simplify the example.

### Example 31-1 – A Batch Request header example

```
POST /v1.0/$batch HTTP/1.1
Host: example.org
Content-Type:
multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b
<BATCH_REQUEST_BODY>
```

NOTE – The batch request boundary must be quoted if it contains any of the following special characters:

```
() <> @
,: / " [ ] ? =
```

### 9.2.1 Batch Request body example

The following example shows a Batch Request that contains the following operations in the order listed

- A query request
- Change Set that contains the following requests:
  - Insert entity (with Content-ID = 1)
  - Update request (with Content-ID = 2)
- A second query request

NOTE – For brevity, in the example, request bodies are excluded in favour of English descriptions inside <> brackets.

Note also that the two empty lines after the Host header of the GET request are necessary: the first is part of the GET request header; the second is the empty body of the GET request, followed by a CRLF ('\r\n') according to [IETF RFC 2046].

### Example 31-2 – A Batch Request body example

```
POST /v1.0/$batch HTTP/1.1
Host: host
Content-Type: multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Length: ###

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http

Content-Transfer-Encoding:binary

GET /v1.0/Things(1)
Host: host

--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type:
multipart/mixed;boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
```

```
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
```

```
Content-Type: application/http
```

```
Content-Transfer-Encoding:
```

```
binary
```

```
Content-ID: 1
```

```
POST /v1.0/Things HTTP/1.1
```

```
Host: host
```

```
Content-Type: application/json
```

```
Content-Length: ###
```

```
<JSON representation of a  
new Thing>
```

```
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
```

```
Content-Type: application/http
```

```
Content-Transfer-Encoding:binary
```

```
Content-ID: 2
```

```
PATCH /v1.0/Things(1) HTTP/1.1
```

```
Host: host
```

```
Content-Type: application/json
```

```
If-Match: xxxxx
```

```
Content-Length: ###
```

```
<JSON representation of  
Things(1)>
```

```
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
```

```
--batch_36522ad7-fc75-4b56-8c71-56071383e77b
```

```
Content-Type: application/http
```

```
Content-Transfer-Encoding:
```

```
binary
```

```
GET /v1.0/Things(3) HTTP/1.1
```

```
Host: host
```

```
--batch_36522ad7-fc75-4b56-8c71-56071383e77b--
```

## 9.2.2 Referencing new entities in a change set example

**Example 31-3:** A Batch Request that contains the following operations in the order listed:

A change set that contains the following requests:

- 1) Insert a new Datastream entity (with Content-ID = 1)
- 2) Insert a second new entity, a Sensor entity in this example (reference request with Content-ID = 1)

```
POST  
/v1.0/$batch HTTP/1.1  
Host: host  
Content-Type:  
multipart/mixed;boundary=batch_36522ad7-fc75-4b56-8c71-56071383e77b  
  
--batch_36522ad7-fc75-4b56-8c71-56071383e77b  
Content-Type:  
multipart/mixed;boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd  
  
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd  
Content-Type:
```

```

application/http
Content-Transfer-Encoding:
binary
Content-ID:
1

POST
/v1.0/Datastreams HTTP/1.1
Host:
host
Content-Type:
application/json
Content-Length:
###

<JSON
representation of a new Datastream>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type:
application/http
Content-Transfer-Encoding:
binary
Content-ID:
2

POST
/v1.0/Sensor HTTP/1.1
Host:
host
Content-Type:
application/json
Content-Length:
###

<JSON
representation of a new Sensor>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_36522ad7-fc75-4b56-8c71-56071383e77b--

```

### 9.3 Batch-processing response

**Example 31-4** – Referencing the batch request example 31-2 above, assume all the requests except the final query request succeed. In this case the response would be:

```

HTTP/1.1
200 Ok
Content-Length:
####
Content-Type:
multipart/mixed;boundary=b_243234_25424_ef_892u748

--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding:
binary

HTTP/1.1
200 Ok
Content-Type:
application/json
Content-Length:
###

<JSON
representation of the Thing entity with id = 1>
--b_243234_25424_ef_892u748
Content-Type:
multipart/mixed;boundary=cs_12u7hdkin252452345eknd_383673037

--cs_12u7hdkin252452345eknd_383673037

```

```
Content-Type:
application/http
Content-Transfer-Encoding:
binary
Content-ID:
1

HTTP/1.1
201 Created
Content-Type:
application/json
Location: http://host/v1.0/Things(99)
Content-Length:
###

<JSON
representation of a new Thing entity>

--cs_12u7hdkin252452345eknd_383673037
Content-Type:
application/http
Content-Transfer-Encoding:
binary
Content-ID:
2

HTTP/1.1
204 No Content
Host:
host

--cs_12u7hdkin252452345eknd_383673037--
--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding:
binary

HTTP/1.1
404 Not Found
Content-Type:
application/json
Content-Length:
###

<Error
message>
--b_243234_25424_ef_892u748--
```

#### 9.4 Asynchronous batch requests

**Example 31-5** – Referencing the example 31-2 above again, assume that when interrogating the monitor URL for the first time only the first request in the batch finished processing and all the remaining requests except the final query request succeed. In this case the response would be:

```
HTTP/1.1
200 Ok
Content-Length:
####
Content-Type:
multipart/mixed;boundary=b_243234_25424_ef_892u748

--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding:
binary

HTTP/1.1
```

```
200 Ok
Content-Type:
application/json
Content-Length:
###
```

```
<JSON
representation of the Thing entity with id = 1>
--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding:
binary
```

```
HTTP/1.1
202 Accepted
Location: http://service-root/async-monitor
Retry-After:
###
```

```
--b_243234_25424_ef_892u748--
```

### Client makes a second request using the returned monitor URL:

```
HTTP/1.1
200 Ok
Content-Length:
####
Content-Type:
multipart/mixed;boundary=b_243234_25424_ef_892u748
```

```
--b_243234_25424_ef_892u748
Content-Type:
multipart/mixed;boundary=cs_12u7hdkin252452345eknd_383673037
```

```
--cs_12u7hdkin252452345eknd_383673037
Content-Type:
application/http
Content-Transfer-Encoding:
binary
Content-ID:
1
```

```
HTTP/1.1
201 Created
Content-Type:
application/json
Location: http://host/v1.0/Things(99)
Content-Length:
###
```

```
<JSON
representation of a new Thing entity>
--cs_12u7hdkin252452345eknd_383673037
Content-Type:
application/http
Content-Transfer-Encoding:
binary
Content-ID:
2
```

```
HTTP/1.1
204 No Content
Host:
host
```

```
--cs_12u7hdkin252452345eknd_383673037--
--b_243234_25424_ef_892u748
Content-Type: application/http
Content-Transfer-Encoding:
binary
```



**Table 25 – Properties of a MultiDatastream entity**

Name	Definition	Data type	Multiplicity and use
<b>name</b>	A property provides a label for Datastream entity, commonly a descriptive name.	CharacterString	One (mandatory)
<b>description</b>	The description of the Datastream entity.	CharacterString	One (mandatory)
<b>unitOfMeasurements</b>	A JSON array of JSON objects that containing three key-value pairs. The name property presents the full name of the unitOfMeasurement; the symbol property shows the textual form of the unit symbol; and the definition contains the URI defining the unitOfMeasurement. (see Req 42 for the constraints between unitOfMeasurement, multiObservationDataType and result)	A JSON array	One (mandatory) NOTE – It is possible an observation does not have a unit of measurement. For example, a count observation does not have a unit of measurement.
<b>observationType</b>	The type of Observation (with unique result type), which is used by the service to encode observations.	ValueCode and its value SHALL be OM_ComplexObservation.	One (mandatory)
<b>multiObservationDataTypes</b>	This property defines the observationType of each element of the result of a complex Observation.	A JSON array of ValueCode. See Table 14 for the available ValueCodes.	One (mandatory)
<b>observedArea</b>	The spatial bounding box of the spatial extent of all FeatureOfInterests that belong to the Observations associated with this MultiDatastream.	GM_Envelope (GeoJSON Polygon)	Zero-to-one

**Table 25 – Properties of a MultiDatastream entity**

<b>Name</b>	<b>Definition</b>	<b>Data type</b>	<b>Multiplicity and use</b>
<b>phenomenonTime</b>	The temporal interval of the phenomenon times of all observations belonging to this MultiDatastream.	TM_Period (ISO 8601 Time Interval)	Zero-to-one
<b>resultTime</b>	The temporal interval of the result times of all observations belonging to this MultiDatastream.	TM_Period (ISO 8601 Time Interval)	Zero-to-one

Table 26 lists direct relations between a MultiDatastream entity and other entity types and Table 27 list direct relations between a MultiDatastream's Observation entity and other entity types.

**Table 26 – Direct relation between a MultiDatastream entity and other entity types**

<b>Entity type</b>	<b>Relation</b>	<b>Description</b>
<b>Thing</b>	Many optional to one mandatory	A Thing has zero-to-many MultiDatastream. A MultiDatastream entity SHALL only link to a Thing as a collection of Observations.
<b>Sensor</b>	Many optional to one mandatory	The Observations in a MultiDatastream are performed by one-and-only-one Sensor. One Sensor MAY produce zero-to-many Observations in different MultiDatastreams.
<b>ObservedProperty</b>	Many optional to many mandatory	The Observations of a MultiDatastream SHALL observe the same ObservedProperties entity set.
<b>Observation</b>	One mandatory to many optional	A MultiDatastream has zero-to-many Observations. One Observation SHALL occur in one-and-only-one MultiDatastream.

**Table 27 – Direct relation between an MultiDatastream's Observation entity and other entity types**

Entity type	Relation	Description
<b>MultiDatastream</b>	Many optional to one mandatory	A MultiDatastream can have zero-to-many Observations. One Observation SHALL occur in one-and-only-one MultiDatastream.
<b>FeatureOfInterest</b>	Many optional to one mandatory	An Observation observes on one-and-only-one FeatureOfInterest. One FeatureOfInterest could be observed by one-to-many Observations.

**Example 32 – MultiDatastream entity example 1**

```
{
  "@iot.id":1,
  "@iot.selfLink":"http://example.org/v1.0/MultiDatastreams(1)",
  "Thing@iot.navigationLink":"MultiDatastreams(1)/Thing",
  "Sensor@iot.navigationLink":"MultiDatastreams(1)/Sensor",
  "ObservedProperty@iot.navigationLink":"MultiDatastreams(1)/ObservedProperties",
  "Observations@iot.navigationLink":"MultiDatastreams/Observations",
  "name": "temperature, RH, visibility",
  "description": "This is a MultiDatastream from a simple weather station measuring air temperature, relative humidity and visibility",
  "observationType":
  "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_ComplexObservation",
  "multiObservationDataTypes": [
    "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
    "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
    "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CategoryObservation"
  ],
  "unitOfMeasurements": [
    {
      "name": "degree Celsius",
      "symbol": "°C",
      "definition": " http://unitsofmeasure.org/ucum.html#para-30"
    },
    {
      "name": "percent",
      "symbol": "%",
      "definition": " http://unitsofmeasure.org/ucum.html#para-29"
    },
    {
      "name": "null",
      "symbol": "null",
      "definition": "null"
    }
  ],
  "observedArea": {
    "type": "Polygon",
    "coordinates": [
      [
        [100,0],[101,0],[101,1],[100,1],[100,0]
      ]
    ]
  },
  "phenomenonTime":"2014-03-01T13:00:00Z/2015-05-11T15:30:00Z",
  "resultTime":"2014-03-01T13:00:00Z/2015-05-11T15:30:00Z"
}
```

**Example 33 – An example ObservedProperties collection of the above MultiDatastream:** Note that the order of the elements in the value array match the order of the related Observations/result array as well as the order of the related unitOfMeasurements array.

```

{
  "value": [
    {
      "@iot.id": 1,
      "@iot.selfLink": "http://example.org/v1.0/ObservedProperties(1)",
      "Datastreams@iot.navigationLink":
        "ObservedProperties(1)/Datastreams",
      "MultiDatastreams@iot.navigationLink":
        "ObservedProperties(1)/ MultiDatastreams",
      "description":
        "The dew point is the temperature at which the water vapor in a sample
        of air at constant barometric pressure condenses into liquid water at the
        same rate at which it evaporates. At temperatures below the dew point, water
        will leave the air.",
      "name": "Dew point temperature"
    },
    {
      "@iot.id": 2,
      "@iot.selfLink": "http://example.org/v1.0/ObservedProperties(2)",
      "Datastreams@iot.navigationLink": "ObservedProperties(2)/Datastreams",
      "MultiDatastreams@iot.navigationLink": "ObservedProperties(2)/ MultiDatastreams",
      "description":
        "Relative humidity (abbreviated RH) is the ratio of the partial pressure
        of water vapor to the equilibrium vapor pressure of water at the same
        temperature.",
      "name": "Relative Humidity"
    },
    {
      "@iot.id": 3,
      "@iot.selfLink": "http://example.org/v1.0/ObservedProperties(3)",
      "Datastreams@iot.navigationLink": "ObservedProperties(3)/Datastreams",
      "MultiDatastreams@iot.navigationLink": "ObservedProperties(3)/MultiDatastreams",
      "description":
        "Visibility is a measure of the distance at which an object or light can
        be clearly discerned. ",
      "name": "Visibility (Weather)"
    }
  ]
}

```

### Example 34 – An example Observation of the above MultiDatastream

Note that the order of the elements in the result array match

- 1) the order of the related ObservedProperties (i.e., Observation(id)/MultiDatastreams(id)/ObservedProperties),
- 2) the order of the related unitOfMeasurements array (i.e., Observation(id)/MultiDatastream(id)/unitOfMeasurements) and
- 3) the order of the related multiObservationDataTypes (i.e., Observation(id)/MultiDatastream(id)/multiObservationDataTypes).

```

{
  "@iot.id": 1,
  "@iot.selfLink": "http://example.org/v1.0/Observations(1)",
  "FeatureOfInterest@iot.navigationLink":
    "Observations(1)/FeatureOfInterest",
  "MultiDatastream@iot.navigationLink":
    "Observations(1)/MultiDatastream",
  "phenomenonTime": "2014-12-31T11:59:59.00+08:00",
  "resultTime": "2014-12-31T11:59:59.00+08:00",
  "result": [
    25,
    65,
    "clear"
  ]
}

```

## 11 SensorThings Data Array Extension

Similar to the SWE DataArray in the OGC SOS, SensorThings API also provides the support of dataArray (in addition to formatting every observation entity as a JSON object) to aggregate multiple Observation entities and reduce the request (e.g., POST) and response (e.g., GET) size. SensorThings mainly use dataArray in two scenarios: (1) get Observation entities in dataArray, and (2) create Observation entities with dataArray.

### 11.1 Retrieve a Datastream's Observation entities in dataArray

In SensorThings services, users are able to request for multiple Observation entities and format the entities in the dataArray format. When a SensorThings service returns a dataArray response, the service groups Observation entities by Datastream or MultiDatastream, which means the Observation entities that link to the same Datastream or the same MultiDatastream are aggregated in one dataArray.

#### 11.1.1 Request

In order to request for dataArray, users must include the query option "\$resultFormat=dataArray" when requesting Observation entities. For example,

[http://example.org/v1.0/Observations?\\$resultFormat=dataArray](http://example.org/v1.0/Observations?$resultFormat=dataArray) .

#### 11.1.2 Response

The response Observations in dataArray format contains the properties listed in Table 28.

**Table 28 – Properties of getting Observation entities in dataArray**

Name	Definition	Data type	Multiplicity and use
<b>Datastream or MultiDatastream</b>	The navigationLink of the Datastream or the MultiDatastream entity used to group Observation entities in the dataArray.	navigationLink	One (mandatory)
<b>components</b>	An ordered array of Observation property names whose matched values are included in the dataArray.	An ordered array of Observation property names	One (mandatory)
<b>dataArray</b>	A JSON Array containing Observation entities. Each Observation entity is represented by the ordered property values, which match with the ordered property names in components.	JSON Array	One (mandatory)

### Example 35 – An example of getting Observation entities from a Datastream in dataArray result format

```
GET /v1.0/Datastreams(1)/Observations?$resultFormat=dataArray
```

```
HTTP/1.1 200 OK  
Host: www.example.org  
Content-Type: application/json
```

```
{  
  "value": [  
    {  
      "Datastream@iot.navigationLink": "Datastreams(1)",  
      "components": [  
        "id",  
        "phenomenonTime",  
        "resultTime",  
        "result"  
      ],  
      "dataArray@iot.count":3,  
      "dataArray": [  
        [  
          1,  
          "2005-08-05T12:21:13Z",  
          "2005-08-05T12:21:13Z",  
          20  
        ],  
        [  
          2,  
          "2005-08-05T12:22:08Z",  
          "2005-08-05T12:21:13Z",  
          30  
        ],  
        [  
          3,  
          "2005-08-05T12:22:54Z",  
          "2005-08-05T12:21:13Z",  
          0  
        ]  
      ]  
    }  
  ]  
}
```

### Example 36 – An example of getting Observation entities from a MultiDatastream in dataArray result format

```
GET /v1.0/MultiDatastreams(1)/Observations?$resultFormat=dataArray
```

```
HTTP/1.1 200 OK  
Host: www.example.org  
Content-Type: application/json
```

```
{  
  "value": [  
    {  
      "MultiDatastream@iot.navigationLink":  
        "MultiDatastreams(1)",  
      "components": [  
        "id",  
        "phenomenonTime",  
        "resultTime",  
        "result"  
      ],  
      "dataArray@iot.count":3,  
      "dataArray": [  
        [  
          1,  
          "2005-08-05T12:21:13Z",  
          "2005-08-05T12:21:13Z",  
          20  
        ],  
        [  
          2,  
          "2005-08-05T12:22:08Z",  
          "2005-08-05T12:21:13Z",  
          30  
        ],  
        [  
          3,  
          "2005-08-05T12:22:54Z",  
          "2005-08-05T12:21:13Z",  
          0  
        ]  
      ]  
    }  
  ]  
}
```

```

    "2010-12-23T11:20:00-0700",
    "2010-12-23T11:20:00-0700",
    [
      10.2,
      65,
      "clear"
    ]
  ],
  [
    2,
    "2010-12-23T11:22:08-0700",
    "2010-12-23T11:20:00-0700",
    [
      11.3,
      63,
      "clear"
    ]
  ],
  [
    3,
    "2010-12-23T11:22:54-0700",
    "2010-12-23T11:20:00-0700",
    [
      9.8,
      67,
      "clear"
    ]
  ]
]
}
]
}

```

## 11.2 Create Observation entities with dataArray

Besides creating Observation entities one by one with multiple HTTP POST requests, there is a need to create multiple Observation entities with a lighter message body in a single HTTP request. In this case, a sensing system can buffer multiple Observations and send them to a SensorThings service in one HTTP request. Herein an Action operation CreateObservations is used.

### 11.2.1 Request

Users can invoke the CreateObservations action by sending a HTTP POST request to the `SERVICE_ROOT_URL/CreateObservations`.

For example, `http://example.org/v1.0/CreateObservations`.

The message body aggregates Observations by Datastreams, which means all the Observations linked to one Datastream SHALL be aggregated in one JSON object. The parameters of each JSON object are shown in the following table.

As an Observation links to one FeatureOfInterest, to establish the link between an Observation and a FeatureOfInterest, users should include the FeatureOfInterest ids in the dataArray. If no FeatureOfInterest id presented, the FeatureOfInterest will be created based on the Location entities of the linked Thing entity by default.

Table 29 lists properties of creating Observation entities with dataArray.

**Table 29 – Properties of creating Observation entities with dataArray**

Name	Definition	Data type	Multiplicity and use
<b>Datastream</b>	The unique identifier of the Datastream linking to the group of Observation entities in the dataArray.	The unique identifier of a Datastream	One (mandatory)
<b>components</b>	An ordered array of Observation property names whose matched values are included in the dataArray. At least the phenomenonTime and result properties SHALL be included. To establish the link between an Observation and a FeatureOfInterest, the component name is "FeatureOfInterest/id" and the FeatureOfInterest ids should be included in the dataArray array. If no FeatureOfInterest id is presented, the FeatureOfInterest will be created based on the Location entities of the linked Thing entity by default.	An ordered array of Observation property names	One (mandatory)
<b>dataArray</b>	A JSON Array containing Observations. Each Observation is represented by the ordered property values. The ordered property values match with the ordered property names in components.	JSON Array	One (mandatory)

**Example 37 – Example of a request for creating Observation entities in dataArray**

```

POST /v1.0/CreateObservations HTTP/1.1
Host: example.org/
Content-Type: application/json

[
  {
    "Datastream": {
      "@iot.id": 1
    },
    "components": [
      "phenomenonTime",
      "result",
      "FeatureOfInterest/id"
    ],
    "dataArray@iot.count":2,
    "dataArray": [
      [
        "2010-12-23T10:20:00-0700",
        20,
        1
      ],
      [
        "2010-12-23T10:21:00-0700",
        30,
        1
      ]
    ]
  },
  {
    "Datastream": {
      "@iot.id": 2
    },
    "components": [

```

```

"phenomenonTime",
"result",
"FeatureOfInterest/id"
],
"dataArray@iot.count":2,
"dataArray": [
  [
    "2010-12-23T10:20:00-0700",
    65,
    1
  ],
  [
    "2010-12-23T10:21:00-0700",
    60,
    1
  ]
]
}
]

```

### 11.2.2 Response

Upon successful completion the service SHALL respond with 201 Created. The response message body SHALL contain the URLs of the created Observation entities, where the order of URLs must match with the order of Observations in the dataArray from the request. In the case of the service having exceptions when creating individual observation entities, instead of responding with URLs, the service must specify "error" in the corresponding array element.

#### Example 38 – An example of a response of creating Observation entities with dataArray

```

POST /v1.0/CreateObservations HTTP/1.1
201 Created
Host: example.org
Content-Type: application/json
[
  "http://examples.org/v1.0/Observations(1)",
  "error",
  "http://examples.org/v1.0/Observations(2)"
]

```

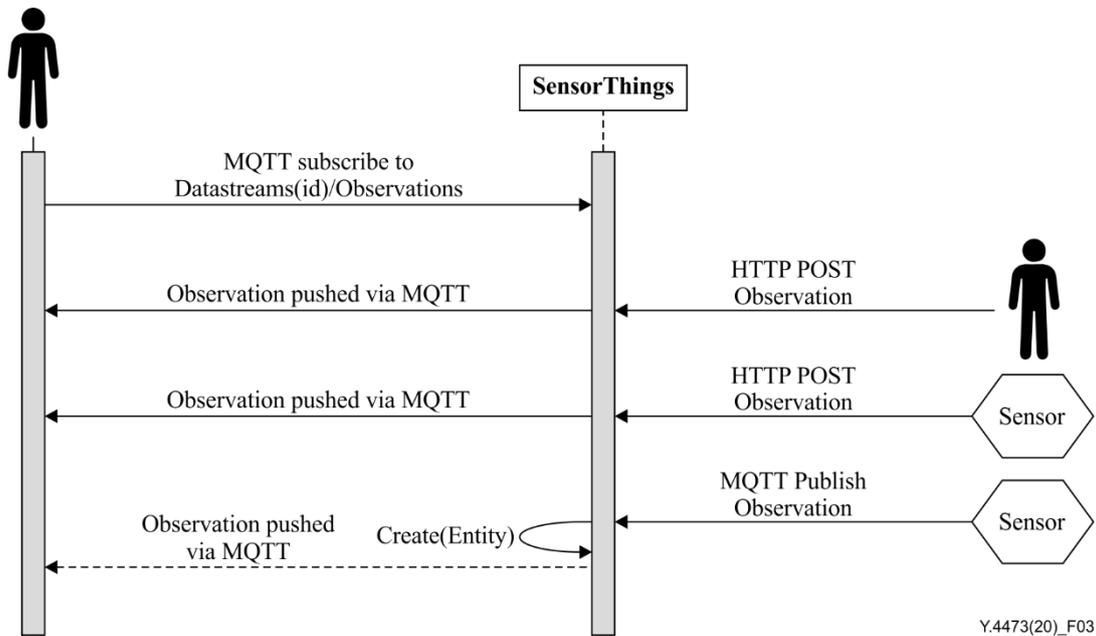
## 12 SensorThings Sensing MQTT Extension

In addition to support HTTP protocol, a SensorThings service MAY support MQTT protocol to enhance the SensorThings service publish and subscribe capabilities. This clause describes the SensorThings MQTT extension.

### 12.1 Create a SensorThings Observation with MQTT Publish

SensorThings MQTT extension provides the capability of creating Observation entity using MQTT protocol. To create an Observation entity in MQTT, the client sends a MQTT Publish request to the SensorThings service and the MQTT topic is the Observations resource path. The MQTT application message contains a single valid Observation entity representation.

Figure 3 shows the sequence diagram for creating Observation using MQTT publish as well as MQTT sending notifications for Observation creation.



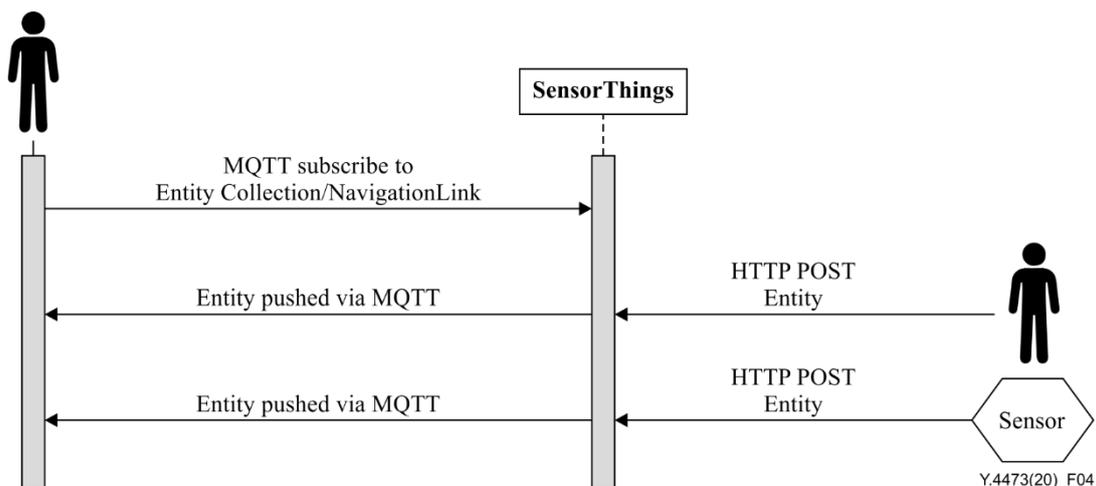
**Figure 3 – Creating Observations using MQTT publish, and receive notifications for Observations with MQTT**

If the MQTT topic for the Observation is a navigationLink from Datastream or FeatureOfInterest, the new Observation entity is automatically linked to that Datastream or FeatureOfInterest respectively.

Similar to creating Observations with HTTP POST, creating Observations with MQTT Publish follow the integrity constraints for creating Observations listed in Table 23.

## 12.2 Receive updates with MQTT Subscribe

To receive notifications from a SensorThings service when some entities updated, a client can send a MQTT Subscribe request to the SensorThings service. SensorThings API defined the following four MQTT subscription use cases. Figure 4 shows the sequence diagram of receiving updates using MQTT Subscribe.



**Figure 4 – Sequence diagram for receiving updates using MQTT subscribe**

### 12.2.1 Receive updates of a SensorThings entity set with MQTT Subscribe

MQTT Control Packet: Subscribe

**Topic Pattern:** RESOURCE\_PATH/COLLECTION\_NAME

**Example Topic:** Datastreams(1)/Observations

**Response:** When a new entity is added to the entity set (e.g., a new Observation created) or an existing entity of the entity set is updated, the service returns a complete JSON representation of the newly created or updated entity.

### 12.2.2 Receive updates of a SensorThings entity with MQTT Subscribe

MQTT Control Packet: Subscribe

**Topic Pattern:** RESOURCE\_PATH\_TO\_AN\_ENTITY

**Example Topic:** Datastreams(1)

**Response:** When a property of the subscribed entity is updated, the service returns a complete JSON representation of the updated entity.

### 12.2.3 Receive updates of a SensorThings entity's property with MQTT Subscribe

MQTT Control Packet: Subscribe

**Topic Pattern:** RESOURCE\_PATH\_TO\_AN\_ENTITY/PROPERTY\_NAME

**Example Topic:** Datastreams(1)/observedArea

**Response:** When the value of the subscribed property is changed, the service returns a JSON object. The returned JSON object follows as defined in Section 7.2.4 – Usage 4: address to a property of an entity.

**Example 39** – An example response of receiving updates of an entity's property with MQTT Subscribe. – The example shows a sample response of the following MQTT topic subscription – Datastreams(1)/description

```
{
  "description": "This is an updated description of a thing"
}
```

### 12.2.4 Receive updates of the selected properties of the newly created entities or updated entities of a SensorThings entity set with MQTT Subscribe

MQTT Control Packet: Subscribe

**Topic Pattern:**

RESOURCE\_PATH/COLLECTION\_NAME?\$select=PROPERTY\_1,PROPERTY\_2, etc.

**Response:** When a new entity is added to an entity set or an existing entity is updated (e.g., a new Observation created or an existing Observation is updated), the service returns a JSON representation of the selected properties of the newly created or updated entity.

NOTE – In the case of an entity's property is updated, it is possible that the selected properties are not the updated property, so that the returned JSON does not reflect the update.

**Example 40** – An example response of receiving updates of the selected property of an entity set with MQTT Subscribe. The example shows a sample response of the following MQTT topic subscription – Datastreams(1)/Observations?\$select=phenomenonTime,result

```
{
  "result": 45,
  "phenomenonTime": "2015-02-05T17:00:00Z"
}
```

## Bibliography

- [b-IETF RFC 5023] IETF RFC 5023 (2007), *The Atom Publishing Protocol*.  
<https://www.ietf.org/rfc/rfc5023.txt>
- [b-OGC 12-000] OGC 12-000 (2020), *SensorML: Model and XML Encoding Standard*.  
<http://www.opengeospatial.org/standards/sensorml>
- [b-UCUM] UCUM (2017), *Unified Code for Units of Measure, Version 2.1*  
<http://unitsofmeasure.org/trac>



## SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
<b>Series Y</b>	<b>Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities</b>
Series Z	Languages and general software aspects for telecommunication systems