International Telecommunication Union

ITU-T Technical Specification

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

(7 April 2019)

ITU-T Focus Group on Data Processing and Management to support IoT and Smart Cities & Communities

Technical Specification D3.2

SensorThings API – Sensing

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 procedures for establishment of focus groups are defined in Recommendation ITU-T A.7. ITU-T Study Group 20 set up the ITU-T Focus Group on Data Processing and Management to support IoT and Smart Cities & Communities (FG-DPM) at its meeting in March 2017. ITU-T Study Group 20 is the parent group of FG-DPM.

Deliverables of focus groups can take the form of technical reports, specifications, etc., and aim to provide material for consideration by the parent group in its standardization activities. Deliverables of focus groups are not ITU-T Recommendations.

Technical Specification D3.2

SensorThings API – Sensing

Summary

The SensorThings API provides an open, geospatial-enabled and unified way to interconnect the Internet of Things devices, data, and applications over the Web. The SensorThings API is an open standard, and that means it is non-proprietary, platform-independent, and perpetual royalty-free. Although it is a new standard, it builds on a rich set of proven-working and widely-adopted open standards, such as the Web protocols and the OGC Sensor Web Enablement (SWE) standards, including the ISO/OGC Observation and Measurement data model [OGC 10-004r3 and ISO 19156:2011]. That also means the SensorThings API is extensible and can be applied to not only simple but also complex use cases.

The SensorThings API – Sensing provides a standard way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems. The SensorThings API follows the REST principles, the use of an efficient JSON encoding, the use of MQTT protocol, the use of the flexible OASIS OData protocol and URL conventions.

Acknowledgements

This Technical Specification was researched and principally authored by Steve Liang (Open Geospatial Consortium), Tania Khalafbeigi (SensorUp and University of Calgary) under supervision of Gyu Myoung Lee (Korea, Rep.of).

Additional information and materials relating to this Technical Specification can be found at: www.itu.int/go/tfgdpm. If you would like to provide any additional information, please contact Denis Andreev at tsbfgdpm@itu.int.

Keywords

Internet of things; iot; ogc; sensor web; smart cities; smart communities

Technical Specification D3.2

SensorThings API – Sensing

Table of Contents

Technical Specification ITU-T D3.2

SensorThings API – Sensing

1 Scope

The SensorThings API provides an open standard-based and geospatial-enabled framework to interconnect the Internet of Things devices, data, and applications over the Web.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Technical Specification. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Technical Specification 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 Technical Specification does not give it, as a stand-alone document, the status of a Recommendation.

[ISO 8601:2004] Data elements and interchange formats – Information interchange - Representation of dates and times.

[OGC/ISO 19156:2011(E)] OGC Abstract Specification Topic 20: Geographic information — Observations and Measurements

OASIS OData Version 4.0 Part 1: Protocol Plus Errata 02

OASIS OData Version 4.0 Part 2: URL Conventions Plus Errata 02

OASIS OData JSON Format Version 4.0 Plus Errata 02

OASIS OData ABNF Construction Rules Errata 02

[RFC 2046] Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types

[RFC 2616] Hypertext Transfer Protocol -- HTTP/1.1

[RFC 4627] Media Type for Javascript Object Notation (JSON), July 2006

Unified Code for Units of Measure (UCUM) – Version 1.9, April 2015

3 Definitions

3.1 Terms defined elsewhere

These Technical Specifications use the following terms defined elsewhere:

3.1.1 Collection: Sets of Resources, which can be retrieved in whole or in part. [RFC5023]

3.1.2 Entity: Entities are instances of entity types. [OASIS OData Version 4.0 Part 1: Protocol Plus Errata 02]

3.1.3 Entity sets: Entity sets are 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. [OASIS OData Version 4.0 Part 1: Protocol Plus Errata 02]

3.1.4 Internet of Thing: 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.2060]

3.1.5 Measurement: A set of operations having the object of determining the value of a quantity [OGC 10-004r3 / ISO 19156:2011]

3.1.6 Observation: Act of measuring or otherwise determining the value of a property [OGC 10-004r3 / ISO 19156:2011]

3.1.7 Observation Result: Estimate of the value of a property determined through a known observation procedure [OGC 10-004r3 / ISO 19156:2011]

3.1.8 Resource: A network-accessible data object or service identified by an URI, as defined in [RFC 2616]

3.1.9 Sensor: 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. [OGC 12-000]

3.2 Terms defined in these Technical Specifications

These Technical Specifications defines the following terms:

3.2.1 REST: The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system. REST focuses on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behavior as a network-based application. An API that conforms to the REST architectural principles/constraints is called a RESTful API.

4 Abbreviations and acronyms

These Technical Specifications use the following abbreviations and acronyms:

API	Application Programming Interface
CS-W	Catalog Service Web
CRUD	Create, Read, Update, and Delete
GML	Geography Markup Language
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol

IoT	Internet of Things
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
OData	Open Data Protocol
OGC	Open Geospatial Consortium
OWS	OGC Web Services
O&M	Observations and Measurements
REST	REpresentational State Transfer
SensorML	Sensor Model Language
SOS	Sensor Observation Service
SPS	Sensor Planning Service
SWE	Sensor Web Enablement
UCUM	Unified Code for Units of Measure
UML	Unified Modeling Language
WoT	Web of Things
XML	eXtensible Markup Language

5 Conventions

None

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 (.). 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 Common control information

Annotation	Definition	Data type	Multiplicity and use
id	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)
selfLink	selfLink is the absolute URL of an entity that is unique among all other entities.	URL	One (mandatory)
navigationLink	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.

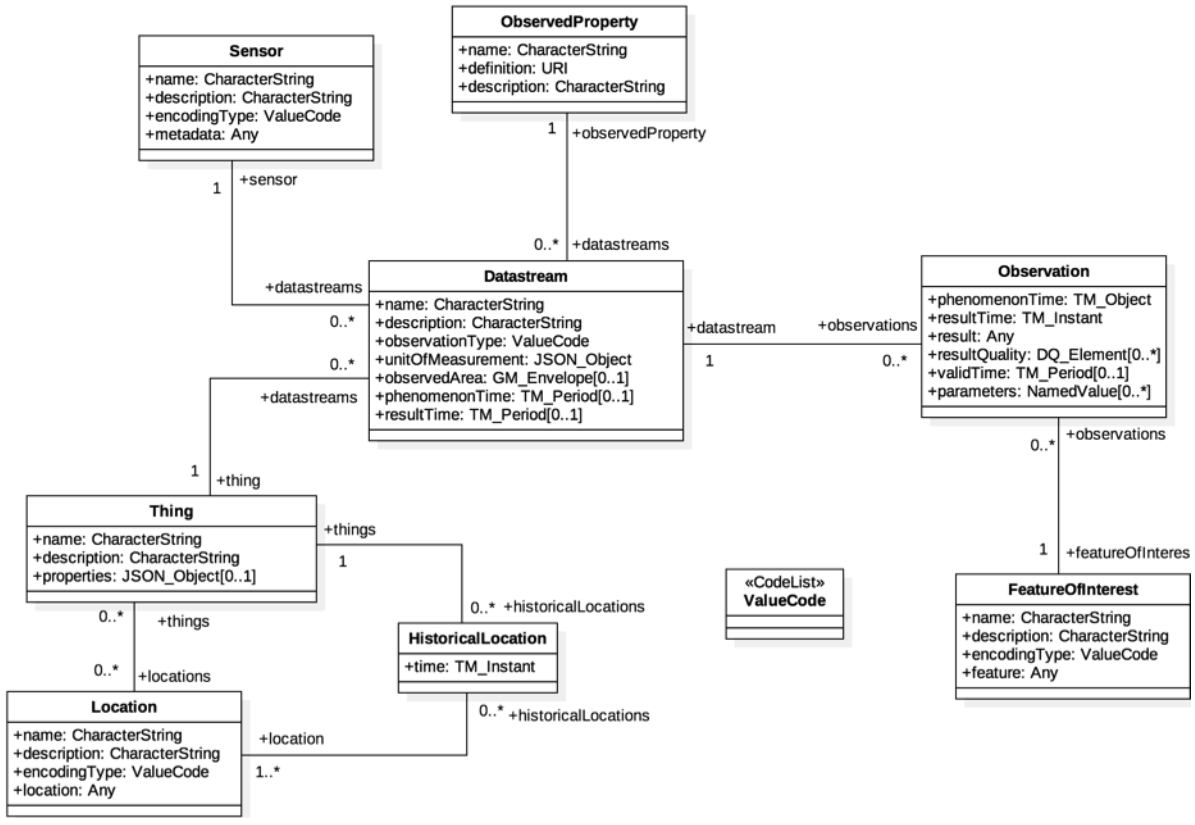

Figure 1 Sensing entities

In this section, we explain the properties in each entity type and the direct relation to the other entity types. In addition, for each entity type, we show an example of the associated JSON encoding.

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.2060].

Table 2 Properties of a Thing entity

Name	Definition	Data type	Multiplicity and use
name	A property provides a label for Thing entity, commonly a descriptive name.	CharacterString	One (mandatory)
description	This is a short description of the corresponding Thing entity.	CharacterString	One (mandatory)
properties	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
Location	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).
HistoricalLocation	One mandatory to many optional	A Thing has zero-to-many HistoricalLocations. A HistoricalLocation has one-and-only-one Thing.
Datastream	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 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. And 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 theFeatureOfInterestof the Thing's Observations. Section 7.1.4 of [OGC 10-004r3 and ISO 19156:2011] provides a detailed explanation of observation location.

Table 4 Properties of a Location entity

Name	Definition	Data type	Multiplicity and use
name	A property provides a label for Location entity, commonly a descriptive name.	CharacterString	One (mandatory)

description	The description about the Location.	CharacterString	One (mandatory)
encodingType	The encoding type of the Location property. Its value is one of the ValueCode enumeration.	ValueCode	One (mandatory)
location	The location type is defined by encodingType.	Any (*i.e.*, the type is depending on the value of the encodingType)	One (mandatory)

Table 5 Direct relation between a Location entity and other entity types

Entity type	Relation	Description
Thing	Many optional to many optional	Multiple Things MAY locate at the same Location. A Thing MAY not have a Location.
HistoricalLocation	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 Innvative Technologies",
  "location": {
    "type": "Feature",
    "geometry": {
      "type": "Point",
      "coordinates": [
        -114.06,
        51.05
```

```
        ]
      }
    }
}
```

Table 6 List of some code values used for identifying types for the encodingType of the Location and FeatureOfInterest entity

Location encodingType	ValueCode Value
GeoJSON	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 we expect to extend SensorThings API's 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 Properties of a HistoricalLocation entity

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

Table 8 Direct relation between an HistoricalLocation entity and other entity types

Entity type	Relation	Description
Location	Many optional to many mandatory	A Location can have zero-to-many HistoricalLocations. One HistoricalLocation SHALL have one or many Locations.

Thing	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 Properties of a Datastream entity

Name	Definition	Data type	Multiplicity and use
name	A property provides a label for Datastream entity, commonly a descriptive name.	CharacterString	One (mandatory)
description	The description of the Datastream entity.	CharacterString	One (mandatory)
unitOfMeasurement	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., a OM_TruthObservation type), the corresponding unitOfMeasurement properties SHALL have null values.
observationType	The type of Observation (with unique result type), which is used by the service to encode observations.	ValueCode see Table 12.	One (mandatory)
observedArea	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)
phenomenonTime	The temporal interval of the phenomenon times of all observations belonging to this Datastream.	TM_Period (ISO 8601 Time Interval)	Zero-to-one (optional)
resultTime	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
Thing	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.
Sensor	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.
ObservedProperty	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.
Observation	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 10-004r3 and ISO 19156:2011 Table 3]. The following table shows some of the valueCodes that maps the UML classes in O&M v2.0 [OGC 10-004r3 and ISO 19156:2011] 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 10-004r3 and ISO 19156:2011 Clause 8.2.2)

O&M 2.0	Value Code Value (observationType names)	Content of result
OM_CategoryObservation	http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CategoryObservation	URI
OM_CountObservation	http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_CountObservation	integer
OM_Measurement	http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement	double
OM_Observation	http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Observation	Any
OM_TruthObservation	http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_TruthObservation	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 property1.

---

[1] In some cases, the Sensor in this data model can also be seen as the Procedure (method, algorithm, or instrument) defined in [OGC 10-004r3 and ISO 19156:2011].

Table 12 Properties of a Sensor entity

Name	Definition	Data type	Multiplicity and use
name	A property provides a label for Sensor entity, commonly a descriptive name.	CharacterString	One (mandatory)
description	The description of the Sensor entity.	CharacterString	One (mandatory)
encodingType	The encoding type of the metadata property. Its value is one of the ValueCode enumeration (Table 14**.** for the available ValueCode).	ValueCode	One (mandatory)
metadata	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 relation between a Sensor entity and other entity types

Entity type	Relation	Description
Datastream	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	http://www.opengis.net/doc/IS/SensorML/2.0

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 Properties of an ObservedProperty entity

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

Table 16 Direct relation between an ObservedProperty entity and other entity types

Entity type	Relation	Description
Datastream	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 10-004r3 and ISO 19156:2011]

Table 17 Properties of an Observation entity

Name	Definition	Data type	Multiplicity and use
phenomenonTime	The time instant or period of when the Observation happens. Note: Many resource-constrained sensing devices do not have a clock. As a result, a client may omit phenonmenonTime when POST new Observations, even though phenonmenonTime is a mandatory property. When a SensorThings service receives a POST Observations without phenonmenonTime, the service SHALL assign the current server time to the value of the phenomenonTime.	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))	One (mandatory)

result	The estimated value of an ObservedProperty from the Observation.	Any (depends on the observationType defined in the associated Datastream)	One (mandatory)
resultTime	The time of the Observation's result was generated. 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.	TM_Instant (ISO 8601 Time string)	One (mandatory)
resultQuality	Describes the quality of the result.	DQ_Element	Zero-to-many
validTime	The time period during which the result may be used.	TM_Period (ISO 8601 Time Interval string)	Zero-to-one
parameters	Key-value pairs showing the environmental conditions during measurement.	NamedValues in a JSON Array	Zero-to-one

Table 18 Direct relation between an Observation entity and other entity types

Entity type	Relation	Description
Datastream	Many optional to one mandatory	A Datastream can have zero-to-many Observations. One Observation SHALL occur in one-and-only-one Datastream.
FeatureOfInterest	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 and ISO 19156:2011]. 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-connect thermostat can be the Location of the thermostat (i.e., the living room where the thermostat is located in). In the case of remote sensing, the FeatureOfInterest can be the geographical area or volume that is being sensed.

Table 19 Properties of a FeatureOfInterest entity

Name	Definition	Data type	Multiplicity and use
name	A property provides a label for FeatureOfInterest entity, commonly a descriptive name.	CharacterString	One (mandatory)
description	The description about the FeatureOfInterest.	CharacterString	One (mandatory)
encodingType	The encoding type of the feature property. Its value is one of the ValueCode enumeration (see **Error! Reference source not found.** for the available ValueCode).	ValueCode	One (mandatory)
feature	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
Observation	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 a service document resources that describe its data model. The service document lists the entity sets that can be 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

```
http://example.org/v1.0/Observations?$orderby=ID&$top=10
_____________________/___________/___________________/
          |                |              |
  service root URI     resource path   query options
```

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. The following lists 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. [Adapted from OData 4.0-JSON-Format section 5]

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. [Adapted from OData 4.0-JSON-Format section 12]

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\n  which the water vapor in air at constant barometric pressure condenses into\n  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\n  partial pressure of water vapor in an air-water mixture to the saturated\n  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 to 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. [Adapted from OData 4.0-Protocol 11.2.3]

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 ISO8601 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 pairs. 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
```

turns 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 as follows. 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. [Adapted from OData 4.0- URL 5.1.2]

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 operations

The SensorThings API supports a set of built-in filter operations, as described in the following table. These built-in filter operator usages and definitions follow the [OData Specification Section 11.2.5.1.1] and [OData Version 4.0 ABNF].

Table 21 Built-in Filter Operators

Operator	Description	Example
Comparison Operators		
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?$filter=result le 100
Logical Operators		
and	Logical and	/Observations?$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')
Arithmetic Operators		
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
mod	Modulo	/Observations?$filter=result mod 2 eq 0
Grouping Operators		
( )	Precedence grouping	/Observations?$filter=(result sub 5) gt 10

7.3.3.5.2 Built-in query functions

The SensorThings API supports a set of functions that can be used with the $filter or $orderby query operations. The following table lists the available functions and they follows the OData Canonical function definitions listed in Section 5.1.1.4 of the [OData Version 4.0 Part 2: URL Conventions] and the syntax rules for these functions are defined in [OData Version 4.0 ABNF].

In order to support spatial relationship functions, 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 [OGC 06-104r4 part 1, clause 6.1.2.3]. The names of these nine functions start with a prefix “st_” following the OGC Simple Feature Access specification [OGC 06-104r4]. In addition, the Well-Known Text (WKT) format is the default input geometry for these nine functions.

Table 22 Built-in Query Functions

Function	Example
String Functions	
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 'ensor Things'
string tolower(string p0)	tolower(description) eq 'sensor things'
string toupper(string p0)	toupper(description) eq 'SENSOR THINGS'
string trim(string p0)	trim(description) eq 'Sensor Things'
string concat(string p0, string p1)	concat(concat(unitOfMeasurement/symbol,', '), unitOfMeasurement/name) eq 'degree, Celsius'
Date Functions	
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()
Math Functions	
round	round(result) eq 32
floor	floor(result) eq 32
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 **(nextLink)**

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, Delete actions can be performed on the SensorThings entity types. The following subsection explains the Create, Update, and Delete protocol.

8.2 Create an entity

Table 23 Integrity constraints when creating an entity

Scenario	Integrity Constraints
Create a Thing **entity**	-
Create a Location **entity**	-
Create a Datastream **entity**	SHALL link to a Thing entity.
	SHALL link to a Sensor entity
	SHALL link to an ObservedProperty entity.
Create a Sensor **entity**	-
Create an ObservedProperty **entity**	-
Create an Observation **entity**	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.
Create a FeatureOfInterest **entity**	-

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 HTTP1.1
  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#DewPointTemperatu
re",
          "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, please refer to the HTTP Status Code section.

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 [RFC5789], 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 [RFC6902] to express a sequence of operations to apply to a SensorThings entity. [Adapted from OData 4.0-Protocol 11.4.3]

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 25 listed 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
Delete a Thing **entity**	Delete all the Datastream entities linked to the Thing entity.
Delete a Location **entity**	Delete all the HistoricalLocation entities linked to the Location entity
Delete a Datastream **entity**	Delete all the Observation entities linked to the Datastream entity.
Delete a Sensor **entity**	Delete all the Datastream entities linked to the Sensor entity.
Delete an ObservedProperty **entity**	Delete all the Datastream entities linked to the ObservedProperty entity.
Delete an Observation **entity**	-

Delete a FeatureOfInterest **entity**	Delete all the Observation entities linked to the FeatureOfInterest entity.
Delete a HistoricalLocation entity **entity.**	-

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 section 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 [OData 4.0 Protocol 11.7] 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 [RFC2046], 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 favor 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 favor 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 according to [RFC2046].

[Adapted from OData 4.0 Protocol 11.7.2]

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

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

<Error
message>
--b_243234_25424_ef_892u748—
```

10. SensorThings MultiDatastream extension

Observation results may have many data types, including primitive types like category or measure, but also more complex types such as time, location and geometry [OGC 10-004r3 and ISO 19156:2011]. SensorThings' MultiDatastream entity is an extension to handle complex observations when the result is an array.

A MultiDatastream groups a collection of Observations and the Observations in a MultiDatastream have a complex result type.

The MultiDatastream extension entities are depicted in Figure 2.

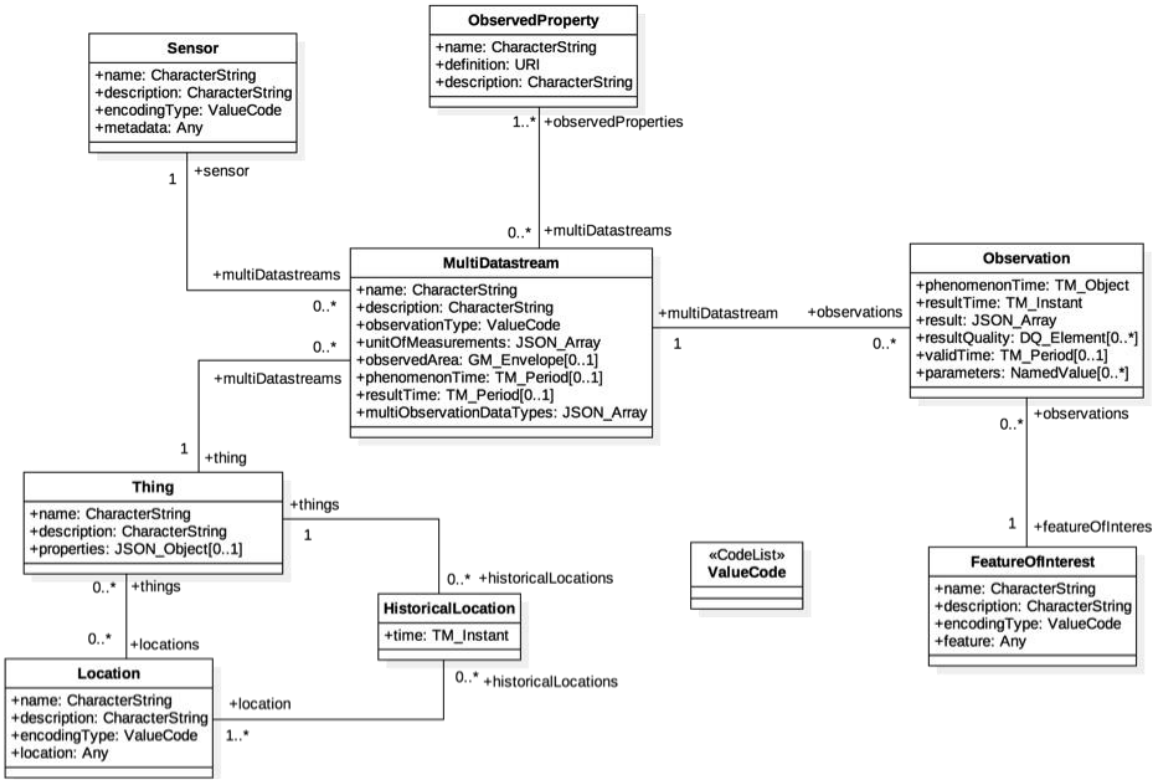

Figure 2 MultiDatastream Extension Entities

Table 25 Properties of a MultiDatastream entity

Name	Definition	Data type	Multiplicity and use
name	A property provides a label for Datastream entity, commonly a descriptive name.	CharacterString	One (mandatory)
description	The description of the Datastream entity.	CharacterString	One (mandatory)
unitOfMeasurements	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	A JSON array	One (mandatory) Note: It is possible an observation does not have a

	the unit symbol; and the definition contains the URI defining the unitOfMeasurement. (see Req 42 for the constraints between unitOfMeasurement, multiObservationDataType and result)		unit of measurement. For example, a count observation does not have a unit of measurement.
observationType	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)
multiObservationDataTypes	This property defines the observationType of each element of the result of a complex Observation.	A JSON array of ValueCode. See Table 12 for the available ValueCodes.	One (mandatory)
observedArea	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
phenomenonTime	The temporal interval of the phenomenon times of all observations belonging to this MultiDatastream.	TM_Period (ISO 8601 Time Interval)	Zero-to-one
resultTime	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 Direct relation between a MultiDatastream entity and other entity types

Entity type	Relation	Description
Thing	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.
Sensor	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.
ObservedProperty	Many optional to many mandatory	The Observations of a MultiDatastream SHALL observe the same ObservedProperties entity set.
Observation	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
MultiDatastream	Many optional to one mandatory	A MultiDatastream can have zero-to-many Observations. One Observation SHALL occur in one-and-only-one MultiDatastream.
FeatureOfInterest	Many optional to one mandatory	An Observation observes on one-and-only-one FeatureOfInterst. 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: Please 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: Please 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.

11.1.2 Response

The response Observations in dataArray format contains the following properties.

Table 28 Properties of getting Observation entities in dataArray

Name	Definition	Data type	Multiplicity and use
Datastream or MultiDatastream	The navigationLink of the Datastream or the MultiDatastream entity used to group Observation entities in the dataArray.	navigationLink	One (mandatory)
components	An ordered array of Observation property names whose matched values are included in the dataArray.	An ordered array of Observation property names	One (mandatory)
dataArray	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,
                   "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. Here we propose an Action operation CreateObservations.

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 Properties of creating Observation entities with dataArray

Name	Definition	Data type	Multiplicity and use
Datastream	The unique identifier of the Datasteam linking to the group of Observation entities in the dataArray.	The unique identifier of a Datastream	One (mandatory)
components	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)
dataArray	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 section 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 contains the sequence diagram for creating Observation using MQTT publish as well as MQTT sending notifications for Observation creation.

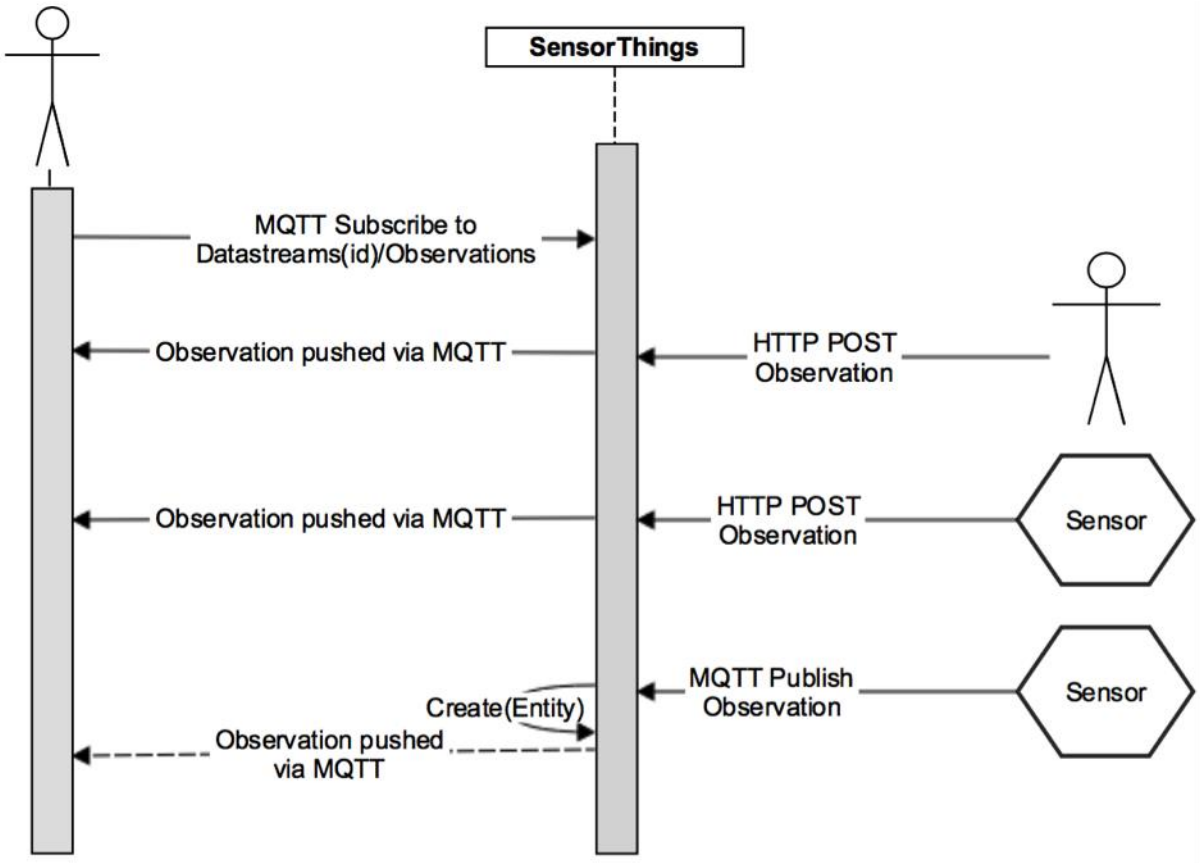

Figure 3 Creating Observations using MQTT publish, and receive notifications for Observations with MQTT

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 Observation 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 contains the sequence diagram of receiving updates using MQTT Subscribe.

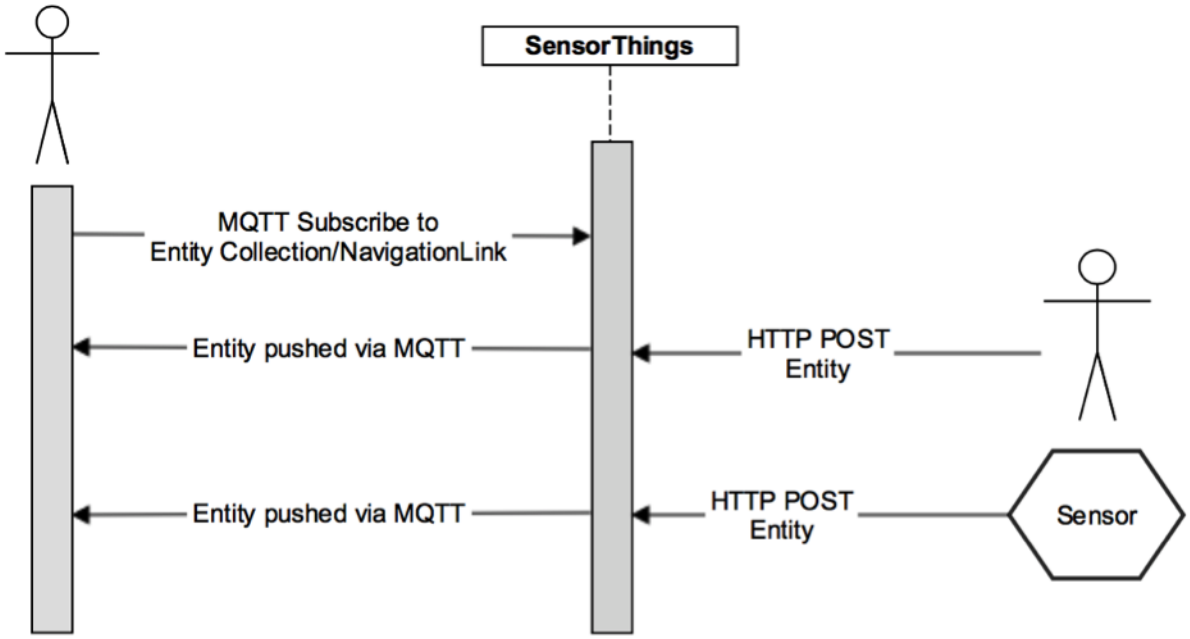

Figure 4 Sequence diagram for receiving updates using MQTT subscribe

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 9.2.4 - 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,…

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,
   "phenonmenonTime": "2015-02-05T17:00:00Z"
}
```

Bibliography

[1] The GeoJSON Format Specification, January 15, 2015. Available Online: https://datatracker.ietf.org/doc/draft-butler-geojson/

[2] ITU-T Y.2060 Overview of the Internet of Things, 2012. Available Online: https://www.itu.int/rec/T-REC-Y.2060-201206-I

[3] OGC and ISO 19156:2001, OGC 10-004r3 and ISO 19156:2011(E), OGC Abstract Specification: Geographic information — Observations and Measurements. Available Online: http://portal.opengeospatial.org/files/?artifact_id=41579

[4] OGC 12-000, OGC® SensorML: Model and XML Encoding Standard. Available Online: http://www.opengeospatial.org/standards/sensorml

[5] RFC 5023, The Atom Publishing Protocol. Available Online: https://www.ietf.org/rfc/rfc5023.txt

[6] RFC 6902, JavaScript Object Notation (JSON) Patch. Available Online: https://www.ietf.org/rfc/rfc6902.txt