Open Geospatial Consortium

Submission Date: 2015-06-18

Approval Date: <yyyy-dd-mm>

Publication Date: <yyyy-dd-mm>

External identifier of this OGC® document: <http://www.opengis.net/doc/IS/iot_sensing/1.0>

Internal reference number of this OGC® document: 15-078

Version: 0.9.4

Category: OpenGIS® Candidate Standard

Editor: Steve Liang (University of Calgary/SensorUp)
Co-Editors: Chih-Yuan Huang (National Central University)
Tania Khalafbeigi (University of Calgary/SensorUp)

OGC® SensorThings API

Part 1: Sensing

Document type:	OGC® Candidate Standard
Document subtype:	if applicable
Document stage:	Draft
Document language:	English

Contents

i. Abstract

The OGC SensorThings API provides an open and unified way to interconnect Internet of Things (IoT) devices, data, and applications over the Web. At a high level the OGC SensorThings API provides two main functionalities and each function is handled by a profile. The two profiles are the Sensing profile and the Tasking profile. The Sensing profile provides a standard way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems. The Tasking profile is planned as a future work activity and will be defined in a separate document as Part 2 of the SensorThings API.

ii. Keywords

ogcdoc, ogc documents, iot, internet of things, sensor things, sensors, swe

iii. Submitting organizations

The following organizations submitted this document to the Open Geospatial Consortium.

a. University of Calgary, Canada

b. National Central University, Taiwan

c. Lockheed Martin, USA

d. AIST, Japan

e. FCU.GIS, Taiwan

f. ITRI, Taiwan

g. GeoConnections, Canada

h. Noblis, USA

i. Fraunhofer, Germany

iv. Document contributor contact points

All questions regarding this document should be directed to the editor or the contributors:

Contact	**Company**
Steve Liang	University of Calgary, Canada / SensorUp Inc.
Alec Chih-Yuan Huang	National Central University, Taiwan
Tania Khalafbeigi	University of Calgary, Canada / SensorUp Inc.
Kyoungsook Kim	AIST, Japan

Contact	Company
Thomas Schwab	Lockheed Martin
Jean Brodeur	NRCan/Canadian Geospatial Data Infrastructure
Marcus Alzona	Noblis

v. Revision history

Date	Release	Editor	Primary clauses modified	Description
15/01/24	0.9.0	Steve Liang	Many.	Updated according to the Calgary and Tokyo TC SWG meeting. Changed trajectory to HistoricalLocation.
15/02/24	0.9	Carl Reed	Many.	Major edit and alignment with OGC document template.
15/04/11	0.9	Steve Liang	Many.	Updated according to the Barcelona TC SWG meeting. Changed ComplexDatastream to MultiDatastream.
15/05/11	0.9.1	Steve Liang	Many.	Updated according to the OGC Architecture Board meeting on May 5th 2015. Added requirement classes.
15/06/11	0.9.2	Steve Liang	Many.	Updated according to the OGC Architecture Board meeting on May 19th 2015. Revised requirement classes to follow the OGC Naming Authority. Changed common properties to common control information. Added support to server-side pagination for expanded related entities.

vi. Changes to the OpenGIS® Abstract Specification

The OGC® Abstract Specification does not require changes to accommodate this OGC® standard.

vii. Future work

- Finish SensorThings API - Tasking profile
- Explore potential harmonization with OASIS OData Protocol.
- Support multiple data models and encodings for `Location` entity type

Foreword

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. The Open Geospatial Consortium shall not be held responsible for identifying any or all such patent rights.

Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the standard set forth in this document, and to provide supporting documentation.

Introduction

The OGC SensorThings API provides an open and unified way to interconnect Internet of Things devices, data, and applications over the Web. The OGC SensorThings API is an open standard: 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 Web protocols and the OGC Sensor Web Enablement (SWE) standards, including the ISO/OGC Observation and Measurement data model [OGC and ISO 19156:2011]. As such, the OGC SensorThings API is extensible and can be applied to both simple and complex use cases.

At a high level, the OGC SensorThings API provides two main functionalities, each handled by a profile. The two profiles are the Sensing profile and the Tasking profile. The Sensing profile provides a standard way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems. This document defines the Sensing profile as Part 1 of the SensorThings API. The Tasking profile provides a standard way for parameterizing - also called tasking - of task-able IoT devices, such as sensors or actuators. The Tasking profile is planned as a future work activity and will be defined in a separate document as Part 2 of the SensorThings API.

The Sensing profile provides functions similar to the OGC Sensor Observation Service (SOS) and the Tasking profile will provide functions similar to the OGC Sensor Planning Service (SPS). The main difference between the SensorThings API and the OGC SOS and SPS is that the SensorThings API is designed specifically for resource-constrained IoT devices and the Web developer community. As a result, the SensorThings API follows REST principles, the use of an efficient JSON encoding, and the use of the flexible OASIS OData protocol and URL conventions.

OpenGIS ® SensorThings API

1. Scope

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

2. Conformance

Conformance with this standard shall be checked using all the relevant tests specified in Annex A (normative), Abstract Test Suite.

The following tables list the requirements classes defined by this standard.

NOTE: The smaller blue text in the following tables is the path fragment that appended to the following URI: http://www.opengis.net/spec/iot_sensing/1.0/, and it provides the URI that can be used to unambiguously identify the requirement and the conformance class.

Requirements class id	**Requirements**	**Description**
req/entities	• req/core/common-control-information • req/core/thing-properties • req/core/location-properties • req/core/historical-location-properties • req/core/datastream-properties • req/core/sensor-properties • req/core/observed-property-properties • req/core/observation-properties • req/core/feature-of-interest-properties • req/core/thing-relations • req/core/location-relations • req/core/historical-location-relations • req/core/datastream-relations • req/core/sensor-relations • req/core/observed-property-relations • req/core/observation-relations • req/core/feature-of-interest-relations • req/core/resource-path-to-entities • req/core/read-entity	Entities of the SensorThings API service and addressing to the entities

Requirements class id	Requirements	Description
req/request-data	• req/request-data/order • req/request-data/expand • req/request-data/select • req/request-data/status-code • req/request-data/query-status-code • req/request-data/orderby • req/request-data/top • req/request-data/skip • req/request-data/count • req/request-data/filter • req/request-data/built-in-filter-operations • req/request-data/built-in-filter-functions • req/request-data/pagination	Requesting data with system query options
req/create-update-delete	• req/create-update-delete/create-entity • req/create-update-delete/link-to-existing-entities • req/create-update-delete/deep-insert • req/create-update-delete/deep-insert-status-code • req/create-update-delete/update-entity • req/create-update-delete/delete-entity • req/historical-location-auto-creation	Creating, updating, and deleting entities
req/batch-request	• req/batch-request/batch-request	Processing multiple requests with a single request
req/multi-datastream	• req/multi-datastream/properties • req/multi-datastream/relations • req/multi-datastream/constraints	Handling complex observations with complex results, especially when the result is an array
req/data-array	• req/data-array/data-array	Serving Observations with the efficient data array encoding

Requirements class id	Requirements	Description
req/mqtt	• req/mqtt/create • req/mqtt/update • req/mqtt/receive-update	Receiving updates, creating and updating entities through MQTT

3. Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this document. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. For undated references, the latest edition of the normative document referred to applies.

ISO 8601:1988(E), Data elements and interchange formats – Information interchange - Representation of dates and times.
OGC and ISO 19156:2011(E), OGC Abstract Specification: 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, the application/json Media Type for Javascript Object Notation (JSON), July 2006
Unified Code for Units of Measure (UCUM) – Version 1.9, April 2015

4. Terms and definitions

For the purposes of this document, the following terms and definitions apply:

Collection

Sets of Resources, which can be retrieved in whole or in part [RFC5023]

Entity

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

Note: Thing, Sensor, Datastream, Observation are some example entity types of the OGC SensorThings API

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]

(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]

Measurement

A set of operations having the object of determining the value of a quantity [OGC and ISO 19156:2011]

Observation

Act of measuring or otherwise determining the value of a property [OGC and ISO 19156:2011]

Observation Result

Estimate of the value of a property determined through a known observation procedure [OGC and ISO 19156:2011]

Resources

A network-accessible data object or service identified by an IRI, as defined in [RFC 2616]

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 has REST architecture is called a RESTful API

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]

5. Conventions

5.1 Presentation of Requirements and Recommendations

Requirements are presented using the following style:

Req <number>	<requirement text>
<requirement id>	

<number> is a unique number within the document.

<requirement text> is the requirement itself. Normative verbs like SHALL are written in capitals.

The smaller blue text at the bottom of the box <requirement id> is the path and it provides the URI of the requirement, which can be used to unambiguously identify the requirement.

Normative verbs like SHALL are written in capitals.

6. Symbols (and abbreviated terms)

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	the 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

7. SensorThings API overview

7.1 Who should use the OGC SensorThings API

Organizations that need a web-based platforms to manage, store, share, analyze IoT-based sensor observation data should use the OGC SensorThings API. The OGC SensorThings API simplifies and accelerates the development of IoT applications. Application developers can use this open standard to connect to various IoT devices and create innovative applications without worrying the daunting heterogeneous protocols of the different IoT devices, gateways and services. IoT device manufacturers can also use OGC SensorThings API as the API can be embedded within various IoT hardware and software platforms, so that the various IoT devices can effortlessly connect with the OGC standard-compliant servers around the world. In summary, the OGC SensorThings API is transforming the numerous disjointed IoT systems into a fully connected platform where complex tasks can be synchronized and performed.

7.2 Benefits of the OGC SensorThings API

In today's world, most IoT devices (e.g., sensors and actuators) have proprietary software interfaces defined by their manufacturers and used selectively. New APIs are often required and developed on an as needed basis, often in an environment with resource limitations and associated risks. This situation requires significant investment on the part of developers for each new sensor or project involving multiple systems and on the part of the providers of sensors, gateways and portals or services where observations and measurements are required.

As a standardized data model and interface for sensors in the WoT and IoT[1], the OGC SensorThings API offers the following benefits: (1) it permits the proliferation of new high value services with lower overhead of development and wider reach, (2) it lowers the risks, time and cost across a full IoT product cycle, and (3) it simplifies both devices-to-devices and devices-to-applications.

8. The SensorThings API Sensing Profile Entities

8.1 Overview

The OGC SensorThings API data model consists of two parts: (1) the Sensing profile and (2) the Tasking profile. The Sensing profile allows IoT devices and applications to CREATE, READ, UPDATE, and DELETE (*i.e.*, HTTP `POST`, `GET`, `PATCH`, and `DELETE`) IoT data and metadata in a SensorThings service.

Managing and retrieving observations and metadata from IoT sensor systems is one of the most common use cases. As a result, the Sensing profile is designed based on the ISO/OGC Observation and Measurement (O&M) model [OGC and ISO 19156:2011].

The key to the model is that an `Observation` is modeled as an act that produces a `result` whose value is an estimate of a property of the observation target or `FeatureOfInterest`. An `Observation` instance is classified by its event time (e.g., `resultTime` and `phenonmenonTime`), `FeatureOfInterest`, `ObservedProperty`, and the procedure used (often a `Sensor`).

Moreover, `Things` are also modeled in the SensorThings API. Further the geographical `Locations` of `Things` are useful in almost every application and as a result are included as well.

In the Sensing profile, a `Thing` has `Locations` and `HistoricalLocations`. A `Thing` also can have multiple `Datastreams`. A `Datastream` is a collection of `Observations` grouped by the same `ObservedProperty` and `Sensor`. An `Observation` is an event performed by a `Sensor` that produces a `result` whose value is an estimate of an `ObservedProperty` of the `FeatureOfInterest`.

8.2 Common Control Information

Req 1	Each entity SHALL have the following common control information listed in Table 8-1.
http://www.opengis.net/spec/iot_sensing/1.0/req/core/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

1 The two terms of IoT and WoT are frequently used interchangeably.

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"`)

[Adapted from OData 4.0-JSON-Format section 18]

Table 8-1 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 URL that retrieves content of related entities.	Relative URL	One-to-many (mandatory)

8.3 The Sensing Profile Core Entities

The SensorThings API Sensing Profile Core Entities are depicted in Figure 1.

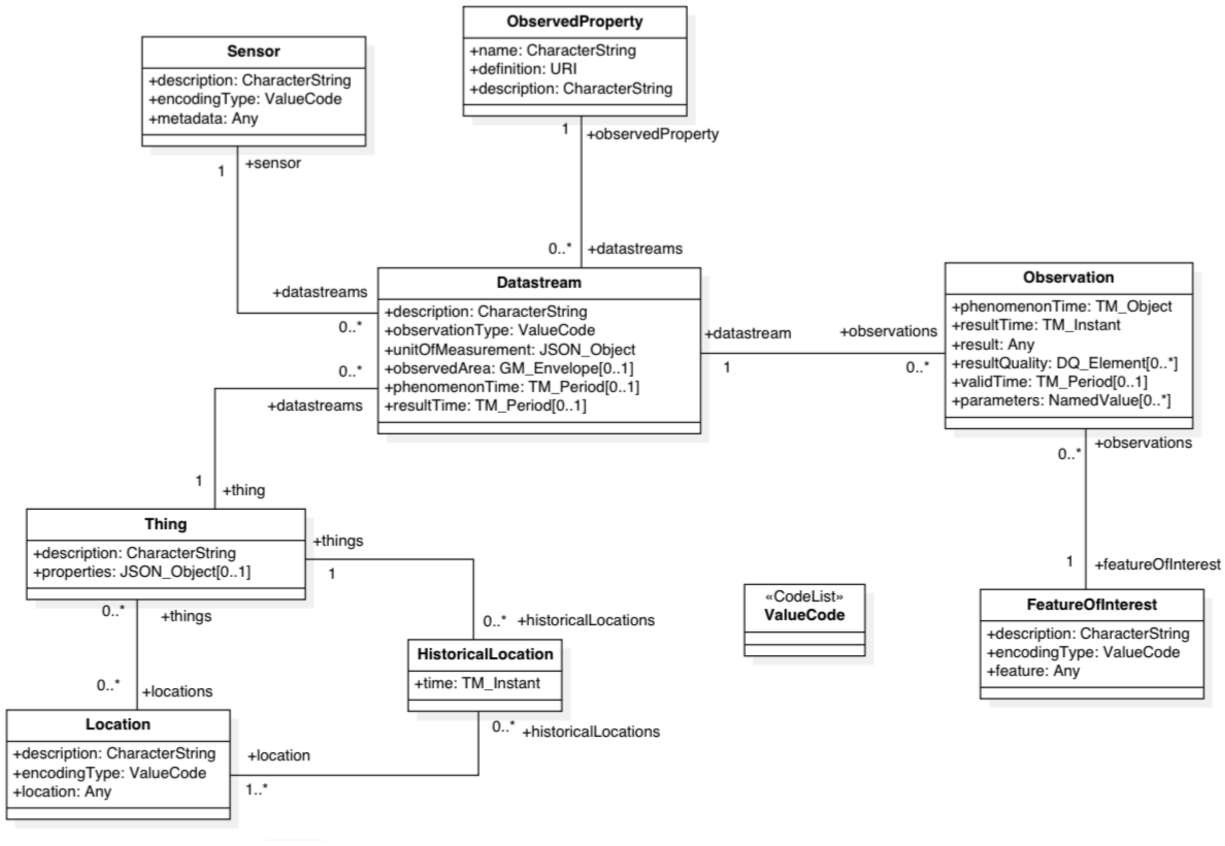

Figure 1 Sensing Profile Core Entities

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.

8.3.1 `Thing`

The OGC 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].

Req 2 Each `Thing` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8-2.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/thing-properties

Table 8-2 Properties of a `Thing` entity

Name	**Definition**	**Data type**	**Multiplicity and use**
`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

Req 3 Each `Thing` entity SHALL have the direct relation between a `Thing` entity and other entity types listed in Table 8-3.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/thing-relations

Table 8-3 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",
  "description": "This thing is an oven.",
  "properties": {
    "owner": "John Doe",
    "color": "Silver"
  }
}
```

8.3.2 `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 the `FeatureOfInterest` of the `Thing`'s `Observations`. Section 7.1.4 of [OGC and ISO 19156:2011] provides a detailed explanation of observation location.

Req 4 Each `Location` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8-4.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/location-properties

Table 8-4 Properties of a `Location` entity

Name	**Definition**	**Data type**	**Multiplicity and use**
`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 (see Table 8-6).	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)

Req 5 Each `Location` entity SHALL have the direct relation between a `Location` entity and other entity types listed in Table 8-5.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/location-relations

Table 8-5 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",
  "location": {
    "type": "Point",
    "coordinates": [-114.06,51.05]
  }
}
```

Table 8-6 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

8.3.3 `HistoricalLocation`

A `Thing`'s `HistoricalLocation` entity set provides the current (*i.e.*, last known) and previous locations of the `Thing` with their time.

Req 6 Each `HistoricalLocation` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8-7.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/historical-location-properties

Req 7 Each `HistoricalLocation` entity SHALL have the direct relation between a `Location` entity and other entity types listed in Table 8-8.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/historical-location-relations

Req 8 When a `Thing` has a new `Location`, a new `HistoricalLocation` SHALL be created and added to the `Thing` automatically by the service. The current `Location` of the `Thing` SHALL only be added to `HistoricalLocation` automatically by the service, and SHALL not be created as `HistoricalLocation` directly by user.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/historical-location-auto-creation

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 8-7 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 8-8 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": 1,
      "@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"
}
```

8.3.4 `Datastream`

A `Datastream` groups a collection of `Observations` and the `Observations` in a `Datastream` measure the same `ObservedProperty` and are produced by the same `Sensor`.

Req 9	Each `Datastream` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8-9. http://www.opengis.net/spec/iot_sensing/1.0/req/core/datastream-properties
Req 10	Each `Datastream` entity SHALL have the direct relation between a `Datastream` entity and other entity types listed in Table 8-10. http://www.opengis.net/spec/iot_sensing/1.0/req/core/datastream-relations

Table 8-9 Properties of a `Datastream` entity

Name	Definition	Data type	Multiplicity and use
`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 IRI 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 8-10.	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
`phenomenonTime`	The temporal bounding box of the phenomenon times of all observations belonging to this `Datastream`.	TM_Period (ISO 8601 Time Interval)	Zero-to-one
`resultTime`	The temporal bounding box of the result times of all observations belonging to this `Datastream`.	TM_Period (ISO 8601 Time Interval)	Zero-to-one

Table 8-10 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",
  "description": "This is a datastream measuring the 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 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 and ISO 19156:2011] to `observationType` names and observation `result` types.

Table 8-11 List of some code values used for identifying types defined in the O&M conceptual model (OGC and ISO 19156:2011)

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`	IRI
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

8.3.5 `Sensor`

A `Sensor` is an instrument that observes a property or phenomenon with the goal of producing an estimate of the value of the property[2].

Req 8 Each `Sensor` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8-12.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/sensor-properties

Req 9 Each `Sensor` entity SHALL have the direct relation between a `Sensor` entity and other entity types listed in Table 8-13.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/sensor-relations

Table 8-12 Properties of a `Sensor` entity

Name	Definition	Data type	Multiplicity and use
`description`	The description of the `Sensor` entity.	CharacterString	One (mandatory)

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

`encodingType`	The encoding type of the `metadata` property. Its value is one of the ValueCode enumeration (see Table 8-14 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 8-13 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 8-14 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`

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",
  "description": "TMP36 - Analog Temperature sensor",
  "encodingType": "application/pdf",
  "metadata": "http://example.org/TMP35_36_37.pdf"
}
```

8.3.6 `ObservedProperty`

An `ObservedProperty` specifies the phenomenon of an `Observation`.

Req 10 Each `ObservedProperty` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8-15.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/observed-property-properties

Req 11 Each `ObservedProperty` entity SHALL have the direct relation between a `ObservedProperty` entity and other entity types listed in Table 8-16.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/observed-property-relations

Table 8-15 Properties of an `ObservedProperty` entity

Name	Definition	Data type	Multiplicity and use
`name`	The name of the `ObservedProperty`.	CharacterString	One (mandatory)
`definition`	The IRI of the `ObservedProperty`. Dereferencing this IRI SHOULD result in a representation of the definition of the `ObservedProperty`.	IRI	One (mandatory)
`description`	A description about the `ObservedProperty`.	CharacterString	One (mandatory)

Table 8-16 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 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.",
  "name": "DewPoint Temperature",
  "definition": "http://dbpedia.org/page/Dew_point"
}
```

8.3.7 `Observation`

An `Observation` is act of measuring or otherwise determining the value of a property [OGC and ISO 19156:2011]

Req 12 Each `Observation` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8-17.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/observation-properties

Req 13 Each `Observation` entity SHALL have the direct relation between an `Observation` entity and other entity types listed in Table 8-18.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/observation-relations

Table 8-17 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 8-18 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
}
```

8.3.8 `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 19156:2001]. 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.

Req 14 Each `FeatureOfInterest` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8-19.   http://www.opengis.net/spec/iot_sensing/1.0/req/core/feature-of-interest-properties
Req 15 Each `FeatureOfInterest` entity SHALL have the direct relation between a `FeatureOfInterest` entity and other entity types listed in Table 8-20.   http://www.opengis.net/spec/iot_sensing/1.0/req/core/feature-of-interest-relations

Table 8-19 Properties of a `FeatureOfInterest` entity

Name	Definition	Data type	Multiplicity and use
`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 Table 8-6 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 8-20 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",
  "description": "This is a weather station.",
  "encodingType": "application/vnd.geo+json",
  "feature": {
    "type": "Point",
    "coordinates": [-114.06,51.05]
  }
}
```

9. SensorThings Service Interface

An OGC 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.

9.1 URI Components

The OGC 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 action 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.

9.2 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.

Req 16 An OGC SensorThings API service SHALL support all the resource path usages listed in Section 9.2. http://www.opengis.net/spec/iot_sensing/1.0/req/core/resource-path-to-entities

9.2.1 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": " http://example.org/v1.0/ObservedProperties"
    },
    {
      "name": "FeaturesOfInterest",
      "url": " http://example.org/v1.0/FeaturesOfInterest"
    }
  ]
}
```

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

9.2.3 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)`

9.2.4 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"
}
```

9.2.5 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
```

9.2.6 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.

9.2.7 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)"
    }
  ]
}
```

9.2.8 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`.

9.3 Requesting Data

Clients issue HTTP `GET` requests to OGC 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.

Req 17 OGC SensorThings API services are hypermedia driven services that return URLs to the client. If a client subsequently requests the advertised resource and the URL has expired, then the service SHOULD respond with `410 Gone`. If this is not feasible, the service SHALL respond with `404 Not Found`.

http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/status-code

9.3.1 Evaluating System Query Options

Req 18 An OGC SensorThings API service SHALL evaluate the system query options following the order specified in Section 9.3.1.

http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/order

The OGC 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`

9.3.2 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.

9.3.2.1 `$expand`

Req 19 The usage of the `$select` query option SHALL be as defined in Section 9.3.2.1.

http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/expand

The `$expand` system query option indicates the related entities to be represented inline. The value of the `$expand` query option must be a comma separated list of navigation property names. Additionally each navigation property can be followed by a forward slash and another navigation property to enable identifying a multi-level relationship.

Example 18: examples of `$expand` query option

Example 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 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 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 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.

9.3.2.2 `$select`

Req 20	The usage of the `$select` query option SHALL be as defined in Section 9.3.2.2.
http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/select	

The `$select` system query option requests that the service to return only the properties explicitly requested by the client. The value of a `$select` query option is a comma-separated list of selection clauses. Each selection clause may be a property name (including navigation property names). The service returns the specified content, if available, along with any available expanded navigation properties.

[Adapted from OData 4.0-Protocol 11.2.4.1]

Example 19: examples of `$select` query option

Example 1: `http://example.org/v1.0/Observations?$select=result,resultTime` returns only the `result` and `resultTime` properties for each `Observation` entity.

Example 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 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.

9.4 Query Entity Sets

The OGC SensorThings API services support requests for data via `HTTP GET` requests. Clients can apply query operators to further process the addressed resources. 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 OGC 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.

Req 21 If a service does not support a system query option, it SHALL fail any request that contains the unsupported option and SHOULD return `501 Not Implemented`.   http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/query-status-code

9.4.1 `$orderby`

Req 22 The usage of the `$orderby` query option SHALL be as defined in Section 9.4.1.   http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/orderBy

The `$orderby` system query option specifies the order in which items are returned from the service.

The value of the `$orderby` system query option contains a comma-separated list of expressions whose primitive result values are used to sort the items. A special case of such an expression is a property path terminating on a primitive property. A type cast using the qualified entity type name is required to order by a property defined on a derived type.

The expression can include the suffix `asc` for ascending or `desc` for descending, separated from the property name by one or more spaces. If `asc` or `desc` is not specified, the service orders by the specified property in ascending order.

Null values come before non-null values when sorting in ascending order and after non-null values when sorting in descending order.

Items are sorted by the result values of the first expression, and then items with the same value for the first expression are sorted by the result value of the second expression, and so on.

[Note: Adapted from OData 4.0-Protocol 11.2.5.2]

Example 20: examples of `$orderby` query option

Example 1: `http://example.org/v1.0/Observations?$orderby=result` returns all `Observations` ordered by the `result` property in ascending order.

Example 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.

9.4.2 `$top`

Req 23	The usage of the `$top` query option SHALL be as defined in Section 9.4.2.
	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/top

The `$top` system query option specifies a non-negative integer n that limits the number of items returned from a collection of entities. The service returns the number of available items up to but not greater than the specified value n.

If no unique ordering is imposed through an `$orderby` query option, the service imposes a stable ordering across requests that include `$top`.

[Note: Adapted from OData 4.0-Protocol 11.2.5.3]

In addition, if the `$top` value exceeds the service-driven pagination limitation (*i.e.*, the largest number of entities the service can return in a single response), the `$top` query option is discarded and the server-side pagination limitation is imposed.

Example 21: examples of `$top` query option

Example 1: `http://example.org/v1.0/Things?$top=5` returns only the first five entities in the `Things` collection.

Example 2: `http://example.org/v1.0/Observations?$top=5&$orderby=phenomenonTime desc` returns the first five `Observation` entries after sorted by the `phenomenonTime` property in descending order.

9.4.3 `$skip`

Req 24	The usage of the `$skip` query option SHALL be as defined in Section 9.4.3.
	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/skip

The `$skip` system query option specifies a non-negative integer n that excludes the first n items of the queried collection from the result. The service returns items starting at position n+1.

Example 22: examples of `$skip` query option

Example 1: `http://example.org/v1.0/Things?$skip=5` returns `Thing` entities starting with the sixth `Thing` entity in the `Things` collection.

Where `$top` and `$skip` are used together, `$skip` is applied before `$top`, regardless of the order in which they appear in the request.

If no unique ordering is imposed through an `$orderby` query option, the service imposes a stable ordering across requests that include `$skip`.

[Note: Adapted from OData 4.0-Protocol 11.2.5.4]

Example 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.

9.4.4 `$count`

Req 25	The usage of the `$count` query option SHALL be as defined in Section 9.4.4.
http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/count	

The `$count` system query option with a value of true specifies that the total count of items within a collection matching the request be returned along with the result.

A `$count` query option with a value of `false` (or not specified) hints that the service does not return a count.

The service returns an HTTP Status code of `400 Bad Request` if a value other than `true` or `false` is specified.

The `$count` system query option ignores any `$top`, `$skip`, or `$expand` query options, and returns the total count of results across all pages including only those results matching any specified `$filter`. Clients should be aware that the count returned inline may not exactly equal the actual number of items returned, due to latency between calculating the count and enumerating the last value or due to inexact calculations on the service.

[Adapted from OData 4.0-Protocol 11.2.5.5]

Example 23: examples of `$count` query option

Example 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": [
    {…},
    {…}
  ]
}
```

9.4.5 `$filter`

Req 26	The usage of the `$filter` query option SHALL be as defined in Section 9.4.5
http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/filter	

The `$filter` system query option allows clients to filter a collection of entities that are addressed by a request URL. The expression specified with `$filter` is evaluated for each entity in the collection, and only items where the expression evaluates to true are included in the response. Entities for which the expression evaluates to false or to null, or which reference properties that are unavailable due to permissions, are omitted from the response.

[Adapted from Data 4.0-URL Conventions 5.1.1]

The expression language that is used in `$filter` operators supports references to properties and literals. The literal values can be strings enclosed in single quotes, numbers and boolean values (true or false) or datetime values represented as ISO 8601 time string.

Example 24: examples of `$filter` query option

Example 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 2: `http://example.org/v1.0/Observations?$filter=Datastream/id eq '1'` returns all `Observations` whose `Datastream`'s `id` is 1.

Example 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 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.

9.4.5.1 Built-in filter operations

The OGC 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].

Req 27	The built-in filter operators SHALL be as defined in Table 9-1.
http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/built-in-filter-operations	

Table 9-1 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`

9.4.5.2 Built-in query functions

The OGC 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.

Req 28	The built-in query functions SHALL be as defined in Table 9-2.
http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/built-in-query-functions	

Table 9-2 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********')`

9.4.6 Server-Driven Paging (`nextLink`)

Req 29 An OGC SensorThings API service SHOULD support the server-driven paging mechanism listed in section 9.4.6.

http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/pagination

Responses that include only a partial set of the items identified by the request URL SHALL contain a link that allows retrieving the next partial set of items. This link is called a `nextLink`; its representation is format-specific. The final partial set of items SHALL NOT contain a `nextLink`.

The `nextLink` annotation indicates that a response is only a subset of the requested collection of entities or collection of entity references. It contains a URL that allows retrieving the next subset of the requested collection.

SensorThings clients SHALL treat the URL of the `nextLink` as opaque, and SHALL NOT append system query options to the URL of a next link. Services may not allow a change of format on requests for subsequent pages using the next link.

[Adapted from OData 4.0-Protocol 11.2.5.7]

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"
}
```

10. Sensing Profile CRUD

10.1 Overview

As many IoT devices are resource-constrained, the SensorThings API adopts the efficient REST web service style. That means the CRUD actions can be performed on the SensorThings entity types. The following subsection explains the CRUD protocol.

10.2 Create an entity

Req 30 To create an entity in a collection, the client SHALL send a HTTP `POST` request to that collection's URL. The `POST` body SHALL contain a single valid entity representation.

If the target URL for the collection is a `navigationLink`, the new entity is automatically linked to the entity containing the `navigationLink`.

Upon successful completion, the response SHALL contain a HTTP location header that contains the `selfLink` of the created entity.

Upon successful completion the service SHALL respond with `either 201 Created`, or `204 No Content`.

[Adapted from Data 4.0-Protocol, 11.4.2 Create an Entity]

In addition, the link between entities SHALL be established upon creating an entity. Two use cases SHALL be considered: (1) link to existing entities when creating an entity, and (2) create related entities when creating an entity. The requests for these two use cases are described in the following subsection.

When clients create resources in a SensorThings service, they SHALL follow the integrity constraints listed in Table 10-1. For example, a `Datastream` entity shall link to a `Thing` entity. When a client wants to create a `Datastream` entity, the client needs to either (1) create a linked `Thing` entity in the same request or (2) link to an already created `Thing` entity. The complete integrity constraints for creating resources are shown in the following table.

Special case #1 - When creating an `Observation` entity that links to a `FeatureOfInterest` entity: Sometimes the `FeatureOfInterest` of an `Observation` is the `Location` of the `Thing`. For example, a wifi-connected thermostat's temperature observation's feature-of-interest can be the location of the smart thermostat, that is the room where the smart thermostat is located in.

In this case, when a client creates an `Observation` entity, the client SHOULD omit the link to a `FeatureOfInterest` entity in the `POST` body message and SHOULD not create a related `FeatureOfInterest` entity with deep insert. And if the service detects that there is no link to a `FeatureOfInterest` entity in the `POST` body message that creates an `Observation` entity, the service SHALL either (1) create a `FeatureOfInterest` entity by using the `location` property from the `Location` of the `Thing` entity when there is no `FeatureOfInterest` whose location property is from the `Location` of the `Thing` entity or (2) link to the `FeatureOfInterest` whose location property is from the `Location` of the `Thing` entity.

Special case #2: In the context of IoT, many `Observations' resultTime` and `phenomenonTime` cannot be distinguished or the `resultTime` is not available. In this case, when a client creates an `Observation` entity, the client MAY omit the `resultTime` and the service SHOULD assign a `null` value to the `resultTime`.

http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/create-entity

Table 10-1 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**	-

10.2.1 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: create a `Thing` entity

```
POST /Things HTTP/1.1

Host: example.org/v1.0
Content-Type: application/json

{
  "description":"This is a smart thermostat."
}
```

10.2.1.1 Link to existing entities when creating an entity

Req 31 A SensorThings API service, that supports entity creation, SHALL support linking new entities to existing entities upon creation. To create a new entity with links to existing entities in a single request, the client SHALL include the unique identifiers of the related entities associated with the corresponding navigation properties in the request body.

In the case of creating an `Observation` whose `FeatureOfInterest` is the `Thing`'s `Location` (that means the `Thing` entity has a related `Location` entity), the request of creating the `Observation` SHOULD NOT include a link to a `FeatureOfInterest` entity. The service will first automatically create a `FeatureOfInterest` entity from the `Location` of the `Thing` and then link to the `Observation`.

In the complex use case of a `Thing` has multiple `Location` representations, the service SHOULD decide the default `Location` encoding when an `Observation`'s `FeatureOfInterest` is the `Thing`'s `Location`.

http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/link-to-existing-entities

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

{
  "Datastream": {
    "@iot.id": 1
  },
  "phenomenonTime": "2013-04-18T16:15:00-07:00",
  "result": 124,
  "FeatureOfInterest": {
    "@iot.id": 2
  }
}
```

10.2.1.2 Create related entities when creating an entity

Req 32 A request to create an entity that includes related entities, represented using the appropriate inline representation, is referred to as a "deep insert". A SensorThings service that supports entity creation SHALL support deep insert.

If the inline representation contains a value for a computed property (*i.e.*, `id`), the service SHALL ignore that value when creating the related entity.

On success, the service SHALL create all entities and relate them. On failure, the service SHALL NOT create any of the entities.

[Adapted from Data 4.0-Protocol 11.4.2.2]

http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/deep-insert

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 /Things HTTP1.1
Host: example.org/v1.0
Content-Type: application/json

{
  "description": "This a Thing with one Datastream.",
     "Locations": [
       {
         "encodingType": "application/vnd.geo+json",
         "location": {
       "type": "POINT",
       "coordinates": [10,10]
      }
    }
  ],
  "Datastreams": [
    {
      "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": {
            "description": "This is CCIT #361, Steve’s office",
            "encodingType": "application/vnd.geo+json",
            "feature": {
              "type": "POLYGON",
              "coordinates": [
                [[100,50],[10,9],[23,4],[100,50]],[[30,20],[10,4],[4,22],[30,20]]
              ]
            }
          }
        }
      ],
      "ObservedProperty": {
        "name": "DewPoint Temperature",
        "definition": "http://sweet.jpl.nasa.gov/ontology/property.owl#DewPointTemperature",
        "description": "The dewpoint temperature is the temperature to which the air must be
cooled, at constant pressure, for dew to form. As the grass and other objects near the ground
cool to the dewpoint, some of the water vapor in the atmosphere condenses into liquid water on
the objects."
         },
         "Sensor": {
           "encodingType": "application/pdf",
       "metadata": "http://datasheets.maxim-ic.com/en/ds/DS18B20.pdf"
      }
    }
  ]
}
```

10.2.2 Response

Req 33 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.

http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/deep-insert-status-code

10.3 Read entities

Req 34 A SensorThings service SHALL support reading resources as defined in Section 10.3.

http://www.opengis.net/spec/iot_sensing/1.0/req/core/read-entity

10.3.1 Request

HTTP Method: `GET`

URI Pattern: Refer to the SensorThings service interface section (*i.e.*, section 9), including resource path and query options

10.3.2 Response

The detail explanation about the encodings of resources in the Sensing Profile can be found in section 8.3.

Upon successfully retrieve resources, the service responds with `200 OK`. Regarding all the HTTP status code, please refer to the HTTP Status Code section.

10.4 Update an entity

Req 35 To update an entity in a collection a SensorThings service SHALL follow the requirements as defined in Section 10.4.

http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/update-entity

10.4.1 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], is 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.

[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 Things(1) HTTP1.1

Host: example.org/v1.0/
Content-Type: application/json

{
  "description":"This thing is an oven."
}
```

10.4.2 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.

10.5 Delete an entity

Req36 To delete an entity in a collection a SensorThings service SHALL follow the requirements as defined in section 10.5.

http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/delete-entity

10.5.1 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 10-2 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 10-2 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.**	-

11. Batch Requests

Req 37	The batch-processing of the SensorThings service SHALL be as defined in Section 11.
http://www.opengis.net/spec/iot_sensing/1.0/req/batch-request	

11.1 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.

11.2 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:

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

11.2.1 Batch request body example

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

1. A query request
2. Change Set that contains the following requests:
    a. Insert entity (with `Content-ID` = 1)
    b. Update request (with `Content-ID` = 2)
3. 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--
```

11.2.2 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:

- Insert a new Datastream entity (with `Content-ID = 1`)
- 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 $1/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--
```

11.3 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--
```

11.4 Asynchronous batch requests

Example 31-5: referencing the example 31-2 above again, assume that when interrogating the monitor URL for the first tim 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–
```

12. 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 and ISO 19156:2008]. 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.

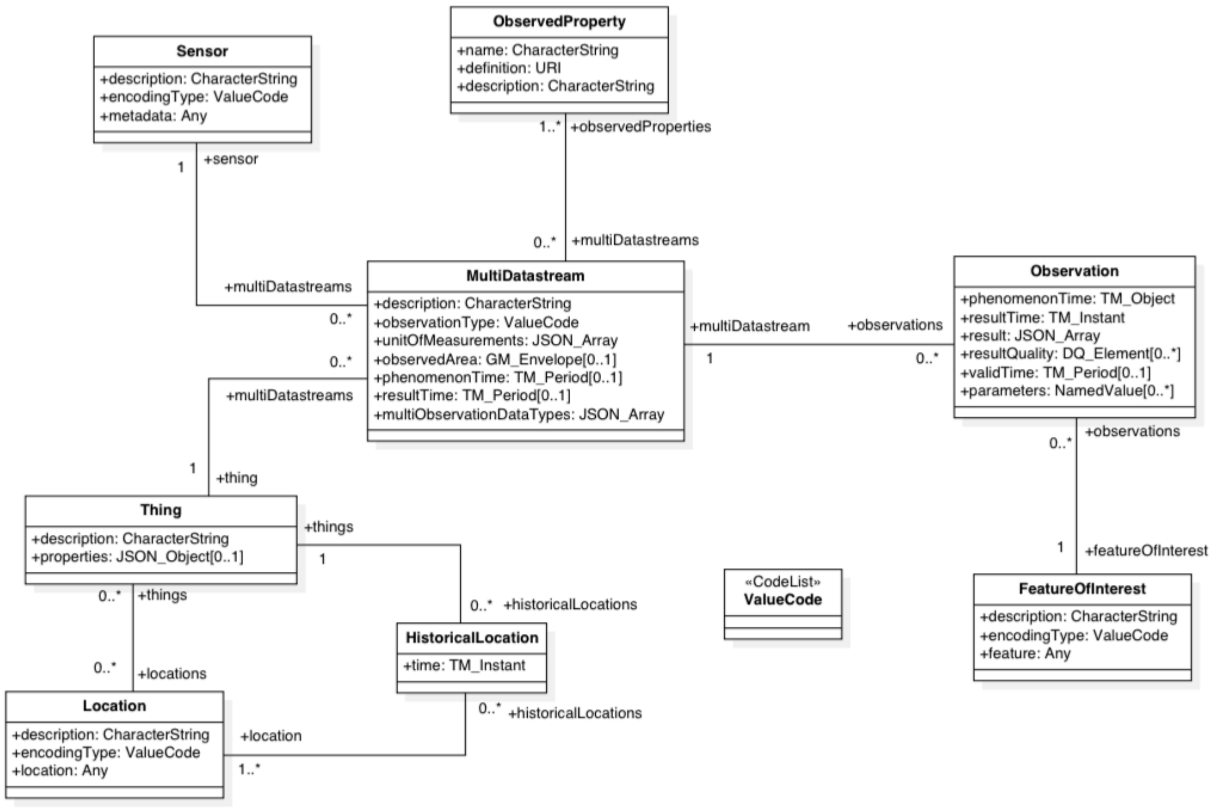
Sensor
+description: CharacterString
+encodingType: ValueCode
+metadata: Any
ObservedProperty
+name: CharacterString
+definition: URI
+description: CharacterString
MultiDatastream
+description: CharacterString
+observationType: ValueCode
+unitOfMeasurements: JSON_Array
+observedArea: GM_Envelope[0..1]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

Figure 2 MultiDatastream Extension Entities

Req 38 Each `MultiDatastream` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 12-1. http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/properties
Req 39 Each `MultiDatastream` entity SHALL have the direct relation between a `Datastream` entity and other entity types listed in Table 12-2. http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/relations

Table 12-1 Properties of a `MultiDatastream` entity

Name	Definition	Data type	Multiplicity and use
`description`	The description of the `Datastream` entity.	CharacterString	One (mandatory)

unitOfMeaseurements	A JSON array of JSON objects that containing three key-value pairs. The name property presents the full name of the unitOfMeasurement; the symbol property shows the textual form of the unit symbol; and the definition contains the IRI defining the unitOfMeasurement. (see Req 40 for the constraints between unitOfMeasurement, multiObservationDataType and result)	A JSON array	One (mandatory) Note: It is possible an observation does not have a unit of measurement. For example, a count observation does not have a unit of measurement.
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 8-11 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 bounding box of the phenomenon times of all observations belonging to this MultiDatastream.	TM_Period (ISO 8601 Time Interval)	Zero-to-one
resultTime	The temporal bounding box of the result times of all observations belonging to this MultiDatastream.	TM_Period (ISO 8601 Time Interval)	Zero-to-one

Table 12-2 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 12-3 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.

Req 40 The size and the order of each element of a MultiDatastream's unitOfMeasurements array (*i.e.*, MultiDatastream(id)/unitOfMeasurements) SHALL match the size and the order of each element of the related ObservedProperties collection (*i.e.*, MultiDatastreams(id)/ObservedProperties).

The size and the order of each element of a MultiDatastream's unitOfMeasurements array (*i.e.*, MultiDatastreams(id)/unitOfMeasurements) SHALL match the size and the order of each element of all related Observations' result (*i.e.*, MultiDatastreams(id)/Observations?$select=result).

The size and the order of each element of a MultiDatastream's unitOfMeasurements array (*i.e.*, MultiDatastreams(id)/unitOfMeasurements) SHALL match the size and the order of each element of the MultiDatastream's multiObservationDataTypes array (*i.e.*, MultiDatastreams(id)/multiObservationDataTypes).

When a complex result's element does not have a unit of measurement (e.g., a OM_TruthObservation type), the corresponding unitOfMeasurement element SHALL have null values.

http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/constraints

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",
   "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"
  ]
}
```

13. SensorThings Data Array Extension

Req 41	To support the SensorThings data array extension, a service SHALL support the retrieval and creation of observations as defined in Section 13. http://www.opengis.net/spec/iot_sensing/1.0/req/data-array/data-array

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`.

13.1 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`.

13.1.1 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`.

13.1.2 Response

The response `Observations` in `dataArray` format contains the following properties.

Table 13-1 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 /Datastreams(1)/Observations?$resultFormat=dataArray
HTTP/1.1 200 OK
Host: www.example.org/v1.0
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"
          ]
        ]
      ]
    }
  ]
}
```

13.2 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`.

13.2.1 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 13-2 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 /CreateObservations HTTP/1.1
Host: example.org/v1.0
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
      ]
    ]
  }
]
```

13.2.2 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)"
]
```

14. SensorThings Sensing Profile 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.

14.1 Create a SensorThings entity with MQTT Publish

Req 42 To allow clients to create entities with `MQTT Publish`, a service SHALL support the creation of entities with MQTT as defined in Section 14.1.

http://www.opengis.net/spec/iot_sensing/1.0/req/mqtt/create

To create an entity in a collection with MQTT, the client sends a `MQTT Publish` request to the SensorThings service and the MQTT topic is the collection's resource path. The MQTT application message contains a single valid entity representation.

If the MQTT topic for the collection is a `navigationLink`, the new entity is automatically linked to the entity containing the `navigationLink`.

Similar to creating entities with `HTTP POST`, creating entities with `MQTT Publish` follow the integrity constraints listed in Table 10-1. The two special cases defined in Req 30 are also applied in the case of creating entities with `MQTT Publish`.

14.1.1 Link to existing entities when creating an entity

To link to existing entities when creating an entity with MQTT, the conditions in Req 31 is applied.

14.1.2 Create related entities when creating an entity (deep insert)

To create related entities when creating an entity with MQTT, the condition in Req 32 is applied.

14.2 Update a SensorThings entity with MQTT Publish

Req 43 To allow clients to update SensorThings entities with `MQTT Publish`, a service SHALL support the updates of entities with MQTT as defined in Section 14.2.

http://www.opengis.net/spec/iot_sensing/1.0/req/mqtt/update

To update a SensorThings entity with `MQTT Publish`, the client sends a `MQTT Publish` request to the SensorThings service and the MQTT topic is the resource path addressing to the single entity.

The properties provided in the payload (*i.e.*, MQTT application message) corresponding to updatable properties SHALL replace the value of the corresponding property in the entity. Missing properties of the containing entity of complex property SHALL not be directly altered.

Key (*i.e.*, `id`) and other non-updatable properties (e.g., `navigationLink`) 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. For example 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 relation.

14.3 Receive updates with MQTT Subscribe

Req 44 To allow clients to receive notifications for the updates of SensorThings entities with MQTT, a service SHALL support the receiving updates with `MQTT Subscribe` as defined in Section 14.3.

http://www.opengis.net/spec/iot_sensing/1.0/req/mqtt/receive-update

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.

14.3.1 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.

14.3.2 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.

14.3.3 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"
}
```

14.3.4 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"
}
```

Annex A

(normative)

Abstract Test Suite

NOTE: The smaller blue text in the following tables is the path fragment that appended to the following URI: http://www.opengis.net/spec/iot_sensing/1.0/, and it provides the URI that can be used to unambiguously identify the requirement and the conformance class.

A.1 Conformance class: SensorThings API Sensing Core

This section describes conformance test for the SensorThings API Sensing Core.

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/core

A.1.1 Test: Common Control Information

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/core/common-control-information
Test purpose	Check if each entity has the common control information as defined in the requirement http://www.opengis.net/spec/iot_sensing/1.0/req/core/common-control-information.
Test method	Inspect the full JSON object of the entity sets (*i.e.*, without `$select`) to identify, if each entity has the common control information required in requirement http://www.opengis.net/spec/iot_sensing/1.0/req/core/common-control-information and the service sends appropriate responses as defined in this specification.

A.1.2 Test: Entity Properties

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/core/common-control-information • http://www.opengis.net/spec/iot_sensing/1.0/req/core/thing-properties • http://www.opengis.net/spec/iot_sensing/1.0/req/core/location-properties • http://www.opengis.net/spec/iot_sensing/1.0/req/core/historical-location-properties • http://www.opengis.net/spec/iot_sensing/1.0/req/core/datastream-properties • http://www.opengis.net/spec/iot_sensing/1.0/req/core/sensor-properties • http://www.opengis.net/spec/iot_sensing/1.0/req/core/observed-property-properties • http://www.opengis.net/spec/iot_sensing/1.0/req/core/observation-properties • http://www.opengis.net/spec/iot_sensing/1.0/req/core/feature-of-interest-properties
Test purpose	Check if each entity has the mandatory properties as defined in this specification.
Test method	Inspect the full JSON object of the different entity sets (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties defined in the corresponding requirements.

A.1.2 Test: Entity Relations

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/core/thing-relations • http://www.opengis.net/spec/iot_sensing/1.0/req/core/location-relations • http://www.opengis.net/spec/iot_sensing/1.0/req/core/historical-location-relations • http://www.opengis.net/spec/iot_sensing/1.0/req/core/datastream-relations • http://www.opengis.net/spec/iot_sensing/1.0/req/core/sensor-relations • http://www.opengis.net/spec/iot_sensing/1.0/req/core/observed-property-relations • http://www.opengis.net/spec/iot_sensing/1.0/req/core/observation-relations • http://www.opengis.net/spec/iot_sensing/1.0/req/core/feature-of-interest-relations
Test purpose	Check if each entity has the mandatory relations as defined in the above listed requirements.
Test method	Inspect the full JSON object of each SensorThings entity set (*i.e.*, without using the `$select` query option) to identify, if each entity has the mandatory relations (*i.e.*, `@iot.navigationLink`) defined in the corresponding requirements.

A.1.3 Test: Resource Path

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/core/resource-path-to-entities
Test purpose	Check if the service supports all the resource path usages as defined in the requirement http://www.opengis.net/spec/iot_sensing/1.0/req/core/resource-path-to-entities.
Test method	Inspect the service to identify, if each resource path usage has been implemented property.

A.1.4 Test: Request Data

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/core/read-entity
Test purpose	Check if the service supports the data request usage as defined in the requirement http://www.opengis.net/spec/iot_sensing/1.0/req/core/read-entity.
Test method	Issue a `HTTP GET` request to the service with an appropriate resource path usage (e.g., service root) to inspect, if the service supports `HTTP GET`. Request a resource that exists to identify, if the response code is `200 OK`. Request a resource that doesn't exist to identify, if the response code is `404 Not Found`.

A.2 Conformance class: SensorThings API Filtering Extension

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/request-data

Dependency: http://www.opengis.net/spec/iot_sensing/1.0/conf/core

A.2.1 Test: Query Option Order

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/order
Test purpose	Check if the results of the service requests are as if the system query options were evaluated in the order as defined in this specification.
Test method	Send a query includes the query options listed in requirement http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/order, and check if the results are evaluated according to the order defined in this specification.

A.2.2 Test: Request Data with `$expand` and `$select`

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/expand • http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/select
Test purpose	Check if the service supports `$expand` and `$select` as defined in this specification.
Test method	Send requests with `$expand` following the different usages as defined in the requirement http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/expand, check if the server returns appropriate result as defined in this specification. Send requests with the `$select` option following the different usages as defined in the requirement http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/select, check if the server returns appropriate result as defined in this specification.

A.2.3 Test: Query Option Response Code

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/query-status-code
Test purpose	Check when a client use a query option that doesn't support by the service, if the service fails the request and responds with `501 Not Implemented` as defined in the requirement

	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/query-status-code.
Test method	(If applicable) Send a query with a query option that is not supported by the service, check if the server returns `501 Not Implemented`.

A.2.4 Test: Sorting Query Option

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/orderby
Test purpose	Check if the service supports the `$orderby` query option as defined in this specification.
Test method	Send a query with the `$orderby` query option, check if the server returns appropriate result as defined in this specification.

A.2.5 Test: Client-driven Pagination Query Option

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/top • http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/skip • http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/count
Test purpose	Check if the service supports the `$top`, `$skip` and `$count` query option as defined in this specification.
Test method	Send a query with the `$top` query option, check if the server returns appropriate result as defined in this specification. Send a query with the `$skip` query option, check if the server returns appropriate result as defined in this specification. Send a query with the `$count` query option, check if the server returns appropriate result as defined in this specification.

A.2.6 Test: Filter Query Option

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/filter • http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/built-in-filter-operations • http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/built-in-filter-functions
Test purpose	Check if the service supports the `$filter` query option and the built-in filter operators and

	built-in filter functions as defined in this specification.
Test method	Send a query with the `$filter` query option, check if the server returns appropriate result as defined in this specification. Send a query with the `$filter` query option for each built-in filter operator, check if the server returns appropriate result as defined in this specification. Send a query with the `$filter` query option for each built-in filter function, check if the server returns appropriate result as defined in this specification.

A.2.7 Test: Server-driven Pagination

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/pagination
Test purpose	Check if the service supports the server-driven pagination as defined in the requirement http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/pagination.
Test method	Send a query to list all entities of an entity set, check if the server returns a subset of the requested entities as defined in this specification.

A.3 Conformance class: SensorThings API Create-Update-Delete

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/create-update-delete

Dependency: http://www.opengis.net/spec/iot_sensing/1.0/conf/core

A.3.1 Test: Sensing Entity Creation

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/create-entity • http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/link-to-existing-entities • http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/deep-insert • http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/deep-insert-status-code
Test purpose	Check if the service supports the creation of entities as defined in this specification.
Test method	For each SensorThings entity type creates an entity instance by following the integrity constraints of Table 10-1 and creating the related entities with a single request (*i.e.*, deep insert), check if the entity instance is successfully created and the server responds as defined

	in this specification. Create an entity instance and its related entities with a deep insert request that does not conform to the specification (e.g., missing a mandatory property), check if the service fails the request without creating any entity within the deep insert request and responds the appropriate HTTP status code. For each SensorThings entity type issue an entity creation request that does not follow the integrity constraints of Table 10-1 with deep insert, check if the service fails the request without creating any entity within the deep insert request and responds the appropriate HTTP status code. For each SensorThings entity type creates an entity instance by linking to existing entities with a single request, check if the server responds as defined in this specification. For each SensorThings entity type creates an entity instance that does not follow the integrity constraints of Table 10-1 by linking to existing entities with a single request, check if the server responds as defined in this specification. Create an `Observation` entity for a `Datastream` without any `Observations` and the `Observation` creation request does not create a new or linking to an existing `FeatureOfInterest`, check if the service creates a new `FeatureOfInterest` for the created `Observation` with the `location` property of the `Thing`'s `Location` entity. Create an `Observation` entity for a `Datastream` that already has `Observations` and the `Observation` creation request does not create a new or linking to an existing `FeatureOfInterest`, check if the service automatically links the newly created `Observation` with an existing `FeatureOfInterest` whose `location` property is from the `Thing`'s `Location` entity. Create an `Observation` entity and the `Observation` creation request does not include `resultTime`, check if the `resultTime` property is created with a `null` value.

A.3.2 Test: Sensing Entity Update

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/entity
Test purpose	Check if the service supports the update of entities as defined in this specification.
Test method	For each SensorThings entity type send an update request with `PATCH`, check (1) if the properties provided in the payload corresponding to updatable properties replace the value of the corresponding property in the entity and (2) if the missing properties of the containing entity or complex property are not directly altered. (Where applicable) For each SensorThings entity type send an update request with `PUT`,

	check if the service responds as defined in Section 10.4. For each SensorThings entity type send an update request with `PATCH` that contains related entities as inline content, check if the service fails the request and returns appropriate HTTP status code. For each SensorThings entity type send an update request with `PATCH` that contains binding information for navigation properties, check if the service updates the `naviationLink` accordingly.

A.3.3 Test: Sensing Entity Deletion

Requirements	• req/sensingDelete/entity
Test purpose	Check if the service supports the deletion of entities as defined in Section 10.5.
Test method	Delete an entity instance, and check if the service responds as defined in Section 10.5.

A.4 Conformance class: SensorThings API Batch Request

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/batch-request

Dependency: http://www.opengis.net/spec/iot_sensing/1.0/conf/core

A.4.1 Test: Batch Request

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/conf/batch-request/batch-request
Test purpose	Check if the service supports the batch request as defined in Section 11.
Test method	Submit batch requests according to the examples listed in Section 11, check if the service responds as defined in this specification.

A.5 Conformance class: SensorThings API Sensing MultiDatastream Extension

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/multi-datastream

Dependency: http://www.opengis.net/spec/iot_sensing/1.0/conf/core

A.5.1 Test: SensorThings API Sensing MultiDatastream Extension

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/conf/multi-datastream/properties • http://www.opengis.net/spec/iot_sensing/1.0/conf/multi-datastream/relations • http://www.opengis.net/spec/iot_sensing/1.0/conf/multi-datastream/constraints
Test purpose	Check if the service's `MultiDatastream` entity has the mandatory properties and relations as defined in this specification.
Test method	Inspect the full JSON object of a `MultiDatastream` entity (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties and relations, and fulfill the constraints defined in the corresponding requirements.

A.6 Conformance class: SensorThings API Sensing Data Array Extension

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/data-array

Dependency: http://www.opengis.net/spec/iot_sensing/1.0/conf/core

A.6.1 Test: SensorThings API Sensing Data Array Extension

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/data-array/data-array
Test purpose	Check if the service supports the data array extension as defined in Section 13.
Test method	Issue a `GET` request for `Datastreams` (and `MultiDatastreams` if applicable) that includes the query option `"$resultFormat=dataArray"`, and then inspect the returned JSON to identify if it fulfills the data array format as defined in Section 13. Create at least two `Datastreams` by using the data array format as defined in Section 13. Inspect the response code and returned JSON to identify if it fulfills the response as defined in Section 13.

A.7 Conformance class: SensorThings API MQTT Extension for Create and Update

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/mqtt

Dependency:

- http://www.opengis.net/spec/iot_sensing/1.0/conf/core
- http://www.opengis.net/spec/iot_sensing/1.0/conf/create-update-delete

A.7.1 Test: SensorThings API MQTT Extension for Create and Update

Requirements	• req/mqtt/create • req/mqtt/update

Test purpose	Check if the service supports the creation and update of entities via MQTT as defined in Section 14.1 and 14.2.
Test method	For each SensorThings entity type creates an entity instance containing binding information for navigation properties using `MQTT Publish`, check if the server responds as defined in Section 14.1. For each SensorThings entity type send an update request with `MQTT`, check (1) if the properties provided in the payload corresponding to updatable properties replace the value of the corresponding property in the entity and (2) if the missing properties of the containing entity or complex property are not directly altered. For each SensorThings entity type send an update request with `MQTT` that contains binding information for navigation properties, check if the service updates the `naviationLink` accordingly.

A.8 Conformance class: SensorThings API MQTT Extension for Receiving Updates

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/mqtt

Dependency:

- http://www.opengis.net/spec/iot_sensing/1.0/conf/core
- http://www.opengis.net/spec/iot_sensing/1.0/conf/create-update-delete

A.8.1 Test: Sensing Profile MQTT Extension for Receiving Updates

Requirements	• http://www.opengis.net/spec/iot_sensing/1.0/req/mqtt/receive-updates
Test purpose	Check if a client can receive notifications for the updates of a SensorThings entity set or an individual entity with MQTT.
Test method	Subscribe to an entity set with `MQTT Subscribe`. Then create a new entity of the subscribed entity set either with `POST` or `MQTT Publish`. Check if a complete JSON representation of the newly created entity through MQTT is received. Subscribe to an entity set with `MQTT Subscribe`. Then update an existing entity of the subscribed entity set either with `POST` or `MQTT Publish`. Check if a complete JSON representation of the updated entity through MQTT is received. Subscribe to an entity's property with `MQTT Subscribe`. Then update the property either with `PATCH` or `MQTT Publish`. Check if the JSON object of the updated property is received. Subscribe to multiple properties of an entity set with `MQTT Subscribe`. Then create a new

	entity of the entity set either with `POST` or `MQTT Publish`. Check if a JSON object of the subscribed properties is received. Subscribe to multiple properties of an entity set with `MQTT Subscribe`. Then update an existing entity of the entity set either with `PATCH` or `MQTT Publish`. Check if a JSON object of the subscribed properties is received.

Bibliography

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

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

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

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