Open Geospatial Consortium

Submission Date: 2015-10-26

Approval Date: 2016-02-19

Publication Date: 2016-08-04

External identifier of this OGC® document: http://www.opengis.net/doc/is/sensorthings/1.0

The Normative version of this document is at: http://docs.opengeospatial.org/is/15-078r6/15-078r6.html

Internal reference number of this OGC® document: 15-078r6

Version: 1.0

Category: OGC® Implementation Standard

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

OGC SensorThings API

Part 1: Sensing

Document type:	OGC® Standard
Document subtype:	Implementation Standard
Document stage:	Approved
Document language:	English

Contents

Figures

Tables

i. Abstract

The OGC SensorThings API provides an open, geospatial-enabled and unified way to interconnect the 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 part. The two parts are the Sensing part and the Tasking part. The Sensing part provides a standard way to manage and retrieve observations and metadata from heterogeneous IoT sensor systems. The Tasking part is planned as a future work activity and will be defined in a separate document as the Part II of the SensorThings API.

ii. Keywords

The following are keywords to be used by search engines and document catalogues.

ogcdoc, OGC document, iot, internet of things, sensor things, sensors, swe, sensor webs, sensor web enablement, sensor networks

iii. Preface

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

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

The Sensing part provides functions similar to the OGC Sensor Observation Service (SOS) and the Tasking part 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 the resource-constrained IoT devices and the Web developer community. As a result, the SensorThings API follows the REST principles, the use of an efficient JSON encoding, the use of MQTT protocol, the use of the flexible OASIS OData protocol and URL conventions.

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.

iv. Submitting organizations

The following organizations submitted this Document to the Open Geospatial Consortium (OGC):

- University of Calgary, Canada
- National Central University, Taiwan
- Lockheed Martin, USA
- AIST, Japan
- FCU.GIS, Taiwan
- ITRI, Taiwan
- GeoConnections, Canada
- Noblis, USA
- Fraunhofer, Germany

v. Submitters

All questions regarding this submission should be directed to the editor or the submitters:

Name	Affiliation
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
Thomas Schwab	Lockheed Martin
Jean Brodeur	NRCan/Canadian Geospatial Data Infrastructure

Marcus Alzona	Noblis

1. Scope

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

2. Conformance

Conformance with this standard shall be checked using all the relevant tests specified in Annex A (normative) of this document. The framework, concepts, and methodology for testing, and the criteria to be achieved to claim conformance are specified in the OGC Compliance Testing Policies and Procedures and the OGC Compliance Testing web site1.

All requirements-classes and conformance-classes described in this document are owned by the standard(s) identified.

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

NOTE: The smaller blue text in the following table 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/thing	• req/thing/properties • req/thing/relations	Thing entity
req/location	• req/location/properties • req/location/relations	Location entity
req/historical-location	• req/historical-location/properties • req//historical-location/relations	HistoricalLocation entity
req/datastream	• req/datastream/properties • req/datastream/relations	Datastream entity
req/sensor	• req/sensor/properties • req/sensor/relations	Sensor Entity

1 www.opengeospatial.org/cite

req/observed-property	• req/observed-property/properties • req/observed-property/relations	ObservedProperty entity
req/observation	• req/observation/properties • req/observation/relations	Observation entity
req/feature-of-interest	• req/feature-of-interest/properties • req/feature-of-interest/relations	FeatureOfInterest entity
req/entity-control-information	• req/entity-control-information/common-control-information	Entities' common control information
req/resource-path	• req/resource-path/resource-path-to-entities	Addressing to the entities of the SensorThings API service
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/create-update-delete/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
req/create-observations-via-mqtt	• req/create-observations-via-mqtt/observations-creation	creating observations through MQTT
req/receive-updates-via-mqtt	• req/receive-updates-via-mqtt/receive-updates	Receiving updates through MQTT

3. References

The following normative documents contain provisions that, 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 8601:2004 Data elements and interchange formats – Information interchange - Representation of dates and times.
OGC 10-004r3 and ISO 19156:2011(E), OGC Abstract Specification Topic 20: Geographic information — Observations and Measurements
OASIS OData Version 4.0 Part 1: Protocol Plus Errata 02
OASIS OData Version 4.0 Part 2: URL Conventions Plus Errata 02

OASIS OData JSON Format Version 4.0 Plus Errata 02
OASIS OData ABNF Construction Rules Errata 02
RFC 2046, Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types
RFC 2616, Hypertext Transfer Protocol -- HTTP/1.1
RFC 4627, 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

This document uses the terms defined in Sub-clause 5.3 of [OGC 06-121r8], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word "shall" (not "must") is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

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

4.1
Collection
Sets of Resources, which can be retrieved in whole or in part. [RFC5023]

4.2
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.

4.3
Entity sets
Entity sets are named collections of entities (e.g. Sensors is an entity set containing Sensor entities). An entity's key uniquely identifies the entity within an entity set. Entity sets provide entry points into an OGC SensorThings API service. [OASIS OData Version 4.0 Part 1: Protocol Plus Errata 02]

4.4
(Internet of) Thing
A thing is an object of the physical world (physical things) or the information world (virtual things) that is capable of being identified and integrated into communication networks. [ITU-T Y.2060]

4.5
Measurement
A set of operations having the object of determining the value of a quantity [OGC 10-004r3 / ISO 19156:2011]

4.6
Observation
Act of measuring or otherwise determining the value of a property [OGC 10-004r3 / ISO 19156:2011]

4.7
Observation Result
Estimate of the value of a property determined through a known observation procedure [OGC 10-004r3 / ISO 19156:2011]

4.8
Resource
A network-accessible data object or service identified by an URI, as defined in [RFC 2616]

4.9
REST
The Representational State Transfer (REST) style is an abstraction of the architectural elements within a distributed hypermedia system. REST focuses on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. It encompasses the fundamental constraints upon components, connectors, and data that define the basis of the Web architecture, and thus the essence of its behavior as a network-based application. An API that conforms to the REST architectural principles/constraints is called a RESTful API.

4.10
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

This sections provides details and examples for any conventions used in the document. Examples of conventions are symbols, abbreviations, use of XML schema, or special notes regarding how to read the document.

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.

5.2 Identifiers

The normative provisions in this specification are denoted by the URI

http://www.opengis.net/spec/iot_sensing/1.0/

All requirements and conformance tests that appear in this document are denoted by partial URIs which are relative to this base.

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 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[2], 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 the connections between devices-to-devices and devices-to-applications.

7.3 SensorThings API Overview

The OGC SensorThings API data model consists of two parts: (1) the Sensing part and (2) the Tasking part. The Sensing part 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.

The Sensing part is designed based on the ISO/OGC Observation and Measurement (O&M) model [OGC 10-004r3 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, and its definition follows the ITU-T definition: "*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].

The geographical `Locations` of `Things` are useful in almost every application and as a result are included as well. For the `Things` whose location changed, the `HistoricalLocations` entities offer the history of the `Thing`'s locations. 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`. Details of each above described entity are provided in Section 8.

7.4 SensorThings API and ISO/OGC Observations and Measurements

Managing and retrieving observations and metadata from IoT sensor systems is one of the most common use cases. As a result, SensorThings API's sensing part is designed based on the ISO/OGC Observation and Measurement (O&M) model [OGC 10-004r3 and ISO 19156:2011]. O&M defines models for the exchange of information describing observation acts, their results as well as the feature involved in sampling when making observations.

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

SensorThings API defines eight entities for the IoT sensing applications. Table 1 lists each component and its relationship with O&M. Low-cost and simple sensors are key enablers for the vision of IoT. As a result, SensorThings API uses the term of `Sensor` to describe the procedure that is used in making an `Observation`, instead of using O&M's term of procedure.

Table 1 SensorThings API Sensing entities and equivalent concepts in O&M 2.0

SensorThings API Entities	**O&M 2.0 Concepts**
`Thing (and Locations, HistoricalLocations)`	-
`Datastream`	-
`Sensor`	Procedure
`Observation`	Observation
`ObservedProperty`	Observed Property
`FeatureOfInterest`	Feature-Of-Interest

7.5 SensorThings API and OASIS OData

SensorThings API follows OData's specification for requesting entities. That means the entity control information, resource path usages, query options, the relevant JSON encodings, and batch-processing request follow OData 4.0. By using OData's standard ways for requesting entities, developers who are familiar with OData can create SensorThings applications easily. However, SensorThings API does not follow the OData Common Schema Definition Language and as a result does not follow its metadata service entity model. Thus, SensorThings API should not be seen as an OData compliant API. SensorThings API's future work will explore possible harmonization between SensorThings API and OData.

7.6 SensorThings API and OGC Key-Value Pair (KVP) Encodings

Please note that SensorThings API's Key-Value Pair (KVP) encoding is different from many existing OGC service implementation standards, such as SOS or Web Map Service (WMS). The main reason is that OData offers a complete set of KVP encodings (see Clause 9.3.3.6) that is designed specifically for RESTful web services, while OGC baseline currently does not have common KVP encodings for the RESTful binding. As a result, OGC SensorThings API version 1.0 chooses to use OData KVP encodings only. It is our future work to support OGC KVP encodings as an extension once a common OGC RESTful binding is available.

7.7 SensorThings API and Security

As things in the Internet of Things are connected to the network. Such ubiquitous network connectivity results in significant security threats. In the IoT reference model defined by ITU-T [ITU-T Y.2060] IoT security capabilities are not an independent layer but must be associated with all layers. The following figure show the ITU-T IoT reference model. The reference model has four layers, namely (1) Applications Layer, (2) Service Support and Application Support Layer, (3) Network Layer, and (4) Device Layer. And security capabilities are a cross-layer component that is associated with the four layers.

Based on the IoT reference model, SensorThings API falls into the scope of the Service Support and Application Support Layer and the security issues should be addressed by the cross-cutting security capabilities. As a result, SensorThings API does not define specific security capabilities. Instead SensorThings API is designed to leverage the existing and future IoT security capabilities.

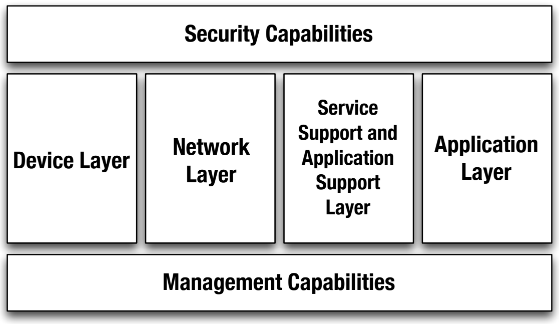

Figure 1 IoT Reference Model (adapted from [ITU-T Y.2060])

8. The SensorThings API Sensing Entities

8.1 Common Control Information

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information/common-control-information

Req 1	Each entity SHALL have the following common control information listed in Table 2.
http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information/common-control-information	

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

When annotating a name/value pair for which the value is represented as a JSON object, each annotation is placed within the object and represented as a single name/value pair. In SensorThings the name always starts with the "at" sign (`@`), followed by the namespace `iot`, followed by a dot (`.`), followed by the name of the term (e.g., `"@iot.id":1`).

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

Table 2 Common control information

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

8.2 The Sensing Entities

The SensorThings API Sensing part's Entities are depicted in Figure 2.

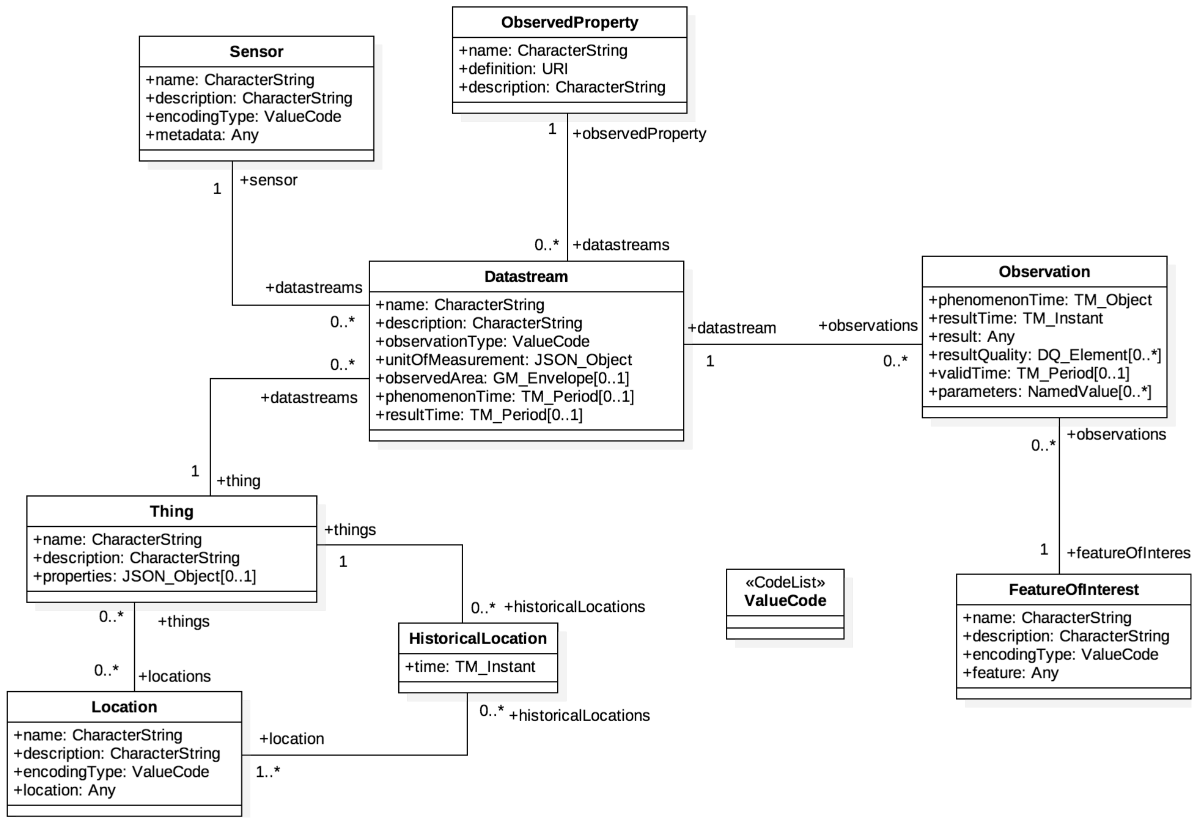

Figure 2 Sensing Entities

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

8.2.1 Thing

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/thing	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/thing/properties
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/thing/relations
Dependency	http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information/common-control-information

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 3.

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

Table 3 Properties of a `Thing` entity

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

Req 3 Each `Thing` entity SHALL have the direct relation between a `Thing` entity and other entity types listed in Table 4.

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

Table 4 Direct relation between a `Thing` entity and other entity types

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

Example 1 an example of a `Thing` entity:

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

8.2.2 Location

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/location	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/location/properties
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/location/relations
Dependency	http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information/common-control-information

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 10-004r3 and ISO 19156:2011] provides a detailed explanation of observation location.

Req 4 Each `Location` entity SHALL have the mandatory properties listed in Table 5.

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

Table 5 Properties of a `Location` entity

Name	Definition	Data type	Multiplicity and use
`name`	A property provides a label for `Location` entity, commonly a descriptive name.	CharacterString	One (mandatory)
`description`	The description about the `Location`.	CharacterString	One (mandatory)
`encodingType`	The encoding type of the `Location` property. Its value is one of the ValueCode enumeration (see Table 7).	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 6.

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

Table 6 Direct relation between a `Location` entity and other entity types

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

Example 2 an example of a Location entity:

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

Table 7 List of some code values used for identifying types for the `encodingType` of the `Location` and `FeatureOfInterest` entity

`Location encodingType`	**ValueCode Value**
GeoJSON	application/vnd.geo+json

A thing can be geo-referenced in different spaces. For example, for some applications it is more suitable to use a topological space model (e.g., IndoorGML) to describe an indoor things' location rather than using a geometric space model (e.g., GeoJSON). Currently GeoJSON is the only `Location` `encodingType` of the SensorThings API. In the future we expect to extend SensorThings API's capabilities by adding additional `encodingType` to the code values listed in the above table. For example, one potential new `Location` `encodingType` can be a JSON encoding for IndoorGML.

8.2.3 HistoricalLocation

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/historical-location	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/historical-location/properties
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/historical-location/relations
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/historical-location-auto-creation
Dependency	http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-

	information/common-control-information

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

Req 6 Each `HistoricalLocation` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8. http://www.opengis.net/spec/iot_sensing/1.0/req/historical-location/properties
Req 7 Each `HistoricalLocation` entity SHALL have the direct relation between a `Location` entity and other entity types listed in Table 9. http://www.opengis.net/spec/iot_sensing/1.0/req/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/create-update-delete/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 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 9 Direct relation between an HistoricalLocation entity and other entity types

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

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

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

8.2.4 Datastream

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/datastream	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/datastream/properties
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/datastream/relations
Dependency	http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information/common-control-information
Dependency	urn:iso:dis:iso:19156:clause:8.2.2

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

Req 9 Each `Datastream` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 10.

http://www.opengis.net/spec/iot_sensing/1.0/req/datastream/properties

Req 10 Each `Datastream` entity SHALL have the direct relation between a `Datastream` entity and other entity types listed in Table 11.

http://www.opengis.net/spec/iot_sensing/1.0/req/datastream/relations

Table 10 Properties of a `Datastream` entity

Name	Definition	Data type	Multiplicity and use
`name`	A property provides a label for `Datastream` entity, commonly a descriptive name.	CharacterString	One (mandatory)
`description`	The description of the `Datastream` entity.	CharacterString	One (mandatory)
`unitOfMeasurement`	A JSON Object containing three key-value pairs. The `name` property presents the full name of the `unitOfMeasurement`; the `symbol` property shows the textual form of the unit symbol; and the `definition` contains the URI defining the `unitOfMeasurement`. The values of these properties SHOULD follow the Unified Code for Unit of Measure (UCUM).	JSON Object	One (mandatory) Note: When a `Datastream` does not have a unit of measurement (e.g., a `OM_TruthObservation` type), the corresponding `unitOfMeasurement` properties SHALL have `null` values.
`observationType`	The type of `Observation` (with unique result type), which is used by the service to encode observations.	ValueCode see Table 12.	One (mandatory)
`observedArea`	The spatial bounding box of the spatial extent of all `FeaturesOfInterest` that belong to the `Observations` associated with this `Datastream`.	GM_Envelope (GeoJSON Polygon)	Zero-to-one (optional)
`phenomenonTime`	The temporal interval of the phenomenon times of all observations belonging to this `Datastream`.	TM_Period (ISO 8601 Time Interval)	Zero-to-one (optional)
`resultTime`	The temporal interval of the result times of all observations belonging to this `Datastream`.	TM_Period (ISO 8601 Time Interval)	Zero-to-one (optional)

Table 11 Direct relation between a `Datastream` entity and other entity types

Entity type	Relation	Description
`Thing`	Many optional to one mandatory	A `Thing` has zero-to-many `Datastreams`. A `Datastream` entity SHALL only link to a `Thing` as a collection of `Observations`.

`Sensor`	Many optional to one mandatory	The `Observations` in a `Datastream` are performed by one-and-only-one `Sensor`. One `Sensor` MAY produce zero-to-many `Observations` in different `Datastreams`.
`ObservedProperty`	Many optional to one mandatory	The `Observations` of a `Datastream` SHALL observe the same `ObservedProperty`. The `Observations` of different `Datastreams` MAY observe the same `ObservedProperty`.
`Observation`	One mandatory to many optional	A `Datastream` has zero-to-many `Observations`. One `Observation` SHALL occur in one-and-only-one `Datastream`.

Example 4 A `Datastream` entity example:

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

The `observationType` defines the result types for specialized observations [OGC 10-004r3 and ISO 19156:2011 Table 3]. The following table shows some of the `valueCodes` that maps the UML classes in O&M v2.0 [OGC 10-004r3 and ISO 19156:2011] to `observationType` names and observation `result` types.

Table 12 List of some code values used for identifying types defined in the O&M conceptual model (OGC 10-004r3 and ISO 19156:2011 Clause 8.2.2)

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

OM_Observation	`http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Observation`	Any
OM_TruthObservation	`http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_TruthObservation`	boolean

8.2.5 Sensor

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/sensor	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/sensor/properties
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/sensor/relations
Dependency	http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information/common-control-information

A `Sensor` is an instrument that observes a property or phenomenon with the goal of producing an estimate of the value of the property[3].

Req 11 Each `Sensor` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 13.   http://www.opengis.net/spec/iot_sensing/1.0/req/sensor/properties
Req 12 Each `Sensor` entity SHALL have the direct relation between a `Sensor` entity and other entity types listed in Table 14.   http://www.opengis.net/spec/iot_sensing/1.0/req/sensor/relations

Table 13 Properties of a `Sensor` entity

Name	Definition	Data type	Multiplicity and use

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

name	A property provides a label for `Sensor` entity, commonly a descriptive name.	CharacterString	One (mandatory)
description	The description of the `Sensor` entity.	CharacterString	One (mandatory)
encodingType	The encoding type of the `metadata` property. Its value is one of the ValueCode enumeration (see Table 15 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 14 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 15 List of some code values used for identifying types for the `encodingType` of the `Sensor` entity

Sensor encodingType	**ValueCode Value**
PDF	`application/pdf`
SensorML	`http://www.opengis.net/doc/IS/SensorML/2.0`

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

Example 5 An example of a `Sensor` entity:

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

8.2.6 ObservedProperty

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/observed-property	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/observed-property/properties
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/observed-property/relations
Dependency	http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information/common-control-information

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

Req 13 Each `ObservedProperty` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 16.

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

Req 14 Each `ObservedProperty` entity SHALL have the direct relation between a `ObservedProperty` entity and other entity types listed in Table 17.

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

Table 16 Properties of an `ObservedProperty` entity

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

Table 17 Direct relation between an `ObservedProperty` entity and other entity types

Entity type	**Relation**	**Description**
`Datastream`	One mandatory to many	The `Observations` of a `Datastream` observe the same `ObservedProperty`. The `Observations` of different `Datastreams` MAY observe the same `ObservedProperty`.

	optional	

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"
}
```

8.2.7 Observation

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/observation	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/observation/properties
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/observation/relations
Dependency	http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information/common-control-information
Dependency	urn:iso:dis:iso:19156:clause:7.2.2

An `Observation` is the act of measuring or otherwise determining the value of a property [OGC 10-004r3 and ISO 19156:2011]

Req 15 Each `Observation` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 18.

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

Req 16 Each `Observation` entity SHALL have the direct relation between an `Observation` entity and other entity types listed in Table 19.

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

Table 18 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 19 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 }
```

8.2.8 FeatureOfInterest

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/feature-of-interest	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/feature-of-interest/properties
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/feature-of-interest/relations
Dependency	http://www.opengis.net/spec/iot_sensing/1.0/req/entity-control-information/common-control-information

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 `Observation`s' `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 17	Each `FeatureOfInterest` entity SHALL have the mandatory properties listed in Table 20. http://www.opengis.net/spec/iot_sensing/1.0/req/feature-of-interest/properties
Req 18	Each `FeatureOfInterest` entity SHALL have the direct relation between a `FeatureOfInterest` entity and other entity types listed in Table 21. http://www.opengis.net/spec/iot_sensing/1.0/req/feature-of-interest/relations

Table 20 Properties of a `FeatureOfInterest` entity

Name	**Definition**	**Data type**	**Multiplicity and use**
`name`	A property provides a label for `FeatureOfInterest` entity, commonly a descriptive name.	CharacterString	One (mandatory)
`description`	The description about the `FeatureOfInterest`.	CharacterString	One (mandatory)
`encodingType`	The encoding type of the feature property. Its value is one of the ValueCode enumeration (see Table 7 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 21 Direct relation between a `FeatureOfInterest` entity and other entity types

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

Example 8 an example of a `FeatureOfInterest` entity

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

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

```
"v"majorversionnumber + "." + minorversionnumber
```

Example 9 complete URI example

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

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

9.2 Resource Path

Requirements Class
http://www.opengis.net/spec/iot_sensing/1.0/req/resource-path

Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/resource-path/resource-path-to-entities
Dependency	

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 19 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/resource-path/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

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/request-data	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/order
Requirement	1.1.1.1.1.1.1.1.1 http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/expand
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/select
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/status-code
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/query-status-code
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/orderby
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/top
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/skip
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/count
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/filter

Dependency	http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398292
Dependency	http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398297
Dependency	http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398299
Dependency	http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398300

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. 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 20 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 SHALL 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

Req 21 If a service does not support a system query option, it SHALL fail any request that contains the unsupported option and SHALL return `501 Not Implemented`.

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

9.3.1 Evaluating System Query Options

Req 22 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 23 The `$expand` system query option indicates the related entities to be represented inline. The value of the `$expand` query option SHALL 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.

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

Example 18: examples of `$expand` query option

Example Request 1: `http://example.org/v1.0/Things?$expand=Datastreams` returns the entity set of `Things` as well as each of the `Datastreams` associated with each `Thing` entity.

Example Request 1 Response:

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

Example Request 2:
`http://example.org/v1.0/Things?$expand=Datastreams/ObservedProperty` returns the collection of `Things`, the `Datastreams` associated with each `Thing`, and the `ObservedProperty` associated with each `Datastream`.

Example Request 3:
`http://example.org/v1.0/Datastreams(1)?$expand=Observations,ObservedProperty` returns the `Datastream` whose `id` is 1 as well as the `Observations` and `ObservedProperty` associated with this `Datastream`.

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

[Adapted from OData 4.0- URL 5.1.2]

Example Request 4:
`http://example.org/v1.0/Datastreams(1)?$expand=Observations($filter=result eq 1)` returns the `Datastream` whose `id` is 1 as well as its `Observations` with a `result` equal to 1.

9.3.2.2 $select

Req 24 The `$select` system query option requests the service to return only the properties explicitly requested by the client. The value of a `$select` query option SHALL be a comma-separated list of selection clauses. Each selection clause SHALL be a property name (including navigation property names). In the response, the service SHALL return the specified content, if available, along with any available expanded navigation properties.

[Adapted from OData 4.0-Protocol 11.2.4.1]

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

Example 19: examples of `$select` query option

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

Example Request 2:
`http://example.org/v1.0/Datastreams(1)?$select=id,Observations&$expand=Observations/FeatureOfInterest` returns the `id` property of the `Datastream` entity, and all the properties of the entity identified by the `Observations` and `FeatureOfInterest` navigation properties.

Example Request 3:
`http://example.org/v1.0/Datastreams(1)?$expand=Observations($select=result)` returns the `Datastream` whose `id` is 1 as well as the result property of the entity identified by the `Observations` navigation property.

9.3.3 Query Entity Sets

9.3.3.1 $orderby

Req 25 The `$orderby` system query option specifies the order in which items are returned from the service. The value of the `$orderby` system query option SHALL contain 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 SHALL be ordered by a property defined on a derived type.

The expression MAY 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 SHALL order by the specified property in ascending order.

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

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

[Note: Adapted from OData 4.0-Protocol 11.2.5.2]

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

Example 20: examples of `$orderby` query option

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

Example Request 2:
`http://example.org/v1.0/Observations?$expand=Datastream&$orderby=Datastreams/id desc, phenomenonTime` returns all `Observations` ordered by the `id` property of the linked `Datastream` entry in descending order, then by the `phenomenonTime` property of `Observations` in ascending order.

9.3.3.2 $top

Req 26 The `$top` system query option specifies the limit on the number of items returned from a collection of entities. The value of the `$top` system query option SHALL be a non-negative integer n. The service SHALL return the number of available items up to but not greater than the specified value n.

If no unique ordering is imposed through an `$orderby` query option, the service SHALL impose a stable ordering across requests that include `$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 SHALL be discarded and the server-side pagination limitation SHALL be imposed.

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

Example 21: examples of `$top` query option

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

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

9.3.3.3 $skip

Req 27 The `$skip` system query option specifies the number for the items of the queried collection that SHALL be excluded from the result. The value of `$skip` system query option SHALL be a non-negative integer n. The service SHALL return items starting at position n+1.

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

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

[Note: Adapted from OData 4.0-Protocol 11.2.5.4]

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

Example 22: examples of `$skip` query option

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

Example Request 2:
`http://example.org/v1.0/Observations?$skip=2&$top=2&$orderby=resultTime` returns the third and fourth `Observation` entities from the collection of all `Observation` entities when the collection is sorted by the `resultTime` property in ascending order.

9.3.3.4 $count

Req 28 The `$count` system query option with a value of true specifies that the total count of items within a collection matching the request SHALL be returned along with the result. A `$count` query option with a value of `false` (or not specified) hints that the service SHALL not return a count.

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

The `$count` system query option SHALL ignore any `$top`, `$skip`, or `$expand` query options, and SHALL return 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]

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

Example 23: examples of `$count` query option

Example Request 1: `http://example.org/v1.0/Things?$count=true` return, along with the `results`, the total number of `Things` in the collection.

Example Response:

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

9.3.3.5 $filter

Req 29 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 SHALL be 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, SHALL be omitted from the response.

[Adapted from Data 4.0-URL Conventions 5.1.1]

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

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

Example 24: examples of `$filter` query option

Example Request 1: `http://example.org/v1.0/Observations?$filter=result lt 10.00` returns all `Observations` whose `result` is less than 10.00.

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

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

Example Request 3:
`http://example.org/v1.0/Things?$filter=geo.distance(Locations/location, geography'POINT(-122, 43)') gt 1` returns Things that the distance between their last known locations and `POINT(-122 43)` is greater than 1.
Example Request 4:
`http://example.org/v1.0/Things?$expand=Datastreams/Observations/FeatureOfInterest&$filter=Datastreams/Observations/FeatureOfInterest/id eq 'FOI_1' and Datastreams/Observations/resultTime ge 2010-06-01T00:00:00Z and Datastreams/Observations/resultTime le 2010-07-01T00:00:00Z` returns `Things` that have any observations of a feature of interest with a unique identifier equals to `'FOI_1'` in June 2010.

9.3.3.5.1 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 30 The built-in filter operators SHALL be as defined in Table 22.

http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/built-in-filter-operations

Table 22 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`

9.3.3.5.2 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 31 The built-in query functions SHALL be as defined in Table 23.

http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/built-in-query-functions

Table 23 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********')`

9.3.3.6 Server-Driven Paging (`nextLink`)

Req 32 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]

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

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. SensorThings Sensing Create-Update-Delete

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/create-entity
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/link-to-existing-entities
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/deep-insert
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/deep-insert-status-code
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/update-entity
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/delete-entity
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/historical-location-auto-creation
Dependency	http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398328
Dependency	http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398329

10.1 Overview

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

10.2 Create an entity

Req 33 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 24. 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 24 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 /v1.0/Things HTTP/1.1

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

{
   "name": "thermostat",
   "description":"This is a smart thermostat with WiFi communication
capabilities."
}
```

10.2.1.1 Link to existing entities when creating an entity

Req 34 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 /v1.0/Observations HTTP/1.1
Host: example.org/
Content-Type: application/json

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

10.2.1.2 Create related entities when creating an entity

Req 35 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 /v1.0/Things HTTP1.1
Host: example.org/
Content-Type: application/json

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

10.2.2 Response

Req 36 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 Update an entity

Req 37 To update an entity in a collection a SensorThings service SHALL follow the requirements as defined in Section 10.3.

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

10.3.1 Request

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

The semantics of `PATCH`, as defined in [RFC5789], are to merge the content in the request payload with the entity's current state, applying the update only to those components specified in the request body. The properties provided in the payload corresponding to updatable properties SHALL replace the value of the corresponding property in the entity. Missing properties of the containing entity or complex property SHALL NOT be directly altered.

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

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

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

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

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

Services MAY additionally support `JSON PATCH` format [RFC6902] to express a sequence of operations to apply to a SensorThings entity.

[Adapted from OData 4.0-Protocol 11.4.3]

HTTP Method: `PATCH` or `PUT`

URI Pattern: An URI addressing to a single entity.

Header: `Content-Type: application/json`

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

Example 29: update the `Thing` whose `id` is 1.

```
PATCH /v1.0/Things(1) HTTP1.1

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

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

10.3.2 Response

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

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

10.4 Delete an entity

Req 38 To delete 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/delete-entity

10.4.1 Request

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

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

Services MAY implicitly delete or modify related entities if required by integrity constraints. Table 25 listed SensorThings API's integrity constraints when deleting an entity.

HTTP Method: `DELETE`

URI Pattern: An URI addressing to a single entity.

Example 30: delete the `Thing` with unique identifier equals to 1

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

Table 25 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

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/batch-request	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/batch-request/batch-request
Dependency	http://docs.oasis-open.org/odata/odata/v4.0/errata02/os/complete/part1-protocol/odata-v4.0-errata02-os-part1-protocol-complete.html#_Toc406398359

Req 39	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/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

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

Note: For brevity, in the example, request bodies are excluded in favor of English descriptions inside <> brackets.

Note also that the two empty lines after the Host header of the `GET` request are necessary: the first is part of the `GET` request header; the second is the empty body of the `GET` request, followed by a `CRLF` according to [RFC2046].

[Adapted from OData 4.0 Protocol 11.7.2]

Example 31-2: a Batch Request body example

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

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

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


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

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

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

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

PATCH /v1.0/Things(1) HTTP/1.1
Host: host
Content-Type: application/json
If-Match: xxxxx
Content-Length: ###

<JSON representation of Things(1)>
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_36522ad7-fc75-4b56-8c71-56071383e77b
Content-Type: application/http
Content-Transfer-Encoding: binary

GET /v1.0/Things(3) HTTP/1.1
Host: host


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

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:

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

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

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

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

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

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

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

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

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 time only the first request in the batch finished processing and all the remaining requests except the final query request succeed. In this case the response would be:

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

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

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

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

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


--b_243234_25424_ef_892u748--
```

Client makes a second request using the returned monitor URL:

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

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

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

HTTP/1.1 201 Created
Content-Type: application/json
```

```
Location: http://host/v1.0/Things(99)
Content-Length: ###

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

HTTP/1.1 204 No Content
Host: host


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

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

<Error message>
--b_243234_25424_ef_892u748–
```

12. SensorThings `MultiDatastream` extension

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/properties
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/relations
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/constraints

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

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

The `MultiDatastream` extension entities are depicted in Figure 3.

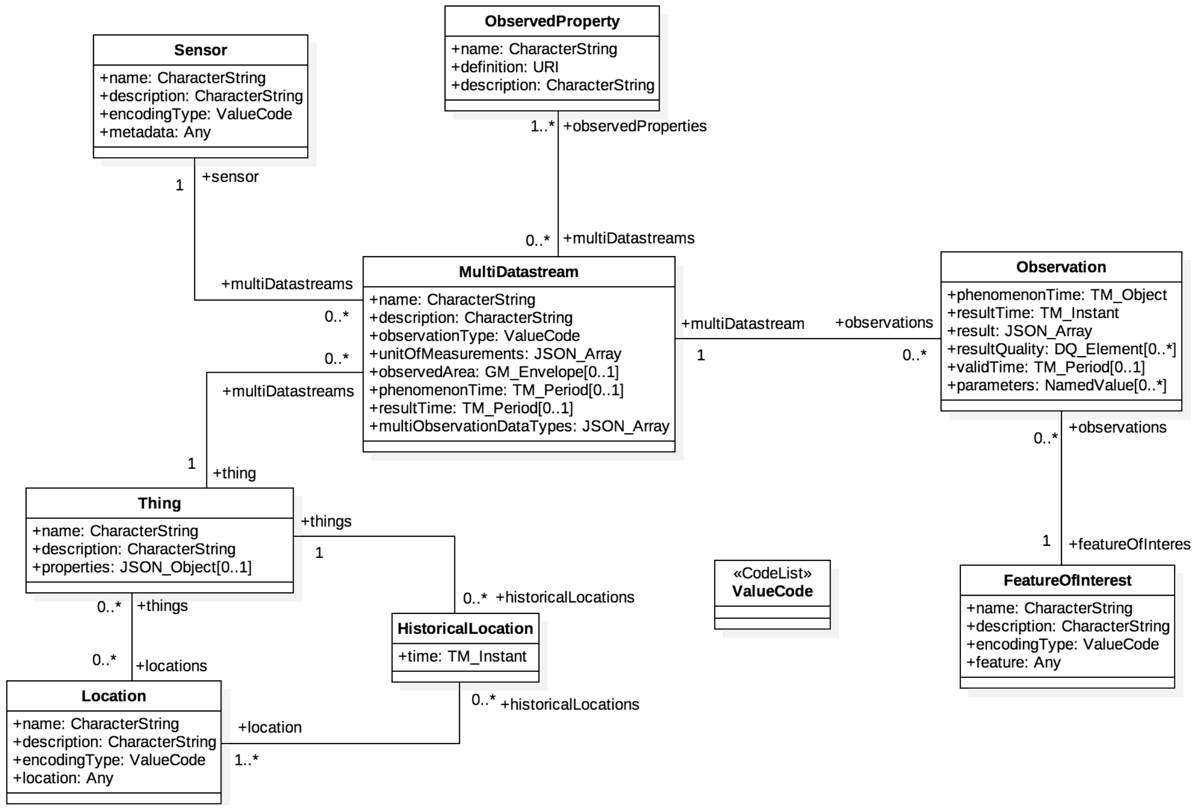

Figure 3 MultiDatastream Extension Entities

Req 40 Each `MultiDatastream` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 26. http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/properties	
Req 41 Each `MultiDatastream` entity SHALL have the direct relation between a `Datastream` entity and other entity types listed in Table 27. http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/relations	

Table 26 Properties of a `MultiDatastream` entity

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

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 URI defining the `unitOfMeasurement`. (see Req 42 for the constraints between `unitOfMeasurement`, `multiObservationDataType` and `result`)	A JSON array	One (mandatory) Note: It is possible an observation does not have a unit of measurement. For example, a count observation does not have a unit of measurement.
observationType	The type of `Observation` (with unique result type), which is used by the service to encode observations.	ValueCode and its value SHALL be `OM_ComplexObservation`.	One (mandatory)
multiObservationDataTypes	This property defines the `observationType` of each element of the result of a complex `Observation`.	A JSON array of ValueCode. See Table 12 for the available ValueCodes.	One (mandatory)
observedArea	The spatial bounding box of the spatial extent of all `FeatureOfInterests` that belong to the `Observations` associated with this `MultiDatastream`.	GM_Envelope (GeoJSON Polygon)	Zero-to-one
phenomenonTime	The temporal interval of the phenomenon times of all observations belonging to this `MultiDatastream`.	TM_Period (ISO 8601 Time Interval)	Zero-to-one
resultTime	The temporal interval of the result times of all observations belonging to this `MultiDatastream`.	TM_Period (ISO 8601 Time Interval)	Zero-to-one

Table 27 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	The `Observations` of a `MultiDatastream` SHALL observe the same `ObservedProperties` entity set.

	mandatory	
Observation	One mandatory to many optional	A MultiDatastream has zero-to-many Observations. One Observation SHALL occur in one-and-only-one MultiDatastream.

Table 28 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 42 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",
  "name": "temperature, RH, visibility",
  "description":  "This  is  a  MultiDatastream  from  a  simple  weather  station
    measuring air temperature, relative humidity and visibility",
  "observationType":             "http://www.opengis.net/def/observationType/OGC-
   OM/2.0/OM_ComplexObservation",
  "multiObservationDataTypes": [
    "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
    "http://www.opengis.net/def/observationType/OGC-OM/2.0/OM_Measurement",
    "http://www.opengis.net/def/observationType/OGC-
OM/2.0/OM_CategoryObservation"
  ],
  "unitOfMeasurements": [
    {
      "name": "degree Celsius",
      "symbol": " °C",
      "definition": " http://unitsofmeasure.org/ucum.html#para-30"
    },
    {
      "name": " percent ",
      "symbol": "%",
      "definition": " http://unitsofmeasure.org/ucum.html#para-29"
    },
    {
      "name": "null",
      "symbol": "null",
      "definition": "null"
      }
     ],
     "observedArea": {
       "type": "Polygon",
       "coordinates": [
         [
           [100,0],[101,0],[101,1],[100,1],[100,0]
         ]
       ]
     },
     "phenomenonTime": "2014-03-01T13:00:00Z/2015-05-11T15:30:00Z",
     "resultTime": "2014-03-01T13:00:00Z/2015-05-11T15:30:00Z"
}
```

Example 33: an example `ObservedProperties` collection of the above `MultiDatastream`: Please note that the order of the elements in the `value` array match the order of the related `Observations/result` array as well as the order of the related `unitOfMeasurements` array.

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

Example 34: an example `Observation` of the above `MultiDatastream`: Please note that the order of the elements in the `result` array match (1) the order of the related `ObservedProperties` (*i.e.*, `Observation(id)/MultiDatastreams(id)/ObservedProperties`), (2) the order of the related `unitOfMeasurements` array (*i.e.*, `Observation(id)/MultiDatastream(id)/unitOfMeasurements`) and (3) the order of the related `multiObservationDataTypes` (*i.e.*, `Observation(id)/MultiDatastream(id)/multiObservationDataTypes).`

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

13. SensorThings Data Array Extension

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/data-array	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/ req/data-array/data-array

Req 43 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 29 Properties of getting `Observation` entities in `dataArray`

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

Example 35: an example of getting `Observation` entities from a `Datastream` in `dataArray` result format:

```
GET /v1.0/Datastreams(1)/Observations?$resultFormat=dataArray
HTTP/1.1 200 OK
Host: www.example.org/
Content-Type: application/json

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

Example 36: an example of getting `Observation` entities from a `MultiDatastream` in `dataArray` result format

```
GET /v1.0/MultiDatastreams(1)/Observations?$resultFormat=dataArray
HTTP/1.1 200 OK
Host: www.example.org
Content-Type: application/json

{      "value":   [          {                "MultiDatastream@iot.navigationLink":
"MultiDatastreams(1)",              "components":   [                    "id",
"phenomenonTime",              "resultTime",              "result"          ],
"dataArray@iot.count":3,          "dataArray":  [            [              1,
"2010-12-23T11:20:00-0700",            "2010-12-23T11:20:00-0700",            [
10.2,            65,              "clear"            ]          ],           [
2,           "2010-12-23T11:22:08-0700",            "2010-12-23T11:20:00-0700",
[            11.3,           63,             "clear"            ]          ],
[          3,           "2010-12-23T11:22:54-0700",          "2010-12-23T11:20:00-
0700",           [            9.8,             67,             "clear"          ]
]       ]     }   ] }
```

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 30 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 `Observatio n` property names	One (mandatory)
`dataArray`	A JSON Array containing `Observations`. Each `Observation` is represented by the ordered property values. The ordered property values match with the ordered property names in `components`.	JSON Array	One (mandatory)

Example 37: example of a request for creating `Observation` entities in `dataArray`

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

[    {       "Datastream":  {          "@iot.id":  1       },       "components":  [
"phenomenonTime",              "result",             "FeatureOfInterest/id"        ],
"dataArray@iot.count":2,      "dataArray":  [        [          "2010-12-23T10:20:00-
0700",         20,         1       ],       [          "2010-12-23T10:21:00-0700",
30,         1       ]      ]   },   {     "Datastream": {       "@iot.id": 2     },
"components":   [                      "phenomenonTime",                    "result",
"FeatureOfInterest/id"       ],       "dataArray@iot.count":2,       "dataArray":  [
[           "2010-12-23T10:20:00-0700",          65,          1        ],        [
"2010-12-23T10:21:00-0700",         60,         1       ]      ]   } ]
```

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 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 Observation with MQTT Publish

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/create-observations-via-mqtt	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/create-observations-via-mqtt/observations-creation
Dependency	http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

Req 44 To allow clients to create observations with `MQTT Publish`, a service SHALL support the creation of observations with MQTT as defined in Section 14.1.

http://www.opengis.net/spec/iot_sensing/1.0/req/create-observations-via-mqtt/observations-creation

SensorThings MQTT extension provides the capability of creating Observation entity using MQTT protocol. To create an Observation entity in MQTT, the client sends a `MQTT Publish` request to the SensorThings service and the MQTT topic is the Observations resource path. The MQTT application message contains a single valid Observation entity representation. Figure 4 contains the sequence diagram for creating Observation using MQTT publish as well as MQTT sending notifications for Observation creation.

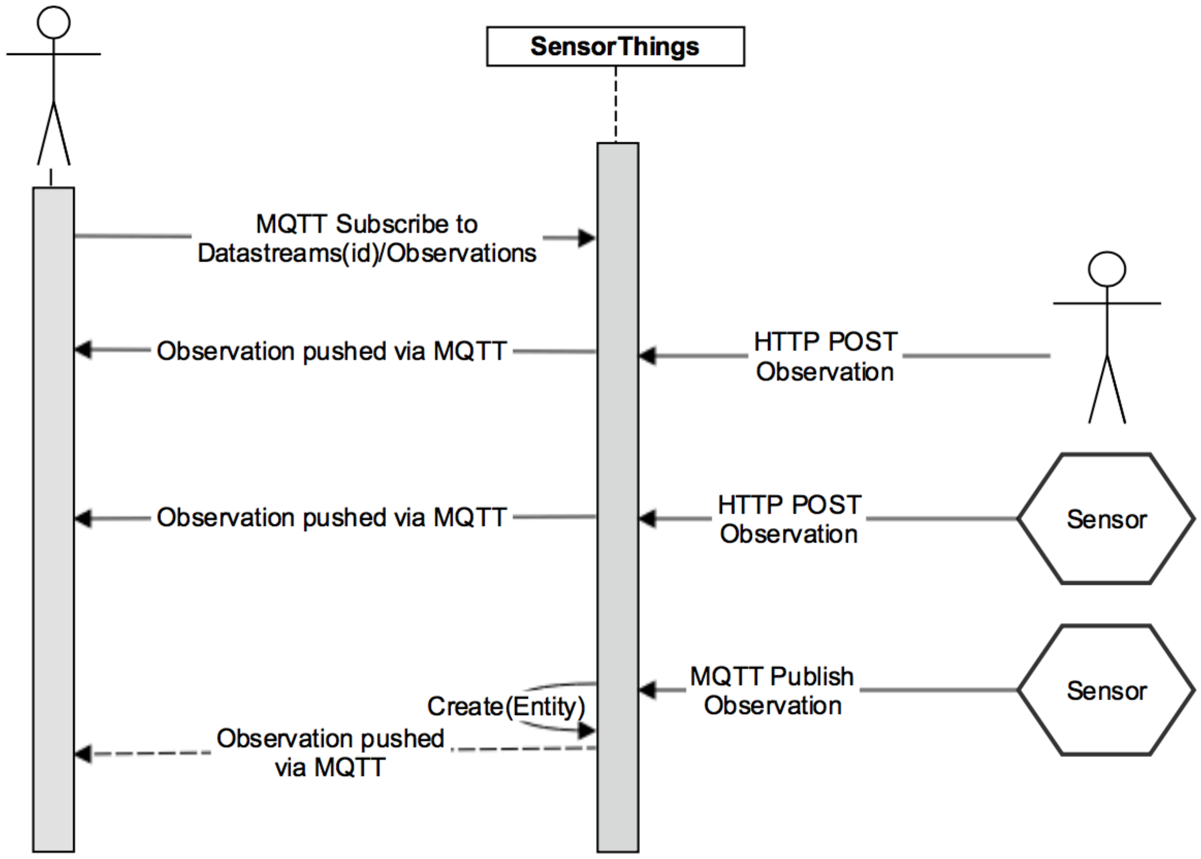
SensorThings
MQTT Subscribe to Datastreams(id)/Observations
Observation pushed via MQTT
HTTP POST Observation
Observation pushed via MQTT
HTTP POST Observation
Sensor
MQTT Publish Observation
Sensor
Create(Entity)
Observation pushed via MQTT

Figure 4 Creating Observations using MQTT publish, and receive notifications for Observations with MQTT

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

Similar to creating Observations with `HTTP POST`, creating Observations with `MQTT Publish` follow the integrity constraints for creating Observation listed in **Error! Reference source not found.**. The two special cases defined in Req 33 are also applied in the case of creating Observations with `MQTT Publish`.

14.1.1 Link to existing entities when creating an Observation entity

To link to existing entities when creating an Observation entity with MQTT, the conditions in Req 34 is applied.

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

To create related entities when creating an entity with MQTT, the condition in Req 35 is applied.

14.2 Receive updates with MQTT Subscribe

Requirements Class	
http://www.opengis.net/spec/iot_sensing/1.0/req/receive-updates-via-mqtt	
Target Type	Web Service
Requirement	http://www.opengis.net/spec/iot_sensing/1.0/req/receive-updates-via-mqtt/receive-updates
Dependency	http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

Req 45 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 14.2.

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

To receive notifications from a SensorThings service when some entities updated, a client can send a `MQTT Subscribe` request to the SensorThings service. SensorThings API defined the following four MQTT subscription use cases. Figure 5 contains the sequence diagram of receiving updates using `MQTT Subscribe`.

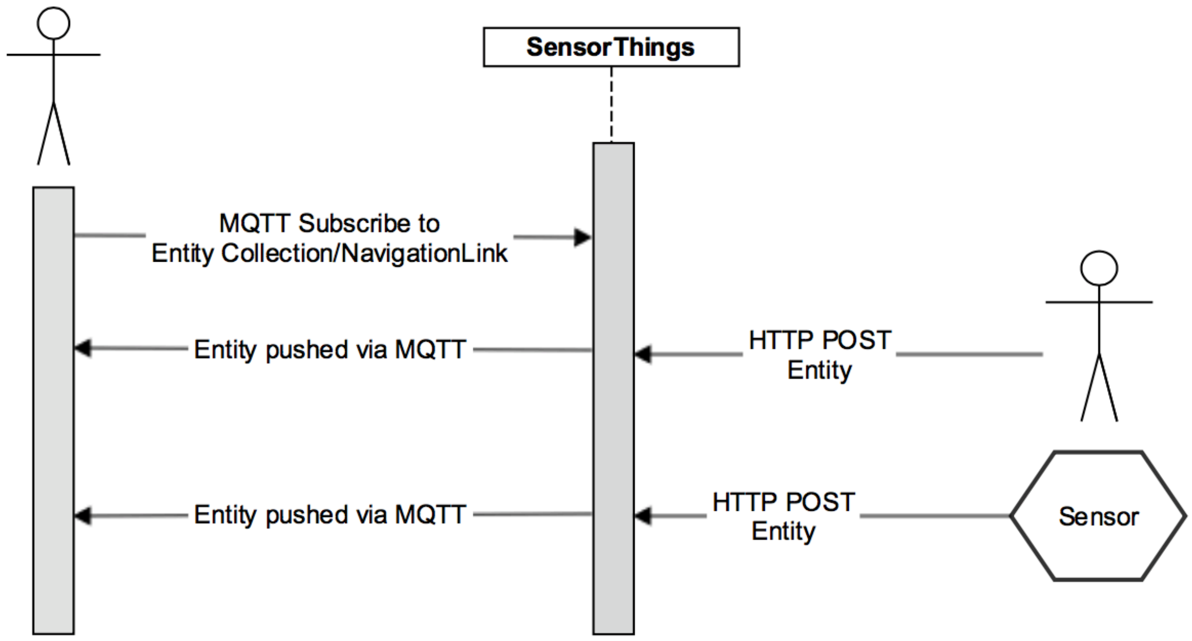

Figure 5 Sequence diagram for receiving updates using MQTT subscribe

14.2.1 Receive updates of a SensorThings entity set with `MQTT Subscribe`

MQTT Control Packet: `Subscribe`

Topic Pattern: `RESOURCE_PATH/COLLECTION_NAME`

Example Topic: `Datastreams(1)/Observations`

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

14.2.2 Receive updates of a SensorThings entity with `MQTT Subscribe`

MQTT Control Packet: `Subscribe`

Topic Pattern: `RESOURCE_PATH_TO_AN_ENTITY`

Example Topic: `Datastreams(1)`

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

14.2.3 Receive updates of a SensorThings entity's property with `MQTT Subscribe`

MQTT Control Packet: `Subscribe`

Topic Pattern: `RESOURCE_PATH_TO_AN_ENTITY/PROPERTY_NAME`

Example Topic: `Datastreams(1)/observedArea`

Response: When the value of the subscribed property is changed, the service returns a JSON object. The returned JSON object follows as defined in Section 9.2.4 - Usage 4: address to a property of an entity.

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

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

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

MQTT Control Packet: `Subscribe`

Topic Pattern: `RESOURCE_PATH/COLLECTION_NAME?$select=PROPERTY_1,PROPERTY_2,…`

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

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

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

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

Annex A: Conformance Class Abstract Test Suite (Normative)

A.1 SensorThings Read (Core) Tests

This section contains the conformance classes for the SensorThings API Read (Core). The SensorThings API service needs to pass all the conformance tests defined in this section.

A.1.1 Conformance class: SensorThings API Entity Control Information

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information/common-control-information

Test: Common Control Information

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information/common-control-information
Requirements	1. http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-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/entity-control-information/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 defined in the above requirement and the service sends appropriate responses as defined in this specification.

A.1.2 Conformance class: SensorThings API `Thing` Entity

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

Test: `Thing` Entity

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/thing/thing-valid
Requirements	2. http://www.opengis.net/spec/iot_sensing/1.0/req/thing/properties 3. http://www.opengis.net/spec/iot_sensing/1.0/req/thing/relations
Test purpose	Check if each `Thing` entity has the mandatory properties and mandatory relations as defined in this specification.
Test method	Inspect the full JSON object of the `Thing` entity sets (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties defined in the corresponding requirement. Inspect the full JSON object of each `Thing` entity set (*i.e.*, without using the `$select` query option) to identify, if each entity has the mandatory relations (*i.e.*,

	`@iot.navigationLink`) defined in the corresponding requirement.

A.1.3 Conformance class: SensorThings API `Location` Entity

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

Test: `Location` Entity

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/location/location-valid
Requirements	4. http://www.opengis.net/spec/iot_sensing/1.0/req/location/properties 5. http://www.opengis.net/spec/iot_sensing/1.0/req/location/relations
Test purpose	Check if each `Location` entity has the mandatory properties and mandatory relations as defined in this specification.
Test method	Inspect the full JSON object of the `Location` entity sets (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties defined in the corresponding requirement. Inspect the full JSON object of each `Location` entity set (*i.e.*, without using the `$select` query option) to identify, if each entity has the mandatory relations (*i.e.*, `@iot.navigationLink`) defined in the corresponding requirement.

A.1.4 Conformance class: SensorThings API `HistoricalLocation` Entity

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location

Test: `Historicalocation` Entity

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location/ historical-location-valid
Requirements	6. http://www.opengis.net/spec/iot_sensing/1.0/req/historical-location/properties 7. http://www.opengis.net/spec/iot_sensing/1.0/req/historical-location/relations
Test purpose	Check if each `Historicalocation` entity has the mandatory properties and mandatory relations as defined in this specification.
Test method	Inspect the full JSON object of the `Historicalocation` entity sets (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties defined in the corresponding requirement. Inspect the full JSON object of each `Historicalocation` 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 requirement.

A.1.5 Conformance class: SensorThings API `Datastream` Entity

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

Test: `Datastream` Entity

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/datastream/datastream-valid
Requirements	9. http://www.opengis.net/spec/iot_sensing/1.0/req/datastream/properties 10. http://www.opengis.net/spec/iot_sensing/1.0/req/datastream/relations
Test purpose	Check if each `Datastream` entity has the mandatory properties and mandatory relations as defined in this specification.
Test method	Inspect the full JSON object of the `Datastream` entity sets (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties defined in the corresponding requirement. Inspect the full JSON object of each `Datastream` entity set (*i.e.*, without using the `$select` query option) to identify, if each entity has the mandatory relations (*i.e.*, `@iot.navigationLink`) defined in the corresponding requirement.

A.1.6 Conformance class: SensorThings API `Sensor` Entity

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

Test: `Sensor` Entity

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/sensor/sensor-valid
Requirements	11. http://www.opengis.net/spec/iot_sensing/1.0/req/sensor/properties 12. http://www.opengis.net/spec/iot_sensing/1.0/req/sensor/relations
Test purpose	Check if each `Sensor` entity has the mandatory properties and mandatory relations as defined in this specification.
Test method	Inspect the full JSON object of the `Sensor` entity sets (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties defined in the corresponding requirement. Inspect the full JSON object of each `Sensor` entity set (*i.e.*, without using the `$select`

	query option) to identify, if each entity has the mandatory relations (*i.e.*, `@iot.navigationLink`) defined in the corresponding requirement.

A.1.7 Conformance class: SensorThings API `ObservedProperty` Entity

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property

Test: `ObservedProperty` Entity

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property/observed-property-valid
Requirements	13. http://www.opengis.net/spec/iot_sensing/1.0/req/observed-property/properties 14. http://www.opengis.net/spec/iot_sensing/1.0/req/observed-property/relations
Test purpose	Check if each `ObservedProperty` entity has the mandatory properties and mandatory relations as defined in this specification.
Test method	Inspect the full JSON object of the `ObservedProperty` entity sets (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties defined in the corresponding requirement. Inspect the full JSON object of each `ObservedProperty` entity set (*i.e.*, without using the `$select` query option) to identify, if each entity has the mandatory relations (*i.e.*, `@iot.navigationLink`) defined in the corresponding requirement.

A.1.8 Conformance class: SensorThings API `Observation` Entity

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

Test: `Observation` Entity

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/observation/observation-valid
Requirements	15. http://www.opengis.net/spec/iot_sensing/1.0/req/observation/properties 16. http://www.opengis.net/spec/iot_sensing/1.0/req/observation/relations
Test purpose	Check if each `Observation` entity has the mandatory properties and mandatory relations as defined in this specification.
Test method	Inspect the full JSON object of the `Observation` entity sets (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties defined in the corresponding requirement.

	Inspect the full JSON object of each `Observation` 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 requirement.

A.1.9 Conformance class: SensorThings API `FeatureOfInterest` Entity

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest

Test: `FeatureOfInterest` Entity

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest/feature-of-interest-valid
Requirements	17. http://www.opengis.net/spec/iot_sensing/1.0/req/feature-of-interest/properties 18. http://www.opengis.net/spec/iot_sensing/1.0/req/feature-of-interest/relations
Test purpose	Check if each `FeatureOfInterest` entity has the mandatory properties and mandatory relations as defined in this specification.
Test method	Inspect the full JSON object of the `FeatureOfInterest` entity sets (*i.e.*, without `$select`) to identify, if each entity has the mandatory properties defined in the corresponding requirement. Inspect the full JSON object of each `FeatureOfInterest` entity set (*i.e.*, without using the `$select` query option) to identify, if each entity has the mandatory relations (*i.e.*, `@iot.navigationLink`) defined in the corresponding requirement.

A.1.10 Conformance class: SensorThings API Resource Path

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path

Test: Resource Path

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path/resource-path-to-entities
Requirements	19. http://www.opengis.net/spec/iot_sensing/1.0/req/resource-path/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/resource-path/resource-path-to-entities.
Test method	Inspect the service to identify, if each resource path usage has been implemented property.

A.2 SensorThings API Filtering Extension Tests

This section contains the conformance classes for the SensorThings API filtering extension. That means a SensorThings API service that allows clients to further filter data with query options needs to pass the conformance tests defined in this section.

A.2.1 Conformance class: SensorThings API Request Data with Filters

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

Dependencies:

1. http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information
2. http://www.opengis.net/spec/iot_sensing/1.0/conf/thing
3. http://www.opengis.net/spec/iot_sensing/1.0/conf/location
4. http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location
5. http://www.opengis.net/spec/iot_sensing/1.0/conf/datastream
6. http://www.opengis.net/spec/iot_sensing/1.0/conf/sensor
7. http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property
8. http://www.opengis.net/spec/iot_sensing/1.0/conf/observation
9. http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest
10. http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path

A.2.1.1 Test: Query Option Order

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/request-data/order
Requirements	22. 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.2.1.2 Test: Request Data with `$expand` and `$select`

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/request-data/expand-and-select
Requirements	23. http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/expand 24. 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.2.1.3 Test: Query Option Response Code

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/request-data/status-codes
Requirements	20. http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/status-code 21. http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/query-status-code
Test purpose	Check when a client requests an entity that is not available in the service, if the service responds with `404 Not Found` or `410 Gone` as defined in the requirement http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/status-code 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	Send a HTTP request for an entity that is not available in the service, check if the server returns `404 Not Found` or `410 Gone`. (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.2.1.4 Test: Sorting Query Option

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/request-data/sorting
Requirements	25. 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.2.1.5 Test: Client-driven Pagination Query Option

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/request-data/client-driven-pagination
Requirements	26. http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/top 27. http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/skip 28. 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.2.1.6 Test: Filter Query Option

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/request-data/filter-query-options
Requirements	29. http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/filter 30. http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/built-in-filter-operations 31. http://www.opengis.net/spec/iot_sensing/1.0/req/request-data/built-in-query-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.2.1.7 Test: Server-driven Pagination

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/request-data/server-driven-pagination

Requirements	32. 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 SensorThings API Create-Update-Delete Extension Tests

This section contains the conformance classes for the SensorThings API create-update-delete extension. That means a SensorThings API service that allows clients to create/update/delete entities needs to pass the conformance tests defined in this section.

A.3.1 Conformance class: SensorThings API Create-Update-Delete

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

Dependencies:

1. http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information
2. http://www.opengis.net/spec/iot_sensing/1.0/conf/thing
3. http://www.opengis.net/spec/iot_sensing/1.0/conf/location
4. http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location
5. http://www.opengis.net/spec/iot_sensing/1.0/conf/datastream
6. http://www.opengis.net/spec/iot_sensing/1.0/conf/sensor
7. http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property
8. http://www.opengis.net/spec/iot_sensing/1.0/conf/observation
9. http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest
10. http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path

A.3.1.1 Test: Sensing Entity Creation

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/create-update-delete/sensing-entity-creation
Requirements	33. http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/create-entity 34. http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/link-to-existing-entities 35. http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/deep-insert 36. http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/deep-insert-status-code 8. http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/historical-location-

	auto-creation
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 24 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 24 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 24 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. Create a `Location` for a `Thing` entity, check if the `Thing` has a `HistoricalLocation` created by the service according to the `Location` entity.

A.3.1.2 Test: Sensing Entity Update

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

Requirements	37. http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/update-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 10.3. 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.3.1.3 Test: Sensing Entity Deletion

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/create-update-delete/sensing-entity-deletion
Requirements	38. http://www.opengis.net/spec/iot_sensing/1.0/req/create-update-delete/delete-entity
Test purpose	Check if the service supports the deletion of entities as defined in Section 10.4.
Test method	Delete an entity instance, and check if the service responds as defined in Section 10.4.

A.4 SensorThings API Batch Request Extension Tests

This section contains the conformance classes for the SensorThings API batch request extension. That means a SensorThings API service that allows clients to send a single `HTTP` request that groups multiple requests needs to pass the conformance tests defined in this section.

A.4.1 Conformance class: SensorThings API Batch Request

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

Dependencies:

1. http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information
2. http://www.opengis.net/spec/iot_sensing/1.0/conf/thing
3. http://www.opengis.net/spec/iot_sensing/1.0/conf/location

4. http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location
5. http://www.opengis.net/spec/iot_sensing/1.0/conf/datastream
6. http://www.opengis.net/spec/iot_sensing/1.0/conf/sensor
7. http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property
8. http://www.opengis.net/spec/iot_sensing/1.0/conf/observation
9. http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest
10. http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path

A.4.1.1 Test: Batch Request

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/batch-request/batch-request
Requirements	39. http://www.opengis.net/spec/iot_sensing/1.0/req/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 SensorThings API `MultipleDatastream` Tests

This section contains the conformance classes for the SensorThings API `MultiDatastream` extension. That means a SensorThings API service that allows clients to group a collection of observations' results into an array (*i.e.*, a complex result type) needs to pass the conformance tests defined in this section.

A.5.1 Conformance class: SensorThings API `MultiDatastream`

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

Dependencies:

1. http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information
2. http://www.opengis.net/spec/iot_sensing/1.0/conf/thing
3. http://www.opengis.net/spec/iot_sensing/1.0/conf/location
4. http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location
5. http://www.opengis.net/spec/iot_sensing/1.0/conf/datastream
6. http://www.opengis.net/spec/iot_sensing/1.0/conf/sensor
7. http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property
8. http://www.opengis.net/spec/iot_sensing/1.0/conf/observation
9. http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest
10. http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path

A.5.1.1 Test: SensorThings API `MultiDatastream`

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/multi-datastream/multi-datastream-valid
Requirements	40. http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/properties 41. http://www.opengis.net/spec/iot_sensing/1.0/req/multi-datastream/relations 42. http://www.opengis.net/spec/iot_sensing/1.0/req/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 SensorThings API Data Array Extension

This section contains the conformance classe for the SensorThings API data array extension. That means a SensorThings API service that allows clients to request the compact data array encoding defined in this specification needs to pass the conformance tests defined in this section.

A.6.1 Conformance class: SensorThings API Data Array

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

Dependencies:

1. http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information
2. http://www.opengis.net/spec/iot_sensing/1.0/conf/thing
3. http://www.opengis.net/spec/iot_sensing/1.0/conf/location
4. http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location
5. http://www.opengis.net/spec/iot_sensing/1.0/conf/datastream
6. http://www.opengis.net/spec/iot_sensing/1.0/conf/sensor
7. http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property
8. http://www.opengis.net/spec/iot_sensing/1.0/conf/observation
9. http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest
10. http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path

A.6.1.1 Test: SensorThings API Sensing Data Array

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/data-array/data-array-valid
Requirements	43. 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 SensorThings API `Observation` Creation via MQTT Extension Tests

This section contains the conformance class for the SensorThings API `Observation` creation extension. That means a SensorThings API service that allows clients to create `Observations` via MQTT needs to pass the conformance tests defined in this section.

A.7.1 Conformance class: SensorThings API `Observation` Creation via MQTT

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/create-observations-via-mqtt

Dependencies:

1. http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information
2. http://www.opengis.net/spec/iot_sensing/1.0/conf/thing
3. http://www.opengis.net/spec/iot_sensing/1.0/conf/location
4. http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location
5. http://www.opengis.net/spec/iot_sensing/1.0/conf/datastream
6. http://www.opengis.net/spec/iot_sensing/1.0/conf/sensor
7. http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property
8. http://www.opengis.net/spec/iot_sensing/1.0/conf/observation
9. http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest
10. http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path
11. http://www.opengis.net/spec/iot_sensing/1.0/conf/create-update-delete

A.7.1.1 Test: SensorThings API `Observation` Creation via MQTT

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/create-observations-via-mqtt/observation-creation
Requirements	44. http://www.opengis.net/spec/iot_sensing/1.0/req/create-observations-via-mqtt/observation-creation
Test purpose	Check if the service supports the creation and update of entities via MQTT as defined in Section 14.1.
Test method	Create an `Observation` entity instance containing binding information for navigation

	properties using `MQTT Publish`, check if the server responds as defined in Section 14.1.

A.8 SensorThings API Receiving Updates via MQTT Extension Tests

This section contains the conformance class for the SensorThings API receiving updates extension. That means a SensorThings API service that allows clients to receive notifications regarding updates of entities via MQTT needs to pass the conformance tests defined in this section.

A.8.1 Conformance class: SensorThings API Receiving Updates via MQTT

Conformance class id: http://www.opengis.net/spec/iot_sensing/1.0/conf/receive-updates-via-mqtt

Dependencies:

1. http://www.opengis.net/spec/iot_sensing/1.0/conf/entity-control-information
2. http://www.opengis.net/spec/iot_sensing/1.0/conf/thing
3. http://www.opengis.net/spec/iot_sensing/1.0/conf/location
4. http://www.opengis.net/spec/iot_sensing/1.0/conf/historical-location
5. http://www.opengis.net/spec/iot_sensing/1.0/conf/datastream
6. http://www.opengis.net/spec/iot_sensing/1.0/conf/sensor
7. http://www.opengis.net/spec/iot_sensing/1.0/conf/observed-property
8. http://www.opengis.net/spec/iot_sensing/1.0/conf/observation
9. http://www.opengis.net/spec/iot_sensing/1.0/conf/feature-of-interest
10. http://www.opengis.net/spec/iot_sensing/1.0/conf/resource-path
11. http://www.opengis.net/spec/iot_sensing/1.0/conf/create-update-delete

A.8.1.1 Test: SensorThings API Receiving Updates via MQTT

Test id	http://www.opengis.net/spec/iot_sensing/1.0/conf/receive-updates-via-mqtt/receive-updates
Requirements	45. http://www.opengis.net/spec/iot_sensing/1.0/req/receive-updates-via-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. 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. 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 with `PATCH`. 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. 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 with `PATCH`. Check if a JSON object of the subscribed properties is received.

Annex B: Revision history

Date	Release	Author	Paragraph modified	Description
2015-01-24	0.9.0	Steve Liang	Many.	Updated according to the Calgary and Tokyo TC SWG meeting. Changed trajectory to HistoricalLocation.
2015-02-24	0.9	Carl Reed	Many.	Major edit and alignment with OGC document template.
2015-04-11	0.9	Steve Liang	Many.	Updated according to the Barcelona TC SWG meeting. Changed ComplexDatastream to MultiDatastream.
2015-05-11	0.9.1	Steve Liang	Many.	Updated according to the OGC Architecture Board meeting on May 5th 2015. Added requirement classes.
2015-06-11	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.
2015-08-24	0.9.3	Steve Liang	Many.	Updated according to the RFC comments. Added requirement classes. Added dependencies. Added a section to describe the relationship between this specification and O&M as well as OData.
2015-10-26	1.0	Steve Liang	Many.	Updated according to the OGC Architecture Board meeting on October 12th 2015. Updated a section to explain the design decision of choosing the OData KVP encoding rather than OGC KVP encoding.

2016-05-19	1.0	Steve Liang	Many.	Updated according to comments received from the OGC TC approval vote and Washington DC TC SWG meeting.
2016-06-30	1.0	Scott Simmons	All	Editing for final publication

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

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

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