OGC® DOCUMENT: 23-019

External identifier of this OGC® document: http://www.opengis.net/doc/IS/SensorThingsAPI/2.0

Open Geospatial Consortium

OGC SENSORTHINGS API 2.0

STANDARD
Implementation

APPROVED

Version: 2.0
Submission Date: 2026-05-26
Approval Date: 2026-09-14
Publication Date: 2026-11-04
Editor: Hylke van der Schaaf

Notice: This document is an OGC Member approved international standard. This document is available on a royalty free, non-discriminatory basis. Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights of which they are aware and to provide supporting documentation.

License Agreement

Use of this document is subject to the license agreement at https://www.ogc.org/license

Suggested additions, changes and comments on this document are welcome and encouraged. Such suggestions may be submitted using the online change request form on OGC web site: http://ogc.standardstracker.org/

Copyright notice

Copyright © 2026 Open Geospatial Consortium
To obtain additional rights of use, visit https://www.ogc.org/legal

Note

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.

CONTENTS

I. ABSTRACT vii

II. KEYWORDS vii

III. PREFACE viii

IV. SECURITY CONSIDERATIONS ix

IV.A. General Considerations ix

IV.A.1. Authentication ix

IV.A.2. Authorization x

IV.A.3. Integrity and Confidentiality x

IV.A.4. Availability x

IV.A.5. Data Assurance x

IV.A.6. Client Implementation x

IV.B. Considerations for the HTTP Interface xi

IV.B.1. Access Control aspects xi

IV.B.2. Confidentiality aspects xi

IV.B.3. Synchronous aspects xii

IV.C. Considerations for the MQTT Interface xii

V. SUBMITTING ORGANIZATIONS xiii

VI. SUBMITTERS xiii

1. SCOPE 2

2. CONFORMANCE 4

3. NORMATIVE REFERENCES 6

4. TERMS AND DEFINITIONS 9

5. ABBREVIATED TERMS 14

6. CONVENTIONS 17

6.1. Introduction 17

6.2. Identifiers 17

7. OGC SENSORTHINGS API OVERVIEW 19

7.1. What is the OGC SensorThings API 19

7.2. Who should use the OGC SensorThings API 19
7.2.1. Data Providers 20
7.2.2. Data Users 20
7.2.3. Potential Use Cases 20
7.3. Benefits of the OGC SensorThings API 21
7.4. OGC SensorThings API Overview 22
7.5. Relation to ISO/OGC Observations, Measurements and Samples 23
7.6. Relation to OASIS-OData 24
7.7. Changes from Version 1.1 24
7.7.1. Changes to the document structure 25
7.7.2. Changes to the API 25
7.7.3. Changes to the Data Model 25

8. THE OGC SENSORTHINGS API CORE DATA MODEL 29
8.1. Introduction 29
8.2. Recurring attributes 31
8.3. Thing 31
8.4. Location 33
8.5. HistoricalLocation 37
8.6. Datastream 38
8.7. Sensor 45
8.8. ObservedProperty 47
8.9. Observation 48
8.10. Feature 50
8.11. FeatureType 52

9. ABSTRACT REST API 55
9.1. Introduction 55
9.2. OData Entity Model 55
9.3. Type Mappings 57
9.3.1. Simple Types 58
9.3.2. Complex Type TM_Interval 58
9.3.3. Complex Type TM_Object 59
9.3.4. Complex Type JSON_Object 59
9.4. OData Common Schema Definition Language 59
9.5. URL Encoding for constants 59
9.6. URL Patterns 60
9.6.1. Introduction 60
9.6.2. Service Document 61
9.6.3. Service Metadata Document 61
9.6.4. Entity Set 61
9.6.5. Entity 61
9.6.6. Entity Attribute 62
9.6.7. Entity Attribute raw value ($value) 62
9.6.8. Related Entity 62
9.6.9. Related EntitySet 63
9.6.10. Association link ($ref) 63

9.6.11. Nested paths 64
9.7. Request Types 64
9.8. Additional request parameters 64
9.9. Requesting Data 65
9.9.1. Introduction 65
9.9.2. Read Paths 66
9.9.3. Request Query Options 71
9.10. Modifying Data 86
9.10.1. Introduction 86
9.10.2. Creating Entities 88
9.10.3. Deep Insert 88
9.10.4. Updating Entities 90
9.10.5. Replacing Entities 90
9.10.6. Deep Update 91
9.10.7. JSON-Patch 92
9.10.8. Return Value 92
9.10.9. Modifying Relations 93
9.10.10. Deleting Entities 94
9.10.11. Deleting Entities by Filter 95

10. HTTP INTERFACE BINDINGS 97
10.1. Introduction 97
10.2. HTTP Binding Advertisement 97
10.3. Request / Response 98
10.4. WebSub Topic Discovery 99
10.4.1. Link Header 'hub' 100
10.4.2. Link Header 'self' 100
10.4.3. Link Header 'help' 101
10.4.4. ODATA query options support 101
10.4.5. ODATA support advertisement 102
10.4.6. ODATA Blacklisting 102
10.4.7. Topics Blacklisting 103

11. MQTT INTERFACE BINDINGS 107
11.1. Introduction 107
11.2. MQTT Binding Advertisement 107
11.3. Request / Response 108
11.4. Publish / Subscribe 111
11.4.1. General 111
11.4.2. $select 112
11.4.3. $expand 113
11.4.4. $filter 113
11.5. Simple Create 113

12. OGC API — COMMON INTERFACE BINDINGS 116
12.1. General 116
12.2. Landing Page 116

12.3. OpenAPI Definition 117

13. MEDIA TYPES FOR ANY DATA ENCODING(S) 119

ANNEX A CONFORMANCE CLASS ABSTRACT TEST SUITE (NORMATIVE) 121
A.1. Conformance Class: HTTP 121
A.2. Conformance Class: WebSub 122
A.3. Conformance Class: MQTT 128
A.4. Conformance Class: Data Model 132
A.5. Conformance Class: Abstract API 136

ANNEX B SHORT CSDL EXAMPLE WITH COMMENTS 149

ANNEX C EXAMPLE SENSORTHINGS API CORE CSDL 152

ANNEX D REVISION HISTORY 161

I ABSTRACT

The OGC SensorThings API provides an open, geospatial-enabled and unified way to interconnect sensor-data producing devices, data, and applications over the Internet. It contains a core data model, an abstract REST API for accessing data, and bindings for this API for HTTP and MQTT.

The data model is an implementation of the core concepts of the Observations, Measurements and Samples OGC/ISO specification. The API allows for creating, reading, updating and deleting all parts of the data model. It also features extensive filtering and response-customization capabilities.

Even though the SensorThings API was initially designed for IoT, the previous versions have shown to be well suited for any use case involving time-series data.

II KEYWORDS

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

ogcdoc, OGC document, API, OData, openapi, html, MQTT, iot, internet of things, sensors, swe, sensor webs, sensor web enablement, sensor networks

PREFACE

The OGC SensorThings API provides an open, geospatial-enabled and unified way to interconnect sensor-data producing devices, data, and applications over the Internet. Version 2.0 brings updates to version 1.1, improving the data model, the API and the modularity of the standard itself.

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 SECURITY CONSIDERATIONS

Writing security considerations for implementing the OGC SensorThings API (STA) 2.0 is challenging. This is because the range of uses of a STA implementation cannot be always known. The use cases for an implementation could vary from automated tools for analytical purposes that fetch data (observations) to produce statistics, reports, etc. to Web-Mapping applications running in a Web Browser that interact with a STA service based on user interactions like pan, zoom in and zoom out. In other words, the use spans from procedural to user driven interactions. Then there are also simple sensor systems that publish data via MQTT as well as observation procedures that subscribe to updates of data. All this must be taken into consideration when speaking of security requirements.

Given that the STA Standard provides two different interfaces — HTTP and MQTT — and the different use cases, this section is separated into three parts: General considerations, considerations for an HTTP interface, and considerations for an MQTT interface.

IV.A. General Considerations

Typical security considerations should address the following topics:

- authentication
- authorization
- integrity
- confidentiality
- availability
- data assurance

IV.A.1. Authentication

The STA Standard specifies a conformance class that allows limiting access to the HTTP and MQTT interface to identified actors only. In particular, the HTTP status code 401 can be used to signal to the caller that authentication such as an API-key, is required.

IV.A.2. Authorization

The STA Standard does not define how authorization can be applied. Any implementation of the STA Standard may implement authorization. The STA Standard defines the use of HTTP status code 403 to signal to the caller that execution of the request is forbidden.

IV.A.3. Integrity and Confidentiality

The STA Standard does not define how to achieve integrity and confidentiality of the stored data. This means that without implementing authorization, any caller can overwrite existing information or view potentially sensitive data.

IV.A.4. Availability

Availability is a key aspect for any service based infrastructure. To ensure availability, any implementation needs to understand the resource allocation for any type of request. The STA HTTP interface is based on ODATA with an extension to define options for geospatial data. To ensure high availability, complex ODATA queries involving complex geospatial conditions may require a high load of resources like CPU and main memory. Any implementation should carefully analyze the request query parameters (the ODATA options) before executing the request. To prevent (perhaps accidental) requests including complex ODATA queries, certain types of requests should require an API-Key.

IV.A.5. Data Assurance

An implementation of the STA Standard may support the creation of data via the HTTP and MQTT interface. A client implementation should parse the uploaded data for scripting content. For example cross-site scripting attacks may get executed in any Web-Application that carelessly includes data fetched from a STA HTTP or MQTT interface into an HTML page. To arrange an effective execution of an attack, the attacker would first upload some JavaScript to include in the result of an observation (or description of another entity). For example `"; alert("xss attack!");//`[1]. If the Web-Application renders the data via `document.getElementById("content").innerHTML = <obervation.result>;` the attack gets executed.

IV.A.6. Client Implementation

A STA HTTP endpoint that supports Cross Origin Resource Sharing (CORS) can be accessed from Web-Applications (JavaScript implementations executing in a Web Browser). A simple Web-Application that fetches observations from such an endpoint to display a time chart potentially generates a high load on the server(s) that host the endpoint. Typical Web-Mapping

[1] https://stackoverflow.com/questions/30661497/xss-prevention-and-innerhtml

libraries like OpenLayers and Leaflet split the canvas into tiles and make a request for each tile in parallel. With fast zoom in and out and panning the area of interest changes and the STA HTTP endpoint receives many requests basically for nothing, because fast zoom and pan means that the user is navigating to an area of interest. If for each of the zoom and pan steps, the STA HTTP endpoint is hit, a high server load is produced for nothing.

Another aspect regarding client development is caching: A Web Browser supports caching of static data such as images. However, the result from a STA endpoint is a dynamically created JSON response. This is typically not automatically cached in the Web Browser. To avoid useless interactions with the endpoint, a Web-Application should implement efficient caching.

A STA deployment should control, via the CORS configuration, which Web-Application to trust.

IV.B. Considerations for the HTTP Interface

Among the typical security considerations that exist for HTTP interfaces, the STA Standard specifies requirements for the use of ODATA query options.

IV.B.1. Access Control aspects

An implementation of a STA HTTP interface is very complex in terms of arranging access control. The complexity of the HTTP interface and the overwhelming flexibility of the ODATA options suggest that Access Control should be integrated into the STA implementation. Even simple access control scenarios that allow or disallow access to an entity type may have side effects to other requests where such entity type is used internally. For example, it may be possible that a particular entity type is not accessible directly via the HTTP interface but is allowed to participate in a `$filter` operation. Also, the different "path to the same resources" challenge indicate that it would be not a good idea to implement access control using the proxy pattern. For example, the combination of `$filter` and `$expand` is already difficult to be implemented into a proxy: `.../Observations?$expand=Datastream($fiter=id eq 1)` and `/Datastreams(1)/Observations` and `/FeaturesOfInterest/Observations?$expand=Datastream($fiter=id eq 1)` return similar data.

Implementing access control at the database level seems not only the best approach, it also seems to provide the best performance.

IV.B.2. Confidentiality aspects

When not all data is public accessible, a specific type of access control needs to be included in the application. Similar to the access control example, the granularity level suggests implementing the confidentiality control directly in the database and not in a proxy.

IV.B.3. Synchronous aspects

The STA HTTP interface operates synchronously. That means that a caller is waiting for the response on the established socket connection. In the case where the implementation takes too long to start sending the first byte to the caller, the risk of a socket timeout arises. To avoid the useless allocation of server resources, any connection timeout should be avoided. To ensure that an implementation can be implemented “streaming ready”, the STA Standard only defines streaming ready response types. For example, one typical format that is not streaming ready is a GeoPackage. Any SQLite database must be fully created at server-side before the first byte of the response can be sent to the caller.

An implementation of the STA Standard should consider streaming records from the database to the HTTP interface as they are loaded from the database instead of loading the full database response into main memory before sending it to the caller. This approach prevents high resource allocation and thereby improves service availability.

IV.C. Considerations for the MQTT Interface

A STA service implementation may expose an MQTT interface to the world. Similar to the HTTP interface, the MQTT interface supports the creation of data via MQTT publish but also receives notifications of changes for registered MQTT topics.

The authentication mechanisms specified in the MQTT Standard can be used in an implementation of the MQTT interface of the STA Standard. This enables an implementation to restrict the subscription to “chatty” topics for some client identities. Any openness of the MQTT interface may be used by adversaries to modify data, read sensitive data or even execute DoS attacks by submitting too many subscriptions for “chatty” topics like `Observations` or bulk publishing of bogus data.

The publish capability for the MQTT interface should be protected via an authentication mechanism to disallow anonymous users (or bots) to create bogus data or flush the service instance with bulk data. Likewise, the subscription to chatty topics or to topics with complex ODATA options should only be allowed for authenticated callers.

V SUBMITTING ORGANIZATIONS

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

- Fraunhofer-Gesellschaft, Germany
- University of Calgary, Canada
- SensorUp Inc., Canada
- Keys, USA
- DataCove e.U., Austria
- Secure Dimensions, Germany
- CREAF, Spain
- BRGM, France

VI SUBMITTERS

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

- Hylke van der Schaaf *(editor)* — Fraunhofer-Gesellschaft
- Steve Liang — SensorUp Inc.
- Marcus Alzona — Keys.com
- Kathi Schleidt — Datacove e.U.
- Andreas Matheus — Secure Dimensions GmbH
- Humaid Imran Kidwai — University of Calgary
- Joan Maso Pau — Universitat Autònoma de Barcelona (CREAF)
- Sylvain Grellet — BRGM
- Mickael Beaufils — BRGM

The submitters would like to acknowledge the following organizations and projects as important supporters and contributors to this version of the specification:

- Bureau de Recherches Géologiques et Minières (BRGM)

- Datacove e.U.
- Fraunhofer IOSB
- OGC / WMO Water Quality IE
- OGC GeoTech IE
- OneWater Data French Research project (ANR-22-PEXO-0009)
- Secure Dimensions GmbH
- SensorUP Inc.
- Universitat Autònoma de Barcelona (CREAF)
- University of Calgary
- Water4All Partnership (European Union's Horizon Europe Programme under Grant Agreement 101060874)

1 SCOPE

1 SCOPE

The OGC SensorThings API Version 2 provides an open Standard-based and geospatial-enabled framework to interconnect the Internet of Things devices, data, and applications over the web. It defines a data model, an abstract REST API for interacting with the data model, and HTTP and MQTT protocol bindings for the REST API. Furthermore, it provides support for push notifications for data changes over MQTT.

The data model, and the API can be extended by specifying requirements for additional protocol bindings or data encodings. Such extensions can be added to future revisions of the Standard, or as separate extension documents.

2 CONFORMANCE

2 CONFORMANCE

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

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

3 NORMATIVE REFERENCES

3 NORMATIVE REFERENCES

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

Steve Liang, Tania Khalafbeigi, Hylke van der Schaaf, Open Geospatial Consortium: OGC 18-088, *OGC SensorThings API Part 1: Sensing Version 1.1*. Open Geospatial Consortium (2021). http://www.opengis.net/doc/is/sensorthings/1.1.0.

Katharina Schleidt, Ilkka Rinne, Open Geospatial Consortium: OGC 20-082r4, *Topic 20 — Observations, measurements and samples*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/as/om/3.0.

International Organization for Standardization (committee): ISO 19156:2023, *Geographic information — Observations, measurements and samples*. International Organization for Standardization, Geneva (2023). https://www.iso.org/standard/82463.html.

John Herring, Open Geospatial Consortium: OGC 06-104r4, *OpenGIS Implementation Specification for Geographic information — Simple feature access — Part 2: SQL option*. Open Geospatial Consortium (2010). https://docs.ogc.org/is/06-104r4/06-104r4/pdf.

Armin Haller, Krzysztof Janowicz, Simon Cox, Danh Le Phuoc, Kerry Taylor, Maxime Lefrançois, Open Geospatial Consortium: OGC 16-079, *Semantic Sensor Network Ontology*. Open Geospatial Consortium (2019). https://www.w3.org/TR/vocab-ssn/.

Charles Heazel, Open Geospatial Consortium: OGC 19-072, *OGC API — Common — Part 1: Core*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/is/ogcapi-common-1/1.0.0.

Alexandre Robin, Open Geospatial Consortium: OGC 24-014, *OGC SWE Common Data Model Encoding Standard*. Open Geospatial Consortium (2025). http://www.opengis.net/doc/IS/SWE/3.0.

Rescorla E, Internet Engineering Task Force: IETF RFC 2818, *HTTP Over TLS*. RFC Publisher (2000). https://www.rfc-editor.org/info/rfc2818.

Nottingham M: IETF RFC 5988, *Web Linking*. RFC Publisher (2010). https://www.rfc-editor.org/info/rfc5988.

Internet Engineering Task Force (committee): IETF RFC 6902, *JavaScript Object Notation (JSON) Patch*. RFC Publisher (2013). https://www.rfc-editor.org/info/rfc6902.

Snell J: IETF RFC 7240, *Prefer Header for HTTP*. RFC Publisher (2014). https://www.rfc-editor.org/info/rfc7240.

OASIS (committee): OASIS OData-Part1, *OData Version 4.01. Part 1: Protocol*. OASIS (2020). http://docs.oasis-open.org/odata/odata/v4.01/os/part1-protocol/odata-v4.01-os-part1-protocol.html.

OASIS (committee): OASIS OData-Part2, *OData Version 4.01. Part 2: URL Conventions*. OASIS (2020). http://docs.oasis-open.org/odata/odata/v4.01/os/part2-url-conventions/odata-v4.01-os-part2-url-conventions.html.

OASIS (committee): OASIS OData-JSON-Format-v4.01, *OData JSON Format v4.01*. OASIS (2020). http://docs.oasis-open.org/odata/odata-json-format/v4.01/os/odata-json-format-v4.01-os.html.

OASIS (committee): OASIS OData-CSDL-JSON-v4.01, *OData Common Schema Definition Language (CSDL) JSON Representation Version 4.01*. OASIS (2020). http://docs.oasis-open.org/odata/odata-csdl-json/v4.01/os/odata-csdl-json-v4.01-os.html.

OASIS (committee): OASIS mqtt-v5.0, *MQTT Version 5.0*. OASIS (2019). http://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

Arliss Whiteside, Jim Greenwood, Open Geospatial Consortium: OGC 06-121r9, *OGC Web Service Common Implementation Specification*. Open Geospatial Consortium (2010). https://docs.ogc.org/is/06-121r9/06-121r9/pdf.

World Wide Web Consortium (committee): W3C REC-websub-20180123, *WebSub*. World Wide Web Consortium (2018). https://www.w3.org/TR/2018/REC-websub-20180123/.

ITU-T Y.2060: *Overview of the Internet of Things*, 2012. Available Online: https://www.itu.int/rec/t-rec-y.2060-201206-i

Open API Initiative: OpenAPI Specification. Available from https://spec.openapis.org/oas/.

4 TERMS AND DEFINITIONS

4 TERMS AND DEFINITIONS

This document uses the terms defined in OGC Policy Directive 49, 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 document and OGC documents do not use the equivalent phrases in the ISO/IEC Directives, Part 2.

This document also uses terms defined in the OGC Standard for Modular specifications (OGC 08-131r3), also known as the ‘ModSpec’. The definitions of terms such as standard, specification, requirement, and conformance test are provided in the ModSpec.

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

4.1. Annotation

Additional information attached to model or instance elements. Annotations can be identified by the `@` character.

4.2. Attribute

Named element that is part of a structured type (Entity Type or Complex Type).

4.3. Complex Type

Named, structured, keyless, element consisting of a set of named attributes.

4.4. EntityId

A globally unique identifier for an Entity. If present in Entity representations, the EntityId is specified in the `@id` annotation.

4.5. Entity

instance of an `Entity Type`. An Entity can have only one Entity Type.

4.6. Entity Set

A named collection of Entities of the same `Entity Type`. Each Entity is uniquely identified among other Entities in the same Entity Set, by its key. Entity Sets are the entry points into the data model.

4.7. Entity Type

Named, structured, keyed element that defines the attributes and relations of Entities.

4.8. (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.

[**SOURCE:** ITU-T-Y.2060]

4.9. Measurement

A set of operations having the object of determining the value of a quantity.

[**SOURCE:** OGC 20-082r4; ISO 19156:2023]

4.10. Observation

Act of measuring or otherwise determining the value of a property.

[**SOURCE:** OGC 20-082r4; ISO 19156:2023]

4.11. Observation Result

Estimate of the value of a property determined through a known observation procedure.

[**SOURCE:** OGC 20-082r4; ISO 19156:2023]

4.12. Primitive Type

Unnamed, unstructured type, such as String, Integer or DateTime. An attribute with a Primitive Type is called a primitive attribute.

4.13. Relation

Named connection between Entities.

4.14. STA API

The API part of the OGC SensorThings API standard. Concrete bindings are provided for HTTP and MQTT.

4.15. STA data model

The data model part of the OGC SensorThings API standard.

4.16. STA endpoint

A point of connection for a client. For example, the HTTP endpoint of a STA service is available at a given domain and path and can be accessed using the HTTP protocol.

4.17. STA implementation

A piece of software that implements the OGC SensorThings API standard, as a client or server.

4.18. STA service

A deployed STA implementation. A single service may expose multiple endpoints.

4.19. STA Standard

The OGC SensorThings API Standard (this document).

5 ABBREVIATED TERMS

5 ABBREVIATED TERMS

API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
CRUD	Create, Read, Update, and Delete
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IoT	Internet of Things
ISO	International Organization for Standardization
JSON	JavaScript Object Notation
O&M	Observations and Measurements
OData	the Open Data Protocol
OGC	Open Geospatial Consortium
OMS	Observations, Measurements and Samples
PDF	Portable Document Format
SensorML	Sensor Model Language
STA	OGC SensorThings API
SWE	Sensor Web Enablement
UCUM	Unified Code for Units of Measure
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WoT	Web of Things
XML	eXtensible Markup Language

YAML	YAML Ain't Markup Language

6 CONVENTIONS

6 CONVENTIONS

6.1. Introduction

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.

6.2. Identifiers

The normative provisions in this standard are denoted by the URI

`http://www.opengis.net/spec/sensorthings/2.0`

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

7

OGC SENSORTHINGS API OVERVIEW

7 OGC SENSORTHINGS API OVERVIEW

7.1. What is the OGC SensorThings API

The OGC SensorThings API (STA) Standard defines an open and unified framework to interconnect IoT sensing devices, data, and applications over the Web. The STA Standard is based on the OGC Observations, Measurements and Samples (OMS) Standard, also formalized as ISO 19156. This Standard provides a conceptual model for capturing all relevant information pertaining to observed or measured data. The STA Standard addresses the syntactic and semantic interoperability of the Internet of Things, complementing the existing IoT networking protocols such CoAP, MQTT, HTTP, 6LowPAN. While the above-mentioned IoT networking protocols are addressing the ability for different IoT systems to exchange information, the OGC SensorThings API Standard addresses the ability for different IoT systems to use and understand the exchanged information. This capability is due to the inclusion of OMS concepts. The OGC SensorThings API Standard is structured to support easy integration into existing Spatial Data Infrastructures or Geographic Information Systems.

While originally conceived within the IoT world, the OGC SensorThings API Standard has been shown to be applicable to a wide range of use cases far beyond IoT. While the Standard carries Sensors in its name, its applicability ranges far beyond traditional sensor environments. It has been successfully utilized for *ex situ* measurements performed in the lab as well as field surveys, where humans take on the role of the Sensor. While excellent for providing real-time data directly from sensors, implementations of the OGC SensorThings API have also successfully been deployed “on top of” legacy databases. Thus, the OGC SensorThings API Standard has evolved from its IoT roots to become a Standard for the provision of observational data, taking into account all relevant measurement metadata as foreseen under the OGC Observations, Measurements and Samples Standard.

7.2. Who should use the OGC SensorThings API

Use of the OGC SensorThings API falls into two distinct groups:

- Data providers, looking for a lightweight approach to exposing their available data; and
- Data users, looking for relevant data including essential measurement metadata.

7.2.1. Data Providers

Organizations that need web-based platforms to manage, store, share, and analyze observational data should use the OGC SensorThings API. The OGC SensorThings API Standard is designed to simplify and accelerate the development of modern location enabled digital monitoring and control applications. Sensor device manufacturers can also use OGC SensorThings API because implementation instances of STA can be embedded within various hardware and software platforms. This enables the various sensing devices to effortlessly connect with the STA Standard-compliant servers around the world.

7.2.2. Data Users

Application developers can implement the STA Standard to connect to various sensor devices and create innovative applications without worrying about the daunting heterogeneous protocols of the disparate vendor hardware including gateways and services. The rich measurement metadata available due to the integration of the OGC Observations, Measurements and Samples Standard enables users to understand the details of how the data was generated, essential for gaging if the data being provided is fit for purpose. The powerful query options defined in the STA Standard enable users to custom tailor their results, retrieving exactly the data they require for a specific purpose with no unnecessary overhead. This enables developers to focus on the domain aspects of their work instead of having to worry about data access issues.

7.2.3. Potential Use Cases

While the OGC SensorThings API Standard was initially designed with IoT applications in mind, it has proven valuable for many non-IoT use cases as well.

- Environmental Monitoring: Managing data from fixed monitoring stations for air quality, water quality, soil quality, biodiversity, weather, and climate observations that may not be considered traditional IoT devices.
- Scientific Research: Handling laboratory measurements, experimental data, and field observations where standardized data management and interoperability are crucial in a flexible manner. The core data model assures consistency while the standardized extension points enable integration of emerging concepts.
- Urban Planning, Smart Cities & Digital Twins: Urban planners & municipal corporations can use implementations of the STA Standard to monitor various environmental and infrastructure metrics, like noise levels, traffic, pollution, or green space usage, helping with sustainable city planning. Large buildings or campuses can have digital twins that track HVAC systems, lighting, occupancy, occupancy status and safety conditions. Cities are using implementations of the OGC SensorThings API Standard to manage their traffic systems in real-time. Implementations of the OGC SensorThings API Standard support efficient, standardized data collection from these disparate systems.

- Historical Data Archives: Providing standardized access to historical observation datasets stored in databases. The same mechanisms originally designed for IoT devices apply seamlessly to legacy databases.
- Citizen Science: Supporting data collection and management from volunteer observers and manual measurements in the framework of citizen science initiatives. Extensions to the STA Standard have been created to better support user management in such scenarios.
- Regulatory Compliance: Managing environmental compliance for monitoring data from both automated and manual sampling programs. While current reporting systems mostly rely on validated data provided as static datasets, approaches for direct provision of near-real-time data are increasingly being established, for which the OGC SensorThings API is uniquely suited.

In summary, implementations of the OGC SensorThings API Standard are transforming the numerous disjointed networked sensor systems into a fully connected platform where complex tasks can be synchronized and performed. The flexible data model and powerful query capabilities make the OGC SensorThings API suitable for any application involving many types of observations over space and time, regardless of whether networked sensor devices are involved.

7.3. 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 Web of Things (WoT) and IoT, the OGC SensorThings API offers the following key benefits.

1. **Interoperability and Integration**
    - Enables seamless integration between different sensor systems, platforms, and applications
    - Reduces vendor lock-in by providing a standardized interface
    - Facilitates data exchange across organizational boundaries and domains
2. **Cost and Time Efficiency**
    - Lowers development costs through standardized implementation patterns
    - Reduces time-to-market for IoT solutions

- Minimizes the need for custom interface development
- Simplifies maintenance and updates across the system

3. **Scalability and Flexibility**
   - Supports both small-scale deployments and large enterprise solutions
   - Adapts to various use cases from environmental monitoring to smart cities
   - Enables easy addition of new sensors and data sources
   - Supports both real-time and historical data management

4. **Data Quality and Governance**
   - Provides consistent metadata management
   - Ensures data provenance through standardized observation tracking
   - Supports regulatory compliance through structured data handling
   - Enables better data discovery and reuse

5. **Modern Web-GIS Support**
   - Aligns with REST architectural principles
   - Supports cloud-native deployments
   - Enables edge computing scenarios

6. **Enhanced Analytics and Innovation**
   - Simplifies data integration for analytics and AI/ML applications
   - Enables standardized access to historical datasets
   - Supports real-time monitoring and alerting
   - Facilitates development of innovative applications and services

These benefits make the OGC SensorThings API particularly valuable in today's rapidly evolving digital landscape, where organizations increasingly rely on sensor data for decision-making and automation.

7.4. OGC SensorThings API Overview

The OGC SensorThings API core specification consists of several parts:

1. The Core data model (Clause 8),
2. The Abstract API (Clause 9),
3. The HTTP Interface Bindings (Clause 10),
4. The MQTT Interface Bindings (Clause 11) and
5. The OGC API — Common Interface Bindings (Clause 12)

The core data model describes with types of entities can be exchanged through the API endpoints, and how these entity types relate to each other. The STA data model is designed based on the OGC/ISO Observation, Measurements and Samples (OMS) model (OGC 20-082r4 and ISO 19156:2023). The data model can be extended, with additional entities and relations, by other specifications or standards.

The abstract definition of the STA-API describes, at an abstract level, how users can interact with the data model. The abstract level describes the actions a user can make, the paths on which a user can make these actions, and the default formatting of data.

The HTTP and MQTT interface bindings describe how the abstract API can be mapped to the physical HTTP and/or MQTT level.

7.5. Relation to ISO/OGC Observations, Measurements and Samples

Managing and retrieving observations and metadata from IoT sensor systems is one of the most common use cases. ISO/OGC Observations, Measurements and Samples(OMS) defines conceptual models for the exchange of information describing observation acts, their results, and the (spatial) feature on which the observation was made. In addition, a wealth of observational metadata is covered, allowing a user to discover methodologies and devices utilized in obtaining the observational data. OMS is Topic 20 of the OGC Abstract Specification, that provides the conceptual foundation for OGC specification development activities related to observations, measurements and samples. As a result, OGC SensorThings API is designed based on the OMS model.

The OGC SensorThings API Standard defines nine entities for the IoT sensing applications and several additional entities in various extensions. In Table 1, each entity type is listed along with

its relationship with OMS. This document uses the term of Sensor to describe the Observer that is used in generating an Observation, instead of using the OMS term of Observer.

Table 1 — OGC SensorThings API entities and equivalent concepts in the Observations, Measurements and Samples standard

STA ENTITIES	OMS CONCEPTS
Thing	Host
Datastream	ObservationCollection, ObservingCapability
Sensor	Observer
Observation	Observation
ObservedProperty	Observed Property
Feature	Feature

The OMS classes Deployment, Observing Procedure, Sample, Sampling, Sampling Procedure, Preparation Procedure and Preparation Step will be covered in a future extension to the OGC SensorThings API Standard.

7.6. Relation to OASIS-OData

OData is an API standard for exchanging relational data. It allows for the definition of a consistent REST API on any relational data model. OData specifies how clients can inspect the data model and how they can perform Create, Read, Update and Delete actions. OData comes with a very powerful query language that allows users to compose the response to queries such that only a minimal number of queries is required to fetch needed data, regardless of the use case of the client. The OData specification also defines filtering mechanisms that support filtering across relations. OData uses JSON-encoding by default, and specifies generic rules for encoding relational data models in JSON.

The OGC SensorThings API v2 interface is not an OData interface and does not claim to be an OData service. It specifies a subset of the OData 4.01 specification and extends it at the same time with certain features optimized for accessing sensor data. A STA Server implementation can implement the full OData specification but does not have to.

7.7. Changes from Version 1.1

From the OGC SensorThings API version 1.1 (OGC 18-088) to 2.0 changes have been made to both the data model, the specification of the STA API and the document structure.

7.7.1. Changes to the document structure

The STA data model, the specification of the STA API, and the protocol bindings are now independent sections that can be used independently. For instance, since MQTT 5 introduced a request/response pattern, it is now possible to use an implementation of the complete STA API over MQTT without using the HTTP bindings to discover content. Several extensions that were defined directly in the version 1.1 document have now been moved to a separate extension document that will be published at a later date. These extensions include, among others, the Batch-Request extension and the DataArrays extension.

7.7.2. Changes to the API

The STA API definition is now aligned with OData version 4.01, removing incompatibilities. The most important change in the API is that the OData `$metadata` page has been added to the STA API, allowing a client to see the exact data model that is available on a server, including any data model extensions (Clause 9.6.3).

A second important change is the behavior when updating an Entity (for example using HTTP PATCH or PUT) and specifying a list of related Entities in a navigation property. In version 1.1 this would add the specified entities to the collection, while in version 2.0 (as in OData 4.01) this replaces the list of related entities with the specified list. (Clause 9.10.6)

The STA API now also has options to directly modify relations between Entities as described in Clause 9.10.9.

7.7.3. Changes to the Data Model

The changes to the data model and its JSON representation are summarised in Table 2 and Figure 1. There are two important changes in the data model.

The first relates to how a Datastream specifies its Observation type. Datastreams now contain two attributes that describe the structure and encoding of the result fields in their Observations. The `resultType` attribute contains a SWE-Common JSON description of the result structure, while the `resultEncoding` contains the SWE-Common encoding of this result structure. The unitOfMeasure is part of the SWE-Common structure description found in `resultType`. Therefore, the `unitOfMeasurement` attribute has been removed. Since SWE-Common can be used to describe any structure, also one covering multiple ObservedProperties, the MultiDatastream extension of the OGC SensorThings API version 1.1 is obsolete.

The second major change is in how FeatureOfInterest is modeled. This is due to an update made to OMS (OGC 20-082r4) that now defines two roles a Feature can have with the act of Observing:

- proximateFeatureOfInterest: entity that is directly of interest in the act of observing; and
- ultimateFeatureOfInterest: entity that is ultimately of interest in the act of observing.

Based on this, the Entity Type `FeatureOfInterest` has been renamed to `Feature`, since the "of Interest" part of the name has nothing to do with the Feature directly, but only with the role it has in the process of being observed. The "of Interest" is moved to the name of the relation an Observation/Datastream has with the Feature. Furthermore, a Feature can now also be linked to Datastreams, as either proximate- or ultimateFeatureOfInterest, instead of just to Observations as FeatureOfInterest. This change allows for a much more detailed modelling of domain features and their relations to the Observations made on them.

Table 2 — Changes in the OGC SensorThings API 2.0 data model compared to version 1.1

ENTITY / ATTRIBUTES	CHANGES
All	• `@iot.id` is renamed to `id` • `@iot.selfLink` is renamed to `@id`, to match OData 4.01 • All `@iot.navigationLink` are renamed to `@navigationLink`, to match OData 4.01 • `@iot.nextLink` is renamed to `@nextLink` to match OData 4.01
Sensor	• `description` attribute is now optional and not mandatory • `definition` attribute added
Thing	• `description` attribute is now optional and not mandatory • `definition` attribute added
Location	• `description` attribute is now optional and not mandatory • For a `Thing` having multiple `Locations`, these `Locations` MAY be in same encoding Types OR the encodingTypes MAY be in different spaces (e.g., one encodingType in Geometrical space and one encodingType in Topological space). • `definition` attribute added
Datastream	• `description` attribute is now optional and not mandatory • `resultType` replaces the `observationType` attribute, this eliminates the `MultiDatastream` entity • `unitOfMeasurement` SHALL be embedded within the `resultType` attribute and does not exist as an independent attribute within the `Datastream` entity • A `Datastream` can link to multiple `ObservedProperties` which was only possible with the `MultiDatastream` entity earlier. The SWE-Common based `resultType` and `resultEncoding` attributes eliminate the need for having a separate MultiDatastream entity • A `Datastream` can now be linked to the `Feature` it observes as two optional links between `Datastream` and `Feature` are introduced, named `UltimateFeaturesOfInterest` and `ProximateFeatureOfInterest`. • `definition` attribute added
Observed Property	`description` attribute is now optional and not mandatory
Observation	• `properties` replaces the `parameters` attribute.

ENTITY / ATTRIBUTES	CHANGES
	• An `Observation` may or may not link to any `Feature` as ProximateFeatureOfInterest in contrast to the mandatory link between `Observation` and `FeatureOfInterest` from v1.x. • `resultTime` is now optional and can be left out completely, instead of defaulting to a `null` value.
Feature	• The `Feature` entity replaces the `FeatureOfInterest` entity from 1.x as it now takes the role of ultimate or proximate feature of interest depending upon the context and links with Observation and Datastream entities • `definition` attribute added
FeatureType	The FeatureType entity is added, and makes it easier to handle type information for Features
TM_Object & TM_Interval	Time attributes that can contain an Interval, and thus can have a start and end time, are now modelled as complex attributes with an explicit `start` and `end` sub-attribute. Their serialization no longer uses the ISO-String notation with two times separated with a `/`.

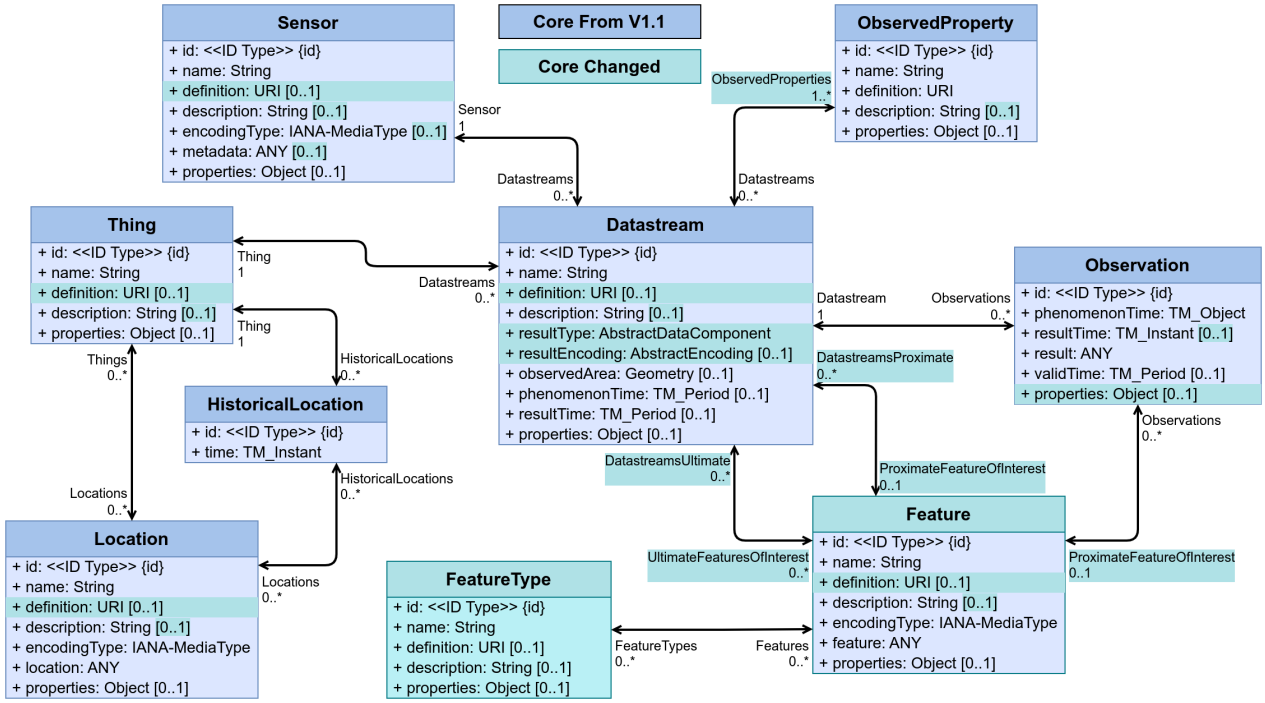


Figure 1 — Sensing Core Changes

8

THE OGC SENSORTHINGS API CORE DATA MODEL

8 THE OGC SENSORTHINGS API CORE DATA MODEL

8.1. Introduction

The core data model of the STA Standard is based on the OGC/ISO Observation, Measurements and Samples (OMS) model (OGC 20-082r4 and ISO 19156:2023). An Observation is modeled as an act that produces a result whose value is an estimate of a property of a given Feature. An Observation instance is characterized by its event time (e.g., resultTime and phenonmenonTime), Feature, ObservedProperty, and the observer that made the Observation (often a Sensor).

The STA Standard also models Things. A Thing draws from the same concept as a Host in OMS where a Host is defined as a collection of Observers (defined as Sensor in the STA Standard). Formally, 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, Sensor and optionally by Feature and ObservingProcedure.

An Observation is an event performed by a Sensor that produces a result whose value is an estimate of an ObservedProperty of any given Feature which may be a proximate or ultimate feature of interest. Details of each above described entity are provided in the sections below.

All data model requirements are grouped in the following requirements class.

REQUIREMENTS CLASS 1: CORE DATA MODEL	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`
OBLIGATION	requirement
TARGET TYPE	Data Model
CONFORMANCE CLASS	Conformance class A.4: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/datamodel/core`

REQUIREMENTS CLASS 1: CORE DATA MODEL	
NORMATIVE STATEMENTS	Requirement 1: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/thing Requirement 2: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/location Requirement 3: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/historical-location Requirement 4: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/datastream Requirement 5: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/sensor Requirement 6: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/observed-property Requirement 7: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/observation Requirement 8: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/feature Requirement 9: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/feature-type

The OGC SensorThings API v2.0 Core data model is depicted in Figure 2.

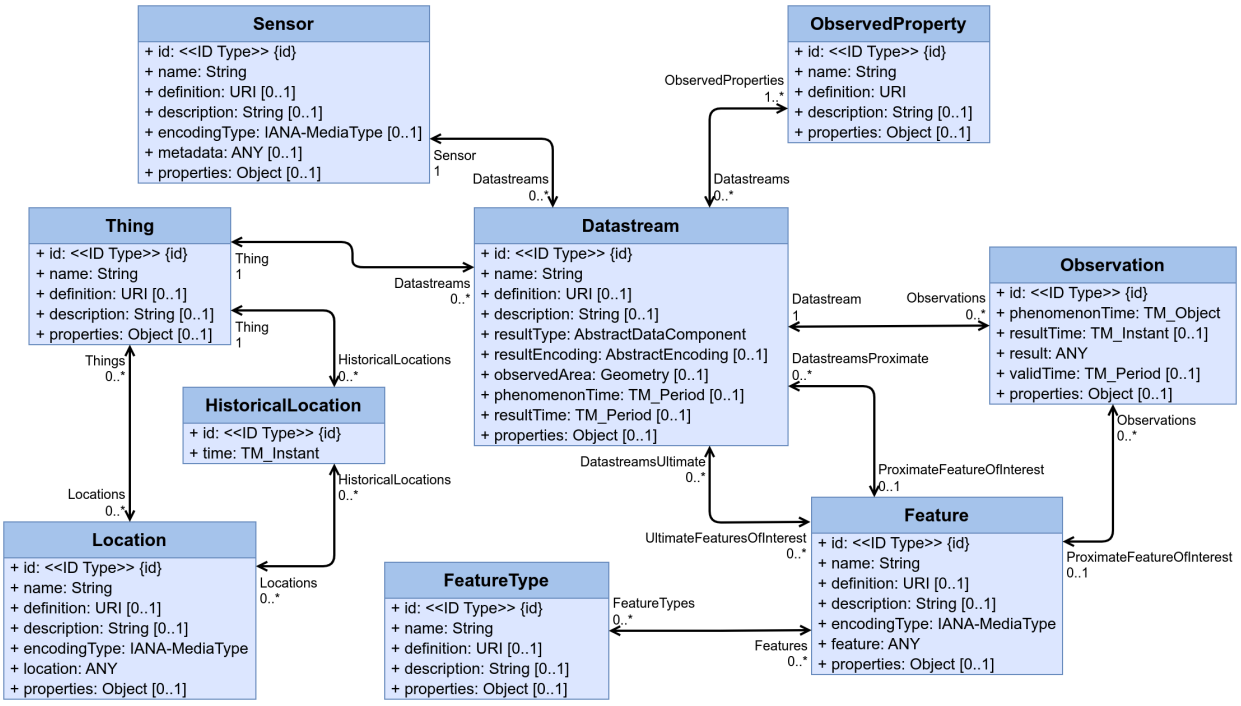


Figure 2 — UML class diagram of the OGC SensorThings API Core data model

In this section, each entity depicted in Figure 2 and its relationships with other entities is defined. Additionally, examples of modelling the entities in different contexts are provided.

8.2. Recurring attributes

Several attributes are used in several classes. They always have the same semantic meaning, regardless of the class they are used in.

`id`	The primary key of the class. An Entity is uniquely identified, among Entities of the same class and in the same service, by its `id` value. The data type of the `id` attribute may vary between services, or even between classes in the same service, but must be the same for all Entities of the same class in the same service. In most cases the value of the `id` attribute of an Entity will be generated by the service when the Entity is created, but a service may allow users to specify a value.
`name`	The name of an Entity is a (short) character string that describes the Entity. It does not need to be unique among entities of the same type, though, when two entities of the same type have the same name, there should be another distinguishing attribute. Two Things with the same name are likely to cause confusion, but two Datastreams with the same name are generally acceptable, as long as these Datastreams are associated to different Things.
`description`	The description of an Entity is a longer, human readable, character string giving a descriptive text of the Entity.
`definition`	The definition of an Entity is a unique external identifier, linking the Entity to an external, authoritative source. The definition enables different services to represent different aspects of the same systems. It can, for instance, be a URL to a feature in an OGC API — Features service, or a DID pointing to a cryptographically signed resource.
`properties`	A set of user-defined named values. The values may be of any type, including nested arrays or name-value sets. In the default JSON encoding the `properties` attribute is a JSON-Object.

8.3. Thing

REQUIREMENT 1: CLASS THING	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/thing`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`

REQUIREMENT 1: CLASS THING	
A	Each Thing entity shall have the mandatory attributes listed in Table 3.
B	Each Thing entity may have the optional attributes listed in Table 3.
C	Each Thing entity shall have the direct relation between a Thing entity and other entity types listed in Table 4.

The OGC SensorThings API Standard follows the ITU-T definition 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. A Thing is related to the Platform entity as described in Section 4.9.2.1 of OGC 16-079 in a way that any entity that can be modelled as a Thing may be subsequently translated to a Platform and vice versa.

Examples of the use of Thing are:

- A weather station that houses a varying set of environmental sensors;
- A thermostat in a room, monitoring and controlling the temperature on that room;
- A drone that mounts a LiDAR sensor for mapping purposes;
- A science vessel that has many sensors installed; and
- An upstream oil well that is equipped with compound gas detection devices.

Table 3 — Attributes of a Thing entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`id`	A unique, read-only attribute that serves as an identifier for the entity.	ANY	1
`name`	The name attribute provides a label for a Thing entity, commonly a descriptive name.	String	1
`definition`	The URI linking the Thing to an external definition.	URI	0..1
`description`	This is a short description of the corresponding Thing entity.	String	0..1
`properties`	A JSON Object containing user-annotated properties.	JSON_ Object	0..1

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

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
`Location`	The Location entity locates the Thing. Multiple Things may be located at the same Location. A Thing can only be in one physical location at the same time, but this location may be described in different ways, using different representations. In such case, the Thing has more than one Locations.	`Location`	0..*
`HistoricalLocations`	The Locations that the Thing was located at in the past.	`HistoricalLocation`	0..*
`Datastreams`	The Datastreams associated to the Thing.	`Datastream`	0..*

```
{
  "@context": "https://example.org/v2.0/$metadata#Things/$entity",
  "@id": "Things(1)",
  "id": 1,
  "name": "Oven",
  "description": "This thing is an oven.",
  "properties": {
    "owner": "Ulrike Schmidt",
    "color": "Black"
  },
  "Locations@navigationLink": "Things(1)/Locations",
  "Datastreams@navigationLink": "Things(1)/Datastreams",
  "HistoricalLocations@navigationLink": "Things(1)/HistoricalLocations"
}
```

Listing 1 — Example of a Thing entity returned by a HTTP end point.

8.4. Location

REQUIREMENT 2: CLASS LOCATION	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/location`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`
A	Each Location entity shall have the mandatory attributes listed in Table 5.

REQUIREMENT 2: CLASS LOCATION	
B	Each Location entity may have the optional attributes listed in Table 5.
C	Each Location entity shall have the direct relation between a Location entity and other entity types listed in Table 6.

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

Section 7.1.4 of [OGC 20-082r4 and ISO 19156:2023] provides a detailed explanation of observation location. Examples of the use of Location are as follow.

- An air quality sensing facility's Location can be the physical location where the facility is situated, but the (proximate) Feature that is characterized by the Observation could be the air envelope around the Sensor which is subsequently used to estimate the air quality of the district where the facility is situated.
- A drone that mounts a LiDAR Sensor may have its Location as the geo-referenced area over which the drone is scheduled to fly, whereas the Feature could be the individual objects mapped by the Sensor within that geo-referenced area

Table 5 — Attributes of a Location entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`id`	A unique, read-only attribute that serves as an identifier for the entity.	ANY	1
`name`	The name attribute provides a label for a Location entity, commonly a descriptive name.	String	1
`definition`	The URI linking the Location to an external definition.	URI	0..1
`encodingType`	The encoding type of the Location attribute. See Table 7 for some suggested values.	IANA-Media Type	1
`location`	The identifiable location of the Thing	ANY	1
`description`	Description of the Location.	String	0..1
`properties`	A JSON Object containing user-annotated properties.	JSON_Object	0..1

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

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
`Things`	The Things located at the source Location. Multiple Things may locate at the same Location.	`Thing`	0..*
`HistoricalLocations`	The HistoricalLocations of things that have been located at this Location.	`HistoricalLocation`	0..*

Table 7 — Non-exhaustive list of IANA-MediaType used for identifying types for the encodingType of the Location and Feature entities

ENCODING TYPE	IANA-MEDIATYPE
`GeoJSON`	`application/geo+json`
`GeoPose`	`application/geopose+json`

```
{
  "@context": "https://example.org/v2.0/$metadata#Locations/$entity",
  "@id": "Locations(1)",
  "id": 1,
  "name": "CCIT",
  "description": "Calgary Center for Innvative Technologies",
  "encodingType": "application/geo+json",
  "location": {
    "type": "Feature",
    "geometry":{
      "type": "Point",
      "coordinates": [-114.06,51.05]
    }
  },
  "Things@navigationLink": "Locations(1)/Things",
  "HistoricalLocations@navigationLink": "Locations(1)/HistoricalLocations",
}
```

Listing 2 — Example of a Location entity using a GeoJSON Feature.

```
{
  "@context": "https://example.org/v2.0/$metadata#Locations/$entity",
  "@id": "Locations(2)",
  "id": 2,
  "name": "IOSB",
  "description": "Fraunhofer-Institut für Optronik, Systemtechnik und
Bildauswertung IOSB",
  "encodingType": "application/geo+json",
  "location": {
    "type": "Point",
    "coordinates": [8.426, 49.015]
  },
  "Things@navigationLink": "Locations(2)/Things",
  "HistoricalLocations@navigationLink": "Locations(2)/HistoricalLocations",
```

```
}
```

Listing 3 — Example of a Location entity using a GeoJSON Geometry.

NOTE: When using a GeoJSON encoding for the location attribute, the value can either be a GeoJSON Feature or a GeoJSON Geometry.

```
{
  "@context": "https://example.org/v2.0/$metadata#Locations/$entity",
  "@id": "Locations(3)",
  "id": 3,
  "name": "Hamburg Port",
  "description": "Location at Hamburg Harbor",
  "encodingType": "text/plain",
  "location": "POINT(9.9937 53.5511)",
  "Things@navigationLink": "Locations(3)/Things",
  "HistoricalLocations@navigationLink": "Locations(3)/HistoricalLocations"
}
```

Listing 4 — Example of a Location entity using WKT.

```
{
  "@context": "https://example.org/v2.0/$metadata#Locations/$entity",
  "@id": "Locations(4)",
  "id": 4,
  "name": "Weather Station Alpha",
  "description": "Rooftop weather monitoring station",
  "encodingType": "application/vnd.ogc.fg+json",
  "location": {
    "type": "Feature",
    "place": {
      "type": "Point",
      "coordinates": [13.4050, 52.5200]
    },
    "geometry": {
      "type": "Point",
      "coordinates": [13.4050, 52.5200]
    },
    "properties": {
      "floor": 5,
      "building": "Tower A",
      "installation_date": "2023-01-15"
    }
  },
  "Things@navigationLink": "Locations(4)/Things",
  "HistoricalLocations@navigationLink": "Locations(4)/HistoricalLocations"
}
```

Listing 5 — Example of a Location entity using JSON-FG.

```
{
  "@context": "https://example.org/v2.0/$metadata#Locations/$entity",
  "@id": "Locations(5)",
  "id": 5,
  "name": "Drone Camera Position",
  "description": "Position and orientation of aerial survey drone",
  "encodingType": "application/geopose+json",
  "location": {
    "position": {
      "lat": 48.8584,
      "lon": 2.2945,
      "h": 300.5
```

```
    },
    "quaternion": {
      "x": 0.0,
      "y": 0.0,
      "z": 0.7071,
      "w": 0.7071
    }
  },
  "Things@navigationLink": "Locations(5)/Things",
  "HistoricalLocations@navigationLink": "Locations(5)/HistoricalLocations"
}
```

Listing 6 — Example of a Location entity using OGC GeoPose.

8.5. HistoricalLocation

REQUIREMENT 3: CLASS HISTORICALLOCATION	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/historical-location`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`
A	Each HistoricalLocation entity shall have the mandatory attributes and may have the optional attributes listed in Table 8.
B	Each HistoricalLocation entity shall have the direct relation between a HistoricalLocation entity and other entity types listed in Table 9.
C	When the Locations relation of a Thing changes, a new HistoricalLocation shall be created and added to the Thing automatically by the service. The current Location(s) of the Thing shall only be added to this autogenerated HistoricalLocation automatically by the service, and shall not be created as HistoricalLocation directly by user.
D	When a user directly adds new HistoricalLocation, and the time of this new HistoricalLocation is later than the latest HistoricalLocation for the Thing, then the Locations of the Thing are changed to the Locations of this new HistoricalLocation.

A Thing's HistoricalLocation entity set provides the times of the current (i.e., last known) and previous locations of the Thing. It can be used to model the path observed by a moving Thing. An example of the use of HistoricalLocation is:

- A drone that measures methane leaks over a large basin may want to record the trajectory through which it flies. HistoricalLocation should then record the individual Locations of the drone over time.

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

SensorThings API enabled system. Another use case is to track the Location of a Thing when a permanent network connection is not available. If the Location of a Thing is changed at a later time, when a network connection is available again, then the auto-generated Time of the HistoricalLocation entity would not reflect the time when the Thing was actually at the set Location, but only the time at which the change was sent to the server. To resolve this, the Location of a Thing can also be changed by adding a HistoricalLocation. If the time of a manually created HistoricalLocation is later than the time of all existing HistoricalLocations, then the Location of the Thing is updated to the Location of this manually created HistoricalLocation.

Table 8 — Attributes of a HistoricalLocation entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`id`	A unique, read-only attribute that serves as an identifier for the entity.	ANY	1
`time`	The time when the Thing is known at the Location.	TM_Instant	1

Table 9 — Direct relation between a HistoricalLocation entity and other entity types

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
Locations	The Locations for this HistoricalLocation.	`Location`	1..*
Thing	The Thing this HistoricalLocation positions in time.	`Thing`	1

```
{
  "@context": "https://example.org/v2.0/$metadata#HistoricalLocations/$entity",
  "@id": "HistoricalLocations(1)",
  "id": 1,
  "time": "2020-03-20T16:35:23.383586Z",
  "Thing@navigationLink": "HistoricalLocations(1)/Thing",
  "Locations@navigationLink": "HistoricalLocations(1)/Locations"
}
```

Listing 7 — Example of a HistoricalLocation entity returned by a HTTP end point.

8.6. Datastream

REQUIREMENT 4: CLASS DATASTREAM	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/datastream`

REQUIREMENT 4: CLASS DATASTREAM	
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`
A	Each Datastream entity shall have the mandatory attributes attributes listed in Table 10.
B	Each Datastream entity may have the optional attributes listed in Table 10.
C	Each Datastream entity shall have the direct relation between a Datastream entity and other entity types listed in Table 11.
D	The service shall ensure the linked ObservedProperties match the definitions used in the `resultType`.

A Datastream groups a collection of Observations into a time series measuring the same ObservedProperties by the same Sensor for the same Thing. Examples of Datastreams could be:

- An air quality monitoring station may have multiple Datastreams, each recording a specific pollutant measured by a specific sensor; and
- A sensor that measures multiple ObservedProperties can generate a single Datastream with a composite resultType.

A Datastream can both have a proximate feature of interest and one or more ultimate features of interest. An ultimate feature of interest is the broadest domain feature that the Observations in the Datastream can be said to represent. The proximate feature of interest is the object on which a measurement is taken as representative of the ultimate feature of interest. An individual Observation can also have a proximate feature of interest, which would, in most cases, be a sample with a limited set of Observations taken on it.

For example, in water quality the ultimate feature of interest could be a river, while the proximate feature of interest is a section of the river, or a specific point in the river where measurements or samples are taken. If samples are taken, these samples are the proximate feature of interest linked directly to the Observations taken on them.

A Datastream can have multiple ultimate features of interest when its Observations are equally representative for multiple domain features. For instance, a well can intersect multiple aquifers in such a way that the water samples taken from this well can not be clearly assigned to one of these aquifers. In this case, Observations on these samples have all the aquifers of the well as ultimate features of interest.

Table 10 — Attributes of a Datastream entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`id`	A unique, read-only attribute that serves as an identifier for the entity.	ANY	1

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`name`	The name attribute provides a label for Datastream entity, commonly a descriptive name.	String	1
`definition`	The URI linking the Datastream to an external definition.	URI	0..1
`description`	Description of the Datastream entity.	String	0..1
`resultType`	The formal description of the structure of the `result` attribute of the Observations in this Datastream. Contains the unit of measurement and the `@id` of the Observed Property.	JSON_Object (SWE-Common AbstractDataComponent)	1
`resultEncoding`	The encoding of the `result` attribute of the Observations in this Datastream. The default encoding, used when this attribute is null or empty, is `JSONEncoding` with the default settings.	JSON_Object (SWE-Common AbstractEncoding)	0..1
`observedArea`	The spatial bounding box of the spatial extent of the Features that belong to the Observations associated with this Datastream. This is usually generated by the server.	Geometry	0..1
`phenomenonTime`	The temporal interval of the phenomenon times of all observations belonging to this Datastream. This is usually generated by the server.	TM_Period	0..1
`resultTime`	The temporal interval of the result times of all observations belonging to this Datastream. This is usually generated by the server.	TM_Period	0..1
`properties`	A JSON Object containing user-annotated properties.	JSON_Object	0..1

TM_Period is by default encoded as a complex type with a start (mandatory) and end (mandatory) attributes of type TM_Instant.

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

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
`Thing`	The Thing this Datastream holds Observations for.	`Thing`	1

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
`Sensor`	The Sensor that made the Observations in this Datastream.	`Sensor`	1
`ObservedProperty`	The ObservedProperty of the Observations in this Datastream. The Observations in a Datastream can hold values for multiple ObservedProperties, but the ObservedProperties are the same for all Observations in the same Datastream. The service must ensure the linked Observed Properties match the definitions used in the `resultType`.	`ObservedProperty`	1..*
`Observations`	The Observations for a Datastream.	`Observation`	0..*
`ProximateFeatureOfInterest`	This relation links to a Feature that has the role proximate feature of interest for all the Observations in a Datastream.	`Feature`	0..1
`UltimateFeaturesOfInterest`	This relation links to zero or more Features that have the role ultimate feature of interest for all the Observations in a Datastream.	`Feature`	0..*

The resultType defines the result types for specialized single and multi observations based on the JSON encoding of the SWE Common Data Model (OGC 24-014). The result of an Observation may be a single simple number or String, or it may contain a complex JSON structure holding multiple values for multiple ObservedProperties. The exact definition for this result structure, and which unit of measurement and which ObservedProperty pertains to each value in the result structure is exactly described by this resultType Object.

The resultType contains references to the ObservedProperty or ObservedProperties of the Observations in the Datastream. When a change is made to the resultType, the server MUST ensure the ObservedProperties linked to the Datastream match the references used in the the `definition` attribute(s) of the resultType, and create or remove relations as needed. If the resultType contains references to non-existing ObservedProperties the request must be rejected with a Bad Request error. The values of the definition attributes in the resultType must be valid entity-ids (`@id`) of ObservedProperties. The structure of the resultType must not be changed once a Datastream has Observations, since the existing Observations would not fit the new structure.

In most cases each Observation holds a single numeric result value, measured by a single Sensor. In this case the resultType is of the SWE-Common class `Quantity` (though a number could also be a `Count`) and thus must have the attributes `type`, `definition`, `label`, and `uom`.

- The `type` attribute defines the type of the result and has the value `Quantity`.
- The `definition` must be the same as the `definition` attribute in the single ObservedProperty linked to the Datastream.
- The `uom` attribute contains the unit of measurement and is an object that must contain either a `code` attribute with the UCUM code of the unit, or a `href` attribute with a URI pointing to a unit definition. It may also contain the `label` and `symbol` attributes that can be displayed in user interfaces.
- The `label` attribute is somewhat redundant in this case, but since it is mandatory it can be set to the same value as the ObservedProperty `name`.

```
{
  "name": "Oven temperature",
  "description": "This is a datastream measuring the air temperature in an
oven.",
  "resultType": {
    "type": "Quantity",
    "label": "Temperature",
    "definition": "ObservedProperties(1)",
    "uom": { "code": "Cel", "label": "degree Celsius", "symbol": "°C" }
  }
}
```

Listing 8 — A Datastream example measuring a scalar Observation

```
{
  "result": 25.1,
  "phenomenonTime":  {
    "start": "2021-13-14T15:16:00Z"
  }
}
```

Listing 9 — An Observation for the Datastream defined in the example above

Another common type of result is a value from a key list. An example would be an Observation of the current weather as `Fair` or `Overcast`, or the geological dating of a rock sample to be `Jurassic`. In the latter case, the code space that defines the values could be the SWEET[2] ontology. In this example the resultType is a `Category` and thus must have the attributes `type`, `definition`, `label`, and `codeSpace`.

- The `type` attribute defines the type of the result and has the value `Category`.
- The `definition` must be the same as the `definition` attribute in the single ObservedProperty linked to the Datastream.
- The `codeSpace` attribute must be a reference that defines the valid values that can be used as results.
- The `label` attribute is somewhat redundant in this case, but since it is mandatory it can be set to the same value as the ObservedProperty `name`.

[2] https://earthportal.eu/ontologies/SWEET

```
{
  "name": "Sample Dating",
  "description": "This is a datastream containing the geological datings of
rock samples.",
  "resultType": {
    "type": "Category",
    "label": "Period",
    "definition": "ObservedProperties(2)",
    "codeSpace": "http://sweetontology.net/stateTimeGeologic/",
    "constraint": {
      "values": [ "cretaceous", "jurassic", "triassic" ]
    }
  }
}
```

Listing 10 — A Datastream example for Observations with category values from a predefined code space, but allowing only three specific values

```
{
  "result": "jurassic",
  "phenomenonTime":  {
    "start": "2021-13-14T15:16:00Z"
  }
}
```

Listing 11 — An Observation for a Datastream defined in the example above

In some cases, a (composite) Sensor generates multiple values that should be kept together. This can be achieved with a `DataRecord` resultType. In the below example, a temperature and pressure value are stored as a pair, per Observation. The resultType has the SWE-Common class `DataRecord`, which is a composite class that contains sub-entries. The mandatory attributes of a DataRecord are `type` and `fields`:

- The `type` attribute defines the type of the result and has the value `DataRecord`.
- The `fields` attribute is an array of other SWE-Common components. In this example there are two `Quantity` entries. Each entry has the mandatory attributes for a `Quantity`.

Since there are two *distinct* `definition` attributes in the resultType, this Datastream must be linked to two ObservedProperties that match the two definitions.

```
{
  "name": "Temperature and Pressure",
  "description": "This is a datastream containing temperature and pressure
measurement sets.",
  "resultType": {
    "type": "DataRecord",
    "fields": [
      {
        "name": "temp",
        "type": "Quantity",
        "label": "Air Temperature",
        "definition": "ObservedProperties(1)",
        "uom": { "code": "Cel", "label": "degree Celsius", "symbol": "°C"  }
      },
      {
        "name": "press",
        "type": "Quantity",
        "label": "Air Pressure",
```

```
        "definition": "ObservedProperties(3)",
        "uom": { "code": "mbar", "label": "Millibar", "symbol": "mBar"  }
      }
    ]
  }
}
```

Listing 12 — A Datastream example measuring multiple observedProperties

```
{
  "result": {
    "temp": 15,
    "press": 1024
  },
  "phenomenonTime": {
    "start": "2021-13-14T15:16:00Z"
  }
}
```

Listing 13 — An Observation for a Datastream defined in the example above

The `resultEncoding` attribute, can be used to specify how the structure from the `resultType` attribute is encoded in the `result` attribute of the Observations. If the `resultEncoding` is empty, or null, a JSONEncoding is used with the default configuration, as seen in the examples above. The example below specifies a modified JSONEncoding, to mimic the result encoding of the MultiDatastream, as specified in version 1.1 of the OGC SensorThings API.

```
{
  "name": "Temperature and Pressure",
  "description": "This is a datastream containing temperature and pressure
measurement sets.",
  "resultType": {
    "type": "DataRecord",
    "fields": [
      {
        "name": "temp",
        "type": "Quantity",
        "label": "Air Temperature",
        "definition": "ObservedProperties(1)",
        "uom": { "code": "Cel", "label": "degree Celsius", "symbol": "°C"  }
      },
      {
        "name": "press",
        "type": "Quantity",
        "label": "Air Pressure",
        "definition": "ObservedProperties(3)",
        "uom": { "code": "mbar", "label": "Millibar", "symbol": "mBar"  }
      }
    ]
  },
  "resultEncoding": {
    "type": "JSONEncoding",
    "recordsAsArrays": true
  }
}
```

Listing 14 — A Datastream example measuring multiple observedProperties using a custom encoding to mimic the old MultiDatastream format

```
{
  "result": [15, 1024]
  },
```

```
  "phenomenonTime": {
    "start": "2021-13-14T15:16:00Z"
  }
}
```

Listing 15 — An Observation for a Datastream defined in the example above

8.7. Sensor

REQUIREMENT 5: CLASS SENSOR	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/sensor`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`
A	Each Sensor entity shall have the mandatory attributes listed in Table 12.
B	Each Sensor entity may have the optional attributes listed in Table 12.
C	Each Sensor entity shall have the direct relation between a Sensor entity and other entity types listed in Table 13.

A Sensor is an entity that observes a property or phenomenon with the goal of producing an estimate of the value of the property. A Sensor may represent a piece of hardware, but a Sensor may also be a human or an algorithm implemented in software.

A Sensor entity in an OGC SensorThings API deployment can represent either a specific sensor instance, or a more generic sensor type. The distinction is generally quite clear from the Sensor metadata. Does the entity have detailed metadata that is specific to a sensor instance, like a serial number and calibration data, or is it used for many Datastreams attached to many different Things.

Table 12 — Attributes of a Sensor entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`id`	A unique, read-only attribute that serves as an identifier for the entity.	ANY	1
`name`	The name attribute provides a label for a Sensor entity, commonly a descriptive name.	String	1
`definition`	The URI linking the Sensor to an external definition.	URI	0..1

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
description	Description of the Sensor entity.	String	0..1
encodingType	The encoding type of the metadata attribute. If the metadata field is present, an encodingType must be specified. See Table 14 for some suggested values.	IANA-Media Type	0..1
metadata	Detailed description of the Sensor or system. The metadata type is defined by encodingType.	String	0..1
properties	A JSON Object containing user-annotated properties as key-value pairs	JSON_ Object	0..1

Table 13 — Direct relation between a Sensor entity and other entity types

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
Datastreams	The Datastreams that hold Observations produced by this Sensor.	Datastream	0..*

Table 14 — Non-exhaustive list of code values used for identifying types for the encodingType of the Sensor entity

SENSOR ENCODING TYPE	IANA-MEDIATYPE
PDF	application/pdf
HTML	text/html

The Sensor encodingType enables clients to know how to interpret the metadata value. Note that the metadata attribute may contain either a URL to metadata content (e.g., an `https://` , `ftp://`, etc. link to a PDF, SensorML, or HTML document) or the metadata content itself (in the case of text/plain or other encodingTypes that can be represented as valid JSON). It is up to clients to perform string parsing necessary to properly handle metadata content.

```
{
  "@context": "https://example.org/v2.0/$metadata#Sensors/$entity",
  "@id": "Sensors(1)",
  "id": 1,
  "name": "TMP36",
  "description": "TMP36 - Analog Temperature sensor",
  "encodingType": "application/pdf",
  "metadata": "http://example.org/TMP35_36_37.pdf",
  "Datastreams@navigationLink": "Sensors(1)/Datastreams"
}
```

Listing 16 — Example of a Sensor entity returned by a HTTP end point.

8.8. ObservedProperty

REQUIREMENT 6: CLASS OBSERVEDPROPERTY	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/observed-property`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`
A	Each ObservedProperty entity shall have the mandatory attributes listed in Table 15.
B	Each ObservedProperty entity may have the optional attributes listed in Table 15.
C	Each ObservedProperty entity shall have the direct relation between an ObservedProperty entity and other entity types listed in Table 16.

An ObservedProperty is a property of a Feature that is being observed by a Sensor, such as temperature, humidity, population count or color. It should be uniquely identified by its `definition`, which should point to an external vocabulary by means of a URL, URI or DID.

Table 15 — Attributes of an ObservedProperty entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`id`	A unique, read-only attribute that serves as an identifier for the entity.	ANY	1
`name`	The name attribute provides a label for an ObservedProperty entity, commonly a descriptive name.	String	1
`definition`	The URI of the ObservedProperty.	URI	1
`description`	Description of the ObservedProperty.	String	0..1
`properties`	A JSON Object containing user-annotated properties as key-value pairs	JSON_ Object	0..1

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

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
`Datastreams`	Datastreams that hold Observations that observed this ObservedProperty.	`Datastream`	0..*

```
{
  "@context": "https://example.org/v2.0/$metadata#ObservedProperties/$entity",
  "@id": "ObservedProperties(1)",
  "id": 1,
  "name": "DewPoint Temperature",
  "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.",
  "definition": "http://dbpedia.org/page/Dew_point",
  "Datastreams@navigationLink": "ObservedProperties(1)/Datastreams"
}
```


Listing 17 — Example of an ObservedProperty entity returned by a HTTP end point.

8.9. Observation

REQUIREMENT 7: CLASS OBSERVATION	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/observation`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`
A	Each Observation entity shall have the mandatory attributes listed in Table 17.
B	Each Observation entity may have the optional attributes listed in Table 17.
C	Each Observation entity shall have the direct relation between an Observation entity and other entity types listed in Table 18.
D	When a service receives a POST Observations without phenomenonTime, the service shall assign the current server time to the value of the phenomenonTime.

An Observation provides a value for an ObservedProperty of a Feature, as observed by a Sensor. This value can be of any type, as described by the resultType of the Datastream that Observation is associated with.

Table 17 — Attributes of an Observation entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`id`	A unique, read-only attribute that serves as an identifier for the entity.	ANY	1

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`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 phenomenonTime when POST new Observations, even though phenomenonTime is a mandatory attribute.	TM_ Object	1
`result`	The estimated value of an ObservedProperty from the Observation.	ANY	1
`resultTime`	The time of the Observation's result was generated	TM_ Instant	0..1
`validTime`	The time period during which the result may be used	TM_ Period	0..1
`properties`	A JSON Object containing user-annotated properties as key-value pairs (usually showing the environmental conditions during measurement)	JSON_ Object	0..1

TM_Object is by default encoded as a complex type with a start (mandatory) and end (optional) attributes of type TM_Instant. This means it can either describe a time instant, when only the start is present, or a time interval with both stand and end are present.

TM_Period is by default encoded as a complex type with a start (mandatory) and end (mandatory) attributes of type TM_Instant. This means it always describes a time interval with fixed starting and ending instants.

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

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
`Datastream`	The Datastream this Observation exists in.	`Datastream`	1
`ProximateFeatureOfInterest`	The Feature observed by the Observation.	`Feature`	0..1

An Observation can be directly linked to a Feature, through the relation `ProximateFeatureOfInterest`. Features linked to an Observation in this way are generally samples, either real, physical ones. Examples are water samples taken from a river, or the place that a moving Thing happened to be in, when it made a measurement.

In case the Feature is a domain object, such as a river, a building, or a plot of land, the Feature is indirectly linked to the Observation through the `UltimateFeaturesOfInterest` or `ProximateFeatureOfInterest` relations on the Datastream. It is also possible for all relations to exist, in which case the `ProximateFeatureOfInterest` of the Observation is a sample of the `ProximateFeatureOfInterest` of the Datastream, which is in turn representative of the `UltimateFeaturesOfInterest`.

A third case is possible: When the target of the observation is (a sub-part of) the Thing itself. For instance, when the Observation is on the battery-level of a drone. In this case neither the `ProximateFeatureOfInterest` on the Observation, nor the `ProximateFeatureOfInterest` or `UltimateFeaturesOfInterest` relations on the Datastream need to be set.

```
{
  "@context": "https://example.org/v2.0/$metadata#Observations/$entity",
  "@id": "Observations(1)",
  "id": 1,
  "phenomenonTime": {
    "start": "2017-11-12T13:00:00Z",
    "end": "2017-11-12T14:00:00Z"
  },
  "resultTime": "2017-11-12T14:00:00Z",
  "result": 12.5,
  "Datastream@navigationLink": "Observations(1)/Datastream",
  "ProximateFeatureOfInterest@navigationLink": "Observations(1)/
ProximateFeatureOfInterest"
}
```

Listing 18 — Example of an Observation entity returned by a HTTP end point.

8.10. Feature

REQUIREMENT 8: CLASS FEATURE	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/feature`
INCLUDED IN	Requirements class 1: `http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`
A	Each Feature entity shall have the mandatory attributes listed in Table 19.
B	Each Feature entity may have the optional attributes listed in Table 19.
C	Each Feature entity shall have the direct relation between a Feature entity and other entity types listed in Table 20.

An Observation assigns a value to a property of a subject by applying an ObservingProcedure. The subject is the Feature that can take the role of proximate feature of interest or ultimate feature of interest of the Observation [OGC 20-082r4 and ISO 19156:2023]. In cases where estimating the value of a property of interest is not possible directly, a proxy feature is used. Such an application typically requires taking a sample of the ultimate feature of interest such that this sample, the proximate feature of interest, represents an approximation of the domain feature.

Some examples of features are as follow.

- The ultimate feature of a WiFi enabled thermostat, with a built-in temperature sensor, could be the room that the thermostat monitors. The exact position of the thermostat is captured in a Location entity of the Thing representing the thermostat.
- In the case of water quality monitoring, water samples are taken from a river at a specific sampling location. The water sample, the sampling location, and the river are all Features. The river Feature is attached to the Datastreams of the Observations taken on the sample through the UltimateFeaturesOfInterest relation. The sampling location is the ProximateFeatureOfInterest attached to the Datastreams of the Observations taken on the sample. Finally, the sample is the ProximateFeatureOfInterest directly attached to the various Observations.
- In the case of remote sensing, the Feature observed can be the individual geographical parcel or swath that is being sensed, while the Location entity captures the position of the device (drone, satellite) that hosts the sensors.

Table 19 — Attributes of a Feature entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY
`id`	A unique, read-only attribute that serves as an identifier for the entity.	ANY	1
`name`	The name attribute provides a label for a Feature entity, commonly a descriptive name.	String	1
`definition`	The URI linking the Feature to an external definition.	URI	0..1
`description`	Description of the Feature.	String	0..1
`encodingType`	The encoding type of the feature attribute. See Table 7 for some suggested values.	IANA-Media Type	1
`feature`	The detailed description of the feature. The data type is defined by encodingType.	ANY	1
`properties`	A JSON Object containing user-annotated properties as key-value pairs	JSON_Object	0..1

Table 20 — Direct relation between a Feature entity and other entity types

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
`Observations`	All Observations that are made on this Feature	`Observation`	0..*
`DatastreamsProximate`	All Datastreams for the Observations of which this Feature is a proximate feature of interest.	`Datastream`	0..*

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
DatastreamsUltimate	All Datastreams for the Observations of which this Feature is an ultimate feature of interest.	Datastream	0..*


```
{
  "@context": "https://example.org/v2.0/$metadata#Features/$entity",
  "@id": "Features(1)",
  "id": 1,
  "name": "0113700020130227",
  "description": "Water Sample from LA NOYE À DOMMARTIN (80) taken on 2013-02-
27 at 10:20:00",
  "encodingType": "application/geo+json",
  "feature": {
    "type": "Point",
    "coordinates": [
      2.38961955,
      49.800951554
    ]
  },
  "FeatureType@navigationLink": "Features(1)/FeatureType",
  "Datastreams@navigationLink": "Features(1)/Datastreams",
  "Observations@navigationLink": "Features(1)/Observations"
}
```


Listing 19 — Example of a Feature entity using a GeoJSON Geometry.


```
{
  "@context": "https://example.org/v2.0/$metadata#Features/$entity",
  "@id": "Features(2)",
  "id": 2,
  "name": "City Center Park",
  "description": "A public park located in the heart of the city",
  "encodingType": "application/vnd.ogc.wkt",
  "feature": "POLYGON((30 10, 40 40, 20 40, 10 20, 30 10))",
  "FeatureType@navigationLink": "Features(2)/FeatureType",
  "Datastreams@navigationLink": "Features(2)/Datastreams",
  "Observations@navigationLink": "Features(2)/Observations"
}
```


Listing 20 — Example of a Feature entity using WKT Geometry.

8.11. FeatureType

REQUIREMENT 9: CLASS FEATURETYPE	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/feature-type
INCLUDED IN	Requirements class 1: http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core

REQUIREMENT 9: CLASS FEATURETYPE	
A	Each FeatureType entity shall have the mandatory attributes listed in Table 21.
B	Each FeatureType entity may have the optional attributes listed in Table 21.
C	Each FeatureType entity shall have the direct relation between a FeatureType entity and other entity types listed in Table 22.

The type or types of each Feature can be specified using the FeatureType class. The `definition` attribute of the FeatureType can be used to point to an external registry or code list, that defines the Type.

Table 21 — Attributes of a FeatureType entity

NAME	DESCRIPTION	DATA TYPE	MULTIPLICITY
`id`	A unique, read-only attribute that serves as an identifier for the entity.	ANY	1
`name`	The name attribute provides a label for a FeatureType entity, commonly a descriptive name.	String	1
`definition`	The URI that defines this FeatureType.	URI	0..1
`description`	Description of the FeatureType.	String	0..1
`properties`	A JSON Object containing user-annotated properties as key-value pairs	JSON_ Object	0..1

Table 22 — Direct relation between a FeatureType entity and other entity types

ROLE	DESCRIPTION	TARGET ENTITY TYPE	MULTIPLICITY
`Features`	All Features of a FeatureType.	`Feature`	0..*

```
{
  "@context": "https://example.org/v2.0/$metadata#FeatureType/$entity",
  "@id": "FeatureType(1)",
  "id": 1,
  "name": "Water Sample",
  "description": "A Sample taken from a river, lake or sea",
  "definition": "https://example.org/water_sample",
  "Features@navigationLink": "FeatureType(1)/Features"
}
```

Listing 21 — Example of a FeatureType entity.

9

ABSTRACT REST API

9 ABSTRACT REST API

9.1. Introduction

This Clause describes the abstract REST API. The abstract REST API can be implemented using various protocols, such as HTTP or MQTT version 5.

REQUIREMENTS CLASS 2: ABSTRACT API	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req-class/api/abstract`
OBLIGATION	requirement
TARGET TYPE	Target Type: Abstract API
CONFORMANCE CLASS	Conformance class A.5: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/abstract`
NORMATIVE STATEMENTS	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read` Requirements class 4: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud`

9.2. OData Entity Model

Every OData service utilizes an entity model which can be distributed over several schemas. The service describes this model through a metadata document accessible by a simple HTTP *GET* request to the `<serviceRoot>/$metadata` path. This Clause describes an overview of the types of entities in the metadata document that will be used to describe the OGC SensorThings API data model and its data model extensions. The entities listed below are also used to describe the REST and MQTT bindings of the STA service.

The contents of this Clause are based on:

- https://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-protocol.html#sec_DataModel

- https://docs.oasis-open.org/odata/odata-csdl-json/v4.01/odata-csdl-json-v4.01.html#sec_EntityModel

NOTE:The OData specification uses the term “property”. To avoid confusion with “ObservedProperty” and the “properties” attribute in almost every Entity Type, this document uses the terms “attribute” where OData uses the term “property”.

The entity model consists of the following elements.

- `Keyed element`: Element that has a key that uniquely identifies an instance of this element amongst other instances of the same type of element.
- `Entity Type`: Named, structured, keyed element that defines the attributes and relations of Entities. The key for an Entity Type consists of one or more primitive attributes of the Entity Type.
- `Entity`: Instance of an `Entity Type`. An Entity can have only one Entity Type.
- `Primitive Type`: Unnamed, unstructured type, such as String, Integer or DateTime. The full list can be found at: https://docs.oasis-open.org/odata/odata-csdl-json/v4.01/odata-csdl-json-v4.01.html#sec_PrimitiveTypes An attribute with a Primitive Type is called a primitive attribute.
- `Complex Type`: Named, structured, keyless, element consisting of a set of named attributes. Instances of Complex Types can not be accessed outside of the Entity they are contained in. An attribute with a Complex Type is called a complex attribute. Examples of complex types / complex attributes are:
  - TM_Interval (Observation/validTime): The complex type TM_Interval has mandatory declared attributes `start` and `end`.
  - TM_Object (Observation/resultTime): The complex type TM_Interval has mandatory declared attribute `start` and optional declared attribute `end`.
  - JSON_Object (properties): The complex type JSON_Object is an open type with no declared attributes. That means users can add any attributes they see fit.
- `Relation`: Named connections between Entities. Relations are defined as `navigation attributes` in Entity Types. Relations have a cardinality.
- `Attributes`: Named elements that are part of structured types (Entity Types and Complex Types). Attributes can be Primitive-, Complex- or Navigation attributes. Attributes defined as part of a structured type (Entity/Complex Type) are `declared attributes`. A structured type can allow attributes that are added dynamically, as long as these `dynamic attributes` do not have the same name as a declared attribute. Structured types that allow dynamic attributes are called `Open Types`.
- `Entity Set`: A named collection of Entities of the same `Entity Type`. Each Entity is uniquely identified among other Entities in the same Entity Set, by its key. Entity Sets are the entry points into the data model.
- `Annotation`: Additional information attached to model or instance elements. annotations can be identified by the `@` character. If the annotation starts with the `@` character, then

the annotation pertains to the object containing the annotation. If the annotation has an attribute name before the `@` charater, then the annotation pertains the attribute with that name. Annotations are server-generated additional information that a client can use when processing the data. A client can ignore any annotations it does not understand. The default annotations can be found in Table 23. Additional annotations may be defined in the service metadata.

- `EntityId`: A globally unique identifier for an Entity. If present in Entity representations, the EntityId is specified in the `@id` annotation.

Each Entity has a primary key that is composed of one or more of the `declared attributes` of the entity type. Which attributes of the Entity Type make up the primary key of the Entity Type is specified in the service metadata document. For the data models specified in this document the primary key is always the field `id`, though the data type of the field may vary between services. In most cases the primary key field will be server generated at the time an Entity is created, but a service may allow for client-specified primary key fields.

URLs in documents may be absolute or relative. Relative URLs are relative to the context URL in the document, or if there is no context url, relative to the request URL. See https://docs.oasis-open.org/odata/odata-json-format/v4.01/odata-json-format-v4.01.html#sec_RelativeURLs.

Table 23 — Common control annotations

NAME	DEFINITION	DATA TYPE
@context	The context url contains the metadata document that describes the context of a payload. See Clause 9.4	URL
@id	The annotation `@id` is the `entity-id`, an absolute or relative URL of an entity that uniquely resolves to this entity. This was known as the `@selfLink` in STA version 1.	URL
@count	The total number of items in the requested entity set.	Integer
@nextLink	In a response that returns a set of entities, this link returns the next set of entities for the main set of the response.	URL
linkName@navigationLink	Annotations of the form `linkName@navigationLink` are the relative or absolute URL that retrieves the related entity or entities for the navigation attribute `linkName`.	URL
linkName@count	The total number of items in the related entity set of the navigation attribute `linkName`. Present when the navigation property `linkName` is an Entity Set and is expanded with the parameter `$count` set to `true`.	Integer
linkName@nextLink	This link returns the next set of entities for the expanded navigation property `linkName` if that navigation property is an Entity Set.	URL

9.3. Type Mappings

9.3.1. Simple Types

Table 24 lists the mapping of types used in the OGC SensorThings API core data model to OData primitive types.

Table 24 — mapping of types used in the SensorThings API core data model to OData primitive types

STA TYPE	ODATA TYPE	DESCRIPTION
ANY	Edm.Untyped	Any primitive or complex type
String	Edm.String	A character string
URI	Edm.String	A character string that must be a valid URI
Geometry	EDM.Geometry	A geometry type
TM_Instant	Edm.DateTimeOffset	A time instant, with timezone information

The STA core data model contains several complex types. They are `TM_Interval`, `TM_Object` and `JSON Object`.

9.3.2. Complex Type TM_Interval

TM_Interval is a time interval with a mandatory start and end time. The interval end is “open” meaning that the end-time is not part of the interval.

Table 25 — Attributes of a TM_Interval complex type

NAME	DESCRIPTION	DATA TYPE	MULTIPLICITY
`start`	The starting time (inclusive) of the time interval.	TM_Instant	1
`end`	The end time (exclusive) of the time interval.	TM_Instant	1

Attributes of the type TM_Interval can be directly compared to other time-related attributes and constants.

9.3.3. Complex Type TM_Object

TM_Object, used in `Observation/phenomenonTime`, is either a time instant or a time interval. This is modelled as a time interval with a mandatory start and an optional end time. If the end time is not present, the TM_Object is a time instant. If the end time is present, the TM_Object is a time interval with an "open" end, meaning that the end-time is not part of the interval.

Table 26 — Attributes of a TM_Object complex type

NAME	DESCRIPTION	DATA TYPE	MULTIPLICITY
`start`	The time instant of the time object, or the starting time (inclusive) of the time interval.	TM_Instant	1
`end`	The end time (exclusive) of the time interval.	TM_Instant	0..1

Filter operators and functions that accept times can work directly on TM_Interval and TM_Object types and will use the `start` or `end` time as appropriate. For instance, finding all Observations with a phenomenonTime before 2025-01-01T00:00:00Z can be done using: `phenomenonTime lt 2025-01-01T00:00:00Z` For phenomenonTimes that are an interval, this will ensure the end of the interval is before the threshold. Users can also filter specifically on the start or end by directly referencing these attributes.

9.3.4. Complex Type JSON_Object

JSON_Object is an open, complex type with no declared attributes. This means users can store any attributes in instances of this type that they see fit.

9.4. OData Common Schema Definition Language

The data model is specified in the metadata document that can be retrieved from the context url. It is described in a machine-readable way using the OData Common Schema Definition Language. See https://docs.oasis-open.org/odata/odata-csdl-json/v4.01/odata-csdl-json-v4.01.html

An example CSDL document describing a service hosting a STA v2.0 core data model can be found in Annex C. A shortened example with comments can be found in Annex B.

9.5. URL Encoding for constants

Encoding rules for constants in resource paths and query options are listed in Table 27

Table 27 — Encoding rules for constants in requests

TYPE	DESCRIPTION	EXAMPLE
String	Quoted using single-quotes (`'`). Single quotes in a string are doubled.	`'degree Celsius'` `'Abby''s Car'`
Numbe	Numbers are not quoted, use a decimal point (`.`), no thousands separator.	`0.31415926535897931e1` `-42`
Datetim	ISO8601 time with timezone. Datetimes are not quoted. Special characters (+) must be URL-Encoded.	`2012-12-03T07:16:23Z` `2012-12-03T07:16:23%2B08:00`
Boolear	Literal value `true` or `false`	`true` `false`
Null	Literal value `null`	`null`
Time Duratio	The keyword `duration` followed by an ISO8601 Duration in single quotes.	`duration'P1DT30M'`
Geome	The keyword `geometry` followed by Well Known Text (WKT) in single quotes.	`geometry'POINT(-122 43)'`
Geogra	The keyword `geography` followed by WKT in single quotes.	`geography'POINT(-122 43)'`

9.6. URL Patterns

9.6.1. Introduction

URL Patterns as defined in the OGC SensorThings API Standard consist of four parts: the Service Root, the STA API version number, the resource path and query options.

The service root URL	the free to chose base path of the service.
The API version number	encoded as: `"v" + majorversionnumber + "." + minorversionnumber`.

The resource path	consists of zero to many parts separated by the slash (/) character. Resource paths are case sensitive.
The query options	start with a question mark (?) character. At the top level, query options are separated with the ampersant (&) character.

```
http://example.org/v2.0/Datastreams(5)/Observations?$orderby=ID&$top=10
_________________/____/___________________________/___________________/
         |          |                |                        |
   service root   version      resource path           query options
```

Listing 22 — Complete URL example using HTTP bindings

```
base/topic/v2.0/Datastreams(5)/Observations?$orderby=ID&$top=10
_________/____/___________________________/___________________/
    |     version            |                    |
root Topic             resource path           query options
```

Listing 23 — Complete URL example using MQTT bindings

9.6.2. Service Document

URL Pattern: `<Service Root>/v2.0`

The URL pattern consisting of just the version number, optionally followed by a slash (/), addresses the service document. The service document content is described in Clause 9.9.2.2.

9.6.3. Service Metadata Document

URL Pattern: `<Service Root>/v2.0/$metadata`

The URL pattern consisting of the STA API version number followed by `/$metadata` addresses the service metadata document that describes the data model that the service exposes. The service metadata document is described in Clause 9.4.

9.6.4. Entity Set

URL Pattern: `<Service Root>/v2.0/EntityTypeName`

The URL pattern consisting of the STA API version number followed by / and the name of an entity type addresses the set of all entities of the requested type.

```
<Service Root>/v2.0/Datastreams
```

Listing 24 — Resource pattern for the Datastreams Entity Set

9.6.5. Entity

URL Pattern: `<Service Root>/v2.0/EntityTypeName(<primaryKey>)`

The URL pattern consisting of the name of an EntitySet URL, followed by, in brackets, a primary key value, addresses the single Entity from the set, that has the given primary key.

```
<Service Root>/v2.0/Datastreams(4)
```

Listing 25 — Resource pattern for a specific Datastream, for a service that uses Integers for the id field of Datastreams

```
<Service Root>/v2.0/Datastreams('xz42df')
```

Listing 26 — Resource pattern for a specific Datastream, for a service that uses Strings for the id field of Datastreams

9.6.6. Entity Attribute

URL Pattern: `<Service Root>/v2.0/EntityTypeName(<primaryKey>)/entityAttribute`

The URL pattern consisting of a URL that addresses an Entity (the base entity), followed by an Entity Attribute, addresses the value of this Entity Attribute of the base Entity. The used URL for the base entity may itself be a related entity URL.

```
<Service Root>/v2.0/Datastreams(4)/name
```

Listing 27 — Resource pattern for the name of a specific Datastream

```
<Service Root>/v2.0/Datastreams(4)/Thing/name
```

Listing 28 — Resource pattern for the name of the Thing of a specific Datastream

9.6.7. Entity Attribute raw value ($value)

URL Pattern: `<Service Root>/v2.0/EntityTypeName(<primaryKey>)/entityAttribute/$value`

The URL pattern consisting of a URL that addresses an Entity (the base entity), followed by an Entity Attribute, followed by `/$value`, addresses the raw value of this Entity Attribute of the base Entity. The URL used for the base entity may itself be a related entity URL.

```
<Service Root>/v2.0/Datastreams(4)/name/$value
```

Listing 29 — Resource pattern for the raw name of a specific Datastream

```
<Service Root>/v2.0/Datastreams(4)/Thing/name/$value
```

Listing 30 — Resource pattern for the raw name of the Thing of a specific Datastream

9.6.8. Related Entity

URL Pattern: `<Service Root>/v2.0/EntityTypeName(<primaryKey>)/EntityNavigationAttribute`

The URL pattern consisting of a URL that returns an Entity (the base entity), followed by a navigation attribute with a cardinality of one, addresses the related entity of the base entity. The URL used for the base entity may itself be a related entity URL.

```
<Service Root>/v2.0/Datastreams(4)/Thing
```

Listing 31 — Resource pattern for the Thing of a specific Datastream

```
<Service Root>/v2.0/Observations(42)/Datastream/Thing
```

Listing 32 — Resource pattern for the Thing of the Datastream of a specific Observation

9.6.9. Related EntitySet

URL Pattern: `<Service Root>/v2.0/EntityTypeName(<primaryKey>)/EntitySetNavigationAttribute`

The URL pattern consisting of a URL that addresses an Entity (the base entity), followed by a navigation attribute with a cardinality of many, addresses the related EntitySet of the base entity. The URL used for the base entity may itself be a related entity URL.

```
<Service Root>/v2.0/Datastreams(4)/Observations
```

Listing 33 — Resource pattern for the Observations of a specific Datastream

```
<Service Root>/v2.0/Datastreams(4)/Thing/Locations
```

Listing 34 — Resource pattern for the Locations of the Thing of a specific Datastream

9.6.10. Association link ($ref)

URL Pattern: `<Service Root>/v2.0/EntityTypeName(<primaryKey>)/NavigationAttribute/$ref`

The URL pattern consisting of a URL that addresses an Entity (the base entity), followed by a navigation attribute, followed by `/$ref`, addresses the entity-id(s) of the related entity or entities of the given navigation attribute.

```
<Service Root>/v2.0/Datastreams(4)/Thing/$ref
```

Listing 35 — Resource pattern for the association link of the Thing of a specific Datastream

```
<Service Root>/v2.0/Datastreams(4)/Observations/$ref
```

Listing 36 — Resource pattern for the association link to the Observations related to a specific Datastream

For navigation attributes with a cardinality of many, the navigation attribute may be followed by a primary-key in brackets, to address the specific association between the base entity and the related entity.

```
<Service Root>/v2.0/Datastreams(4)/Observations(1)/$ref
```

Listing 37 — Resource pattern for the association link between a specific Datastream and a specific Observation

9.6.11. Nested paths

URL Pattern: `<Service Root>/v2.0/EntityTypeName(<primaryKey>)/EntitySetNavigationAttribute(<primaryKey>)`

The pattern Clause 9.6.9 and Clause 9.6.5 can be combined to address a specific entity from a related set. This will return a `Not Found` error when the requested entity is not actually in the related set.

```
<Service Root>/v2.0/Datastreams(4)/Observations(5321)
```

Listing 38 — Resource pattern for a specific Observation of a specific Datastream

The above example addresses the same entity as `<Service Root>/v2.0/Observations(5321)`, except when Observation `5321` is not actually contained in Datastream `4`, since in that case any action on the resource would return a `Not Found` error.

9.7. Request Types

To the resources paths above the following request types can be sent:

Read	Read data from the resource path.
Create	Create a new Entity, or new Entities, at the resource path.
Update	Change the Entity or Entities at the resource path. Only the attributes present in the request are updated.
Replace	Replace all the data of the resource with different data.
Delete	Remove the resource.
Execute	Execute an function on a resource.

NOTE: The `Execute` request type is not used in the core STA API but may be used in extensions.

9.8. Additional request parameters

Besides the query options, there are a few other request parameters that are used to convey information between client and server.

Content-Type	Used to indicate the type of content that is being transferred.
Prefer	Analogous to IETF RFC 7240, used by a client to indicate how the server should respond to modification requests.
Location	Used by the server to convey the self-link of a newly created entity.

9.9. Requesting Data

9.9.1. Introduction

This section describes how to read data from an OGC SensorThings API compatible service. Details on how to send requests and how to receive responses are described in the protocol bindings sections. For example, when using the HTTP bindings, GET is used for read requests.

Adapted from:

- https://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part1-protocol.html
- https://docs.oasis-open.org/odata/odata/v4.01/odata-v4.01-part2-url-conventions.html

REQUIREMENTS CLASS 3: REQUESTING DATA	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
OBLIGATION	requirement
TARGET TYPE	Interface Binding
CONFORMANCE CLASS	Conformance class A.6: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/read`
NORMATIVE STATEMENTS	Requirement 10: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/paths`

REQUIREMENTS CLASS 3: REQUESTING DATA	
	Requirement 11: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/precedence Requirement 12: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/select Requirement 13: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/select_distinct Requirement 14: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/expand Requirement 15: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/top Requirement 16: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/skip Requirement 17: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/count Requirement 18: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/orderby Requirement 19: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/filter Requirement 20: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/format Requirement 21: http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/pagination

REQUIREMENT 10: REQUESTING DATA	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/api/read/paths
INCLUDED IN	Requirements class 3: http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read
STATEMENT	A service SHALL support all the read requests listed in section Clause 9.9.2.

9.9.2. Read Paths

9.9.2.1. Context URL

Most responses to read requests contain a context URL that describes the content of the payload. The context URL consists of the URL of the service metadata document (See Clause 9.6.3, Clause 9.4) followed by a fragment that identifies the relevant portion of the document.

A read Requests to the Service Document URL pattern (Clause 9.6.3) returns the service metadata document as described in Clause 9.4.

9.9.2.2. Service Document

A read Requests to the Service Document URL pattern (Clause 9.6.2) returns the service document. The Service document is a JSON object with attributes named `@context`, `value`, `serverSettings`.

The `@context` attribute SHALL contain the Context URL (Clause 9.9.2.1): The URL of the OData context document that describes the data model and entry points of the service.

The content of the attribute named `value` SHALL be a JSON Array containing one element for each entity set of the STA Service. Each element of the `value` array SHALL be a JSON object with at least two name/value pairs. One element has the name `name` containing the name of the entity set (e.g., Things, Locations, etc.) and one element has the name `url` containing the URL of the entity set, which may be an absolute or a relative URL.

Note: Adapted from OASIS OData-JSON-Format-v4.01, Section 5

The content of the attribute named `serverSettings` SHALL be a JSON Object describing the features the server supports that can not easily be detected by querying the service. The serverSettings object SHALL contain the attribute `conformance` of the type Array. It contains the URIs of all requirements classes and optional requirements that the service implements. Security extensions can modify the list of requirements to only show those requirements that the user is allowed to use. For example, if a user is not allowed to delete entities, the security extension can hide the create-update-delete/delete-entity requirement. In the most extreme case, a security extension would hide all requirements for a user that is not authenticated, except its own requirement and the instructions on how to authenticate.

The serverSettings object SHALL contain the attribute `functions` of the type Array, containing the function names of the filter functions that the service implements.

Extensions that need to expose additional server settings may do so in a attribute of the serverSettings object that is named after the conformance class URI of the requirement that defines this setting.

```
{
  "@context": "<Service Root>/v2.0/$metadata",
  "value": [
    {
      "name": "Things",
      "url": "<Service Root>/v2.0/Things"
    },
    {
      "name": "Locations",
      "url": "<Service Root>/v2.0/Locations"
    },
    {
      "name": "Datastreams",
      "url": "<Service Root>/v2.0/Datastreams"
    },
    {
      "name": "Sensors",
```

```
      "url": "<Service Root>/v2.0/Sensors"
    },
    {
      "name": "Observations",
      "url": "<Service Root>/v2.0/Observations"
    },
    {
      "name": "ObservedProperties",
      "url": "<Service Root>/v2.0/ObservedProperties"
    },
    {
      "name": "Features",
      "url": "<Service Root>/v2.0/Features"
    },
    {
      "name": "FeatureTypes",
      "url": "<Service Root>/v2.0/FeatureTypes"
    }
  ],
  "serverSettings": {
    "conformance": [
      "http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core",
      "http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read",
      "http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud",
      "http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/
select_distinct",
      "http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/deep_update",
      "http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/filtered_
delete",
      "http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/json_patch",
      "http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/replace",
      "http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http",
      "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/request_
response",
      "http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/mqtt",
      "http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/request_
response",
      "http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub",
      "http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/
select",
      "http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/
expand",
      "http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/
filter",
      "http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/simple_
create"
    ],
    "functions": [
      "any","cast","ceiling","concat","endswith","floor","floor","geo.
distance","geo.intersects","geo.length",
      "indexof","interval","length","now","round","round","startswith","st_
contains","st_crosses","st_disjoint",
      "st_equals","st_intersects","st_overlaps","st_relate","st_touches","st_
within","substring","substringof",
      "tolower","toupper","trim"
    ],
    "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http": {
      "endpoints": [
        "https://server.example.org/service/path/v2.0"
      ]
    },
    "http://www.opengis.net/spec/sensorthings/2.0/req/bindings/mqtt": {
      "endpoints": [
```

```
        "mqtt://server.example.org:1833",
        "wss://server.example.org/sensorThings",
      ]
    }
  }
}
```

Listing 39 — An example service document.

9.9.2.3. EntitySet

When a Read request is made to a URI pattern that addresses an entity set (Clause 9.6.4, Clause 9.6.9), the service returns a JSON object with the attributes `@context` and `value` and optionally `@count` and `@nextLink`. The `@context` annotation SHALL be a context URI as described in Clause 9.9.2.1. The value of the `value` attribute SHALL be an array containing entities in the specified entity set, or an empty array if there are no entities in the addressed set. The `@count` annotation SHALL, if returned by default, or requested explicitly, contain the total number of items in the set that match the request, as described in Clause 9.9.3.6. If service-driven pagination is in effect, the `@nextLink` annotation SHALL contain a link to the next set of entities, as described in Clause 9.9.3.12.

Read requests to an entity set can use all query options: `$filter`, `$count`, `$orderby`, `$skip`, `$top`, `$expand`, `$select` and `$format`.

```
{
  "@context": "<Service Root>/v2.0/$metadata#ObservedProperties",
  "@count": 36,
  "value": [
    {
      "@id": "<Service Root>/v2.0/ObservedProperties(1)",
      "id": 1,
      "name": "SO2",
      "definition": "http://dd.eionet.europa.eu/vocabulary/aq/pollutant/1",
      "description": "SO2",
      "properties": {
        "eionetId": 1,
        "owner": "http://dd.eionet.europa.eu",
        "recommendedUnit": "µg/m3"
      },
      "Datastreams@navigationLink": "<Service Root>/v2.0/ObservedProperties(1)/
Datastreams"
    },
    {
      "@id": "<Service Root>/v2.0/ObservedProperties(2)",
      "id": 2,
      "name": "PM2.5",
      "definition": "http://dd.eionet.europa.eu/vocabulary/aq/pollutant/6001",
      "description": "PM2.5",
      "properties": {
        "eionetId": 6001,
        "owner": "http://dd.eionet.europa.eu",
        "recommendedUnit": "µg/m3"
      },
      "Datastreams@navigationLink": "<Service Root>/v2.0/ObservedProperties(2)/
Datastreams"
    }, { … }, { … }, { … }
  ],
  "@nextLink": "<Service Root>/v2.0/ObservedProperties?$top=5&$skip=5"
```

```
}
```

Listing 40 — Possible result from a read request to an Entity Set resource returning ObservedProperties, with $top=5 and $count=true

9.9.2.4. Single Entity

When a Read request is made to a URI pattern that addresses a single entity (Clause 9.6.5, Clause 9.6.8), the service returns a JSON object representing the entity, with the added attributes `@context`. The `@context` annotation SHALL be a context URI as described in Clause 9.9.2.1.

Read requests to a URI pattern that returns a single entity can use the `$expand`, `$select` and `$format` query options.

```
{
  "@context": "<Service Root>/v2.0/$metadata#Things/$entity",
  "@id": "<Service Root>/v2.0/Things(1)",
  "id": 1,
  "name": "Oven",
  "description": "This thing is an oven.",
  "properties": {
    "owner": "Noah Liang",
    "color": "Black"
  },
  "HistoricalLocations@navigationLink": "<Service Root>/v2.0/Things(1)/
HistoricalLocations",
  "Locations@navigationLink": "<Service Root>/v2.0/Things(1)/Locations",
  "Datastreams@navigationLink": "<Service Root>/v2.0/Things(1)/Datastreams"
}
```

Listing 41 — Possible result from a request to an Entity resource returning a Thing

9.9.2.5. Single Entity Attribute

When a Read request is made to a URI pattern that addresses a single entity attribute (Clause 9.6.6), the service returns the JSON representation of this entity attribute.

```
{
  "@context": "<Service Root>/v2.0/$metadata#Edm.String",
  "value": "Oven"
}
```

Listing 42 — Possible result from a request to an Entity resource returning the name of a Thing

9.9.2.6. Raw Value of a Single Entity Attribute

When a Read request is made to a URI pattern that addresses the raw value of a single entity attribute (Clause 9.6.7), the service returns the entity attribute.

Listing 43 — Possible result from a request to an Entity resource returning the raw value of the name of a Thing

9.9.2.7. Relation Reference

When a Read request is made to a URI pattern that addresses a relation reference (Clause 9.6.10), the service responds with the entity-id(s) of the target Entity(s).

```
{
  "@context": "<Service Root>/v2.0/$metadata#$ref",
  "@id": "<Service Root>/v2.0/Things(42)"
}
```

Listing 44 — Response to a read request for <Service Root>/v2.0/Datastream(10643)/Thing

```
{
  "@context": "http://host/service/$metadata#Collection($ref)",
  "value": [
    { "@id": "<Service Root>/v2.0/Datastreams(10643)" },
    { "@id": "<Service Root>/v2.0/Datastreams(10759)" }
  ]
}
```

Listing 45 — Response to a read request for <Service Root>/v2.0/Things(42)/Datastreams

```
{
  "@context": "<Service Root>/v2.0/$metadata#$ref",
  "@id": "<Service Root>/v2.0/Locations(1)"
}
```

Listing 46 — Response to a read request for <Service Root>/v2.0/Things(1)/Locations(1)

9.9.3. Request Query Options

9.9.3.1. Introduction

Read requests can be modified using various query options. Query options are added to the URL of the request, as specified in Clause 9.6.

9.9.3.2. Evaluation Order

The STA Standard adapts many of OData's system query options and their usage. These query options support 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
- $format

REQUIREMENT 11: REQUESTING DATA	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/precedence
INCLUDED IN	Requirements class 3: http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read
STATEMENT	A service SHALL evaluate the query options supported, in the order defined in section Clause 9.9.3.2.

9.9.3.3. $select

The $select system query option requests the service to return only the attributes 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 attribute name (including navigation attribute names). For navigation attributes, $select controls the inclusion of the navigationLink in the response.

In the response, the service SHALL return the specified content, if available. Expanded navigation attributes do not need to be added to the $select list, they SHALL always be included in the response. The $select option can be applied to any request that returns an Entity or an EntitySet.

Note: Adapted from OASIS OData-Part1, Section 11.2.5.1

```
v2.0/Things?$select=id,name
```

Listing 47 — Resource pattern returning only the id and name of the Entities in the Things EntitySet.

REQUIREMENT 12: SELECT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/select`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
STATEMENT	A service SHALL support the `$select` query option as described in section Clause 9.9.3.3.

9.9.3.4. $select distinct

It is quite useful to give Entities common attributes, like a “type”. But when filtering on such a common attribute the client needs to know what the used values are. Distinct select allows a client to request all distinct values for a field or a set of fields.

Distinct select can be used in expands and can be ordered. When combining $orderby with a distinct select, it is only possible to order by the exact fields that are selected.

Note that selecting distinct values for the primary key field (`id`) makes no sense, since this field is unique for each entity.

To request the distinct values for a set of selected fields, add the `distinct:` keyword at the start of the `$select` parameter.

The returned data is formatted just like a non-distinct request of the same type would be.

```
v2.0/Things?$select=distinct:properties/type
```

Listing 48 — request returning all distinct values of the properties/type field of all Things

```
{
    "value": [
        { "properties": { "type": "waterBody" } },
        { "properties": { "type": "station" } },
        { "properties": { "type": "aquifer" } }
    ]
}
```

Listing 49 — possible response to the above request

REQUIREMENT 13: SELECT DISTINCT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/select_distinct`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
STATEMENT	If a service advertises this requirement in the service document then the service SHALL support the `distinct:` keyword in the `$select` query option as described in section Clause 9.9.3.4.

9.9.3.5. $expand

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

Expanded navigation attributes do not need to be added to `$select`, they are implicitly selected.

The `$expand` option can be applied to any request that returns an Entity or an EntitySet.

Note: Adapted from OASIS OData-Part1, Section 11.2.5.2

```
v2.0/Things?$expand=Datastreams($expand=ObservedProperty)
```

Listing 50 — Resource pattern returning Things, with their Datastreams, and the ObservedProperty for each Datastream.

```
v2.0/Datastreams?$expand=Observations($select=result,phenomenonTime;$orderby=
phenomenonTime desc;$top=1),ObservedProperty
```

Listing 51 — Resource pattern returning Datastream as well as the result and phenomenonTime of the last Observation (as ordered by phenomenonTime) and the ObservedProperty associated with this Datastream.

REQUIREMENT 14: EXPAND	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/expand`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
STATEMENT	A service SHALL support the `$expand` query option as described in section Clause 9.9.3.5.

9.9.3.6. $top

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

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

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.

Note: Adapted from OASIS OData-Part2, Section 11.2.6.3

```
v2.0/Things?$top=5
```

Listing 52 — Resource pattern returning only the first five entities in the Things EntitySet.

```
v2.0/Observations?$top=5&$orderby=phenomenonTime desc
```

Listing 53 — Resource pattern returning the first five Observation entries after sorting by the phenomenonTime attribute in descending order.

REQUIREMENT 15: TOP	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/top`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
STATEMENT	A service SHALL support the `$top` query option as described in section Clause 9.9.3.6.

9.9.3.7. $skip

The `$skip` system query option specifies the number for the items of the queried EntitySet that SHALL be excluded from the result. The value of `$skip` system query option SHALL be a non-negative integer n. The service SHALL return items starting at position n+1. 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 OASIS OData-Part1, Section 11.2.6.4

```
v2.0/Things?$skip=5
```

Listing 54 — Resource pattern returning Thing entities starting with the sixth Thing entity in the Things EntitySet.

```
v2.0/Observations?$skip=2&$top=2&$orderby=resultTime
```

Listing 55 — Resource pattern returning the third and fourth Observation entities from the collection of all Observation entities when the collection is sorted by the resultTime attribute in ascending order.

REQUIREMENT 16: SKIP	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/skip`

REQUIREMENT 16: SKIP	
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
STATEMENT	A service SHALL support the `$skip` query option as described in section Clause 9.9.3.7.

9.9.3.8. $count

The `$count` system query option with a value of `true` specifies that the total count of items within an EntitySet matching the request SHALL be returned along with the result. A `$count` query option with a value of `false` specifies that the service SHALL not return a count.

The service SHALL reject the request and inform the client that the request is malformed 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 in line 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.

For requests to an EntitySet, the count is returned in the `@count` attribute of the returned object. When a count is requested on an expanded EntitySet, the count is returned in the `<navigationlink name>@count` attribute of the containing Entity.

Note: Adapted from OASIS OData-Part1, Section 11.2.6.5

REQUIREMENT 17: COUNT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/count`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
STATEMENT	A service SHALL support the `$count` query option as described in section Clause 9.9.3.8.

9.9.3.9. $orderby

The `$orderby` system query option specifies the order in which items are returned from the service. The value of the `$orderby` system query option 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 attribute path terminating on a primitive attribute.

The expression MAY include the suffix `asc` for ascending or `desc` for descending, separated from the attribute name by one or more spaces. If asc or desc is not specified, the service SHALL order by the specified attribute 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 OASIS OData-Part1, Section 11.2.6.2

```
v2.0/Observations?$orderby=result
```

Listing 56 — Resource pattern returning all Observations ordered by the result attribute in ascending order.

```
v2.0/Observations?$orderby=Datastreams/id desc, phenomenonTime
```

Listing 57 — Resource pattern returning all Observations ordered by the id attribute of the linked Datastream entry in descending order, then by the phenomenonTime attribute of Observations in ascending order.

REQUIREMENT 18: ORDER BY	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/orderby`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
STATEMENT	A service SHALL support the `$orderby` query option as described in section Clause 9.9.3.9.

9.9.3.10. $filter

9.9.3.10.1. Introduction

The `$filter` option can be used to filter the entities returned by a request to any EntitySet. 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 attributes that are unavailable due to permissions, SHALL be omitted from the response.

Note: Adapted from OASIS OData-Part2, Section 5.1.1

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

values. Encoding rules for constants are listed in Table 27 Attributes of Entities are addressed by their name.

Note: Adapted from OASIS OData-Part1, Section 11.2.6.1

```
v2.0/Datastreams(42)/Observations?$filter=result gt 5
```

Listing 58 — Observations of Datastream 42 that have a result greater than 5.

```
v2.0/Locations?$filter=st_within(location, geometry'POLYGON ((30 10, 10 20, 20
40, 40 40, 30 10))')
```

Listing 59 — Locations inside a given area.

Sub-proprties of complex attributes are addressed using the complex attribute name, followed by a `/`, followed by the sub-attribute name.

```
v2.0/Observations?$filter=validTime/start lt 2012-12-03T07:16:23Z
```

Listing 60 — Observations with a valid time that started before the given timestamp.

Entities can be filtered on attributes of related entities by using the navigationAttribute, followed by a `/`, followed by an attribute of the related entity. This can be used recursively. For filtering on attributes of entities in a related EntitySet see the `any()` function.

```
v2.0/Observations?$filter=Datastream/name eq 'Temperature'
```

Listing 61 — Observations of all Datastreams that are named Temperature.

```
v2.0/Observations?$filter=Datastream/Thing/name eq 'House 1'
```

Listing 62 — Observations of all Things that are named House 1.

9.9.3.10.2. Built-in filter operations

The STA Standard supports a set of built-in filter operations, as described in the following table. These built-in filter operator usages and definitions follow OASIS OData-Part1, Section 11.2.6.1.1. The operator precedence is described in OASIS OData-Part2, Section 5.1.1.17.

Table 28 — Built-in Filter Operators

OPERATOR	DESCRIPTION	EXAMPLE
Comparison Operators		
`eq`	Equal	`/Datastreams?$filter=resultType/type eq 'Quantity'`
`ne`	Not equal	`/Datastreams?$filter=resultType/type ne 'Quantity'`
`gt`	Greater than	`/Observations?$filter=result gt 20.0`

OPERATOR	DESCRIPTION	EXAMPLE
`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?$filter=result le 100`
`in`	Containment	• `/Things?$filter=properties/type in ('Room','Corridor')` Where `properties/type` is a string. • `/Things?$filter='Vehicle' in properties/tags` Where `proprties/tags` is an array of strings.
Logical Operators		
`and`	Logical and	`/Observations?$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?$filter=result add 5 gt 10` • `/Observations?$filter=validTime gt now() add duration'PT1H'`
`sub`	Subtraction	• `/Observations?$filter=result sub 5 gt 10` • `/Observations?$filter=phenomenonTime gt now() sub duration'P1D'`
`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) mul 2 gt 10`

9.9.3.10.3. Built-in query functions

The STA Standard specifies 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 OASIS OData-Part2, Section 5.1.1.4.

In order to support spatial relationship functions, the STA Standard 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 Standard OGC 06-104r4, Clause 6.1.2.3. The names of these nine functions start with a prefix `st_` following the OGC Simple Feature Access Standard. In addition, the Well-Known Text (WKT) format is the default input geometry for these nine functions.

The results of the time and date functions are time zone dependent. Therefore these functions have an optional second argument to specify the time zone to use for the conversion.

In some cases, the service may not have enough information to deduce the types of parameters used in functions. This may, for instance, happen when combining two values from JSON objects, or attributes of type `ANY`. In such cases, the `cast` function can be used.

Table 29 — Built-in Query Functions

FUNCTION	DEFINITION	EXAMPLE
String Functions		
contains	`bool contains(string p0, string p1)`	`contains(description, 'SensorThings')`
substring	`bool substringof(string p0, string p1)`	`substringof('SensorThings',description)`
endswith	`bool endswith(string p0, string p1)`	`endswith(description,'Things')`
startswith	`bool startswith(string p0, string p1)`	`startswith(description,'Sensor')`
length	`int length(string p0)`	`length(description) eq 13`
indexof	`int indexof(string p0, string p1)`	`indexof(description,'Sensor') eq 1`
substring	• `string substring(string p0, int p1)` • `string substring(string p0, int p1, int p2)`	• `substring(description,1) eq 'ensor Things'` • `substring(description,2,4) eq 'nsor'`
tolower	`string tolower(string p0)`	`tolower(description) eq 'sensorthings'`

FUNCTION	DEFINITION	EXAMPLE
toupper	string toupper(string p0)	toupper(description) eq 'SENSORTHINGS'
trim	string trim(string p0)	trim(description) eq 'SensorThings'
concat	string concat(string p0, string p1)	concat(concat(unitOfMeasurement/symbol,', '), unitOfMeasurement/name) eq 'degree, Celsius'
Date Functions		
now	Edm.DateTimeOffset now()	resultTime ge now()
interval	TM_Interval interval(start, end or duration)	interval(properties/referenceTime, duration'PT1H')
year	• Edm.Int32 year(Edm.DateTimeOffset t) • Edm.Int32 year(Edm.DateTimeOffset t, String zoneId)	• year(resultTime) eq 2015 • year(resultTime, 'CET') eq 2015
month	• Edm.Int32 month(Edm.DateTimeOffset t) • Edm.Int32 month(Edm.DateTimeOffset t, String zoneId)	• month(resultTime) eq 12 • month(resultTime, '-09:00') eq 12
day	• Edm.Int32 day(Edm.DateTimeOffset t) • Edm.Int32 day(Edm.DateTimeOffset t, String zoneId)	• day(resultTime) eq 8 • day(resultTime, 'America/Santiago') eq 8
hour	• Edm.Int32 hour(Edm.DateTimeOffset t) • Edm.Int32 hour(Edm.DateTimeOffset t, String zoneId)	• hour(phenomenonTime/start) eq 1 • hour(phenomenonTime/start, 'EDT') eq 1
minute	• Edm.Int32 minute(Edm.DateTimeOffset t) • Edm.Int32 minute(Edm.DateTimeOffset t, String zoneId)	• minute(resultTime) eq 0 • minute(resultTime, Datastream/properties/timeZone) eq 0
second	• Edm.Int32 second(Edm.DateTimeOffset t) • Edm.Int32 second(Edm.DateTimeOffset t, String zoneId)	• second(resultTime) eq 0 • second(resultTime, 'Asia/Singapore') eq 0
date	• Edm.Date date(Edm.DateTimeOffset t)	• date(resultTime) ne date(validTime) • date(resultTime) ne date(validTime, '+10')

FUNCTION	DEFINITION	EXAMPLE
	• Edm.Date date(Edm.DateTimeOffset t, String zoneId)	
time	• Edm.TimeOfDay time(Edm.DateTimeOffset t) • Edm.TimeOfDay time(Edm.DateTimeOffset t, String zoneId)	• time(resultTime) le validTime • time(resultTime, '+5:00') le validTime
totaloffs	• Edm.Int32 totaloffsetminutes(Edm DateTimeOffset t) • Edm.Int32 totaloffsetminutes(Edm DateTimeOffset t, String zoneId)	• totaloffsetminutes(resultTime) eq 60 • totaloffsetminutes(resultTime, '-03:30') eq 60
fractiona	• Edm.Int32 fractionalseconds(Edm DateTimeOffset t) • Edm.Int32 fractionalseconds(Edm DateTimeOffset t, String zoneId)	• fractionalseconds(resultTime) eq 0 • fractionalseconds(resultTime, 'CEST') eq 0
Math Functions		
round	int round(number p0)	round(result) eq 32
floor	int floor(number p0)	floor(result) eq 32
ceiling	int ceiling(number p0)	ceiling(result) eq 33
Geospatial Functions		
geo.distance	• double geo.distance(geometry p0, geometry p1) • double geo.distance(geography p0, geography p1)	geo.distance(location, geometry'POINT (30 10)')
geo.length	double geo.length(LineString p0)	geo.length(geometry'LINESTRING (30 10, 10 30, 40 40)')
geo.intersec	• bool geo.intersects(geometry p0, geometry p1) • bool geo.intersects(geography p0, geography p1)	geo.intersects(location, geometry'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))')
Spatial Relationship Functions		

FUNCTION	DEFINITION	EXAMPLE
st_equals	• bool st_equals(geometry p0, geometry p1) • bool st_equals(geography p0, geography p1)	st_equals(location, geometry'POINT (30 10)')
st_disjoint	• bool st_disjoint(geometry p0, geometry p1) • bool st_disjoint(geography p0, geography p1)	st_disjoint(location, geometry'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))')
st_touches	• bool st_touches(geometry p0, geometry p1) • bool st_touches(geography p0, geography p1)	st_touches(location, geometry'LINESTRING (30 10, 10 30, 40 40)')
st_within	• bool st_within(geometry p0, geometry p1) • bool st_within(geography p0, geography p1)	st_within(location, geometry'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))')
st_overlaps	• bool st_overlaps(geometry p0, geometry p1) • bool st_overlaps(geography p0, geography p1)	st_overlaps(location, geometry'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))')
st_crosses	• bool st_crosses(geometry p0, geometry p1) • bool st_crosses(geography p0, geography p1)	st_crosses(location, geometry'LINESTRING (30 10, 10 30, 40 40)')
st_intersect	• bool st_intersects(geometry p0, geometry p1) • bool st_intersects(geography p0, geography p1)	st_intersects(location, geometry'LINESTRING (30 10, 10 30, 40 40)')
st_contains	• bool st_contains(geometry p0, geometry p1) • bool st_contains(geography p0, geography p1)	st_contains(location, geometry'POINT (30 10)')
st_relate	• bool st_relate(geometry p0, geometry p1, string p2) • bool st_relate(geography p0,	st_relate(location, geometry'POLYGON ((30 10, 10 20, 20 40, 40 40, 30 10))', 'T********')

FUNCTION	DEFINITION	EXAMPLE
	geography p1, string p2)	
Collection Functions		
any	bool EntitySetNavProp/any(name: expression)	Observations/any(o: o/result gt 5 and o/phenomenonTime gt 2024-01-01T00:00:00Z)
Type Functions		
cast	type cast(expression, typeName)	cast(result, Edm.Decimal) gt cast(Datastream/properties/threshold, Edm.Decimal)

9.9.3.10.4. Any

The `any()` function applies a Boolean expression to each member of a collection and returns true if the expression returns true for any member of the collection. The `any` function can be used by adding a slash and the `any` keyword after a filter path ending in an EntitySet. The argument of the any function is a case-sensitive, alphanumeric lambda variable name, followed by a colon (`:`), followed by a Boolean expression that can use the defined variable to access attributes of the entities in the collection.

Path expressions not prefixed by the lambda variable are evaluated in the context of the collection that is the target of the `$filter` containing the `any` function.

```
Datastreams?$filter=Observations/any(o: o/result gt properties/threshold and o/phenomenonTime gt 2024-01-01T00:00:00Z)
```

Listing 63 — Request for all Datastreams that have at least one Observation that has both a result greater than the threshold custom attribute in the containing Datastream, and a phenomenonTime after 2024-01-01 00:00:00Z

In the example above, the `properties/threshold` path is evaluated in the context of the Datastreams collection being filtered.

```
Things?$filter=Datastreams/any(d1: d1/ObservedProperty/name eq 'NO2') and Datastreams/any(d2: d2/ObservedProperty/name eq 'O3')
```

Listing 64 — Request for all Things that have both a Datastream measuring NO2 and a Datastream measuring O3

REQUIREMENT 19: FORMAT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/filter`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`

REQUIREMENT 19: FORMAT	
A	A service SHALL support the `$filter` query option as described in section Clause 9.9.3.10.
B	A service SHALL list the supported filter function in the service document as described in section Clause 9.9.2.2.

9.9.3.11. $format

9.9.3.11.1. Introduction

The format of the data returned by read request can be controlled with the `$format` query option. The default value for the `$format` option is `application/json`.

REQUIREMENT 20: FORMAT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/format`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
STATEMENT	A service SHALL support the `$format` query option as described in section Clause 9.9.3.11 and the JSON format described in Clause 9.9.3.11.2.

9.9.3.11.2. application/json

This format encodes responses in JSON as specified in OData JSON Format Version 4.01. The JSON format can be explicitly requested using the `$format` query option with a value of `application/json`, or the shorter alias `json`. The format option allows the format parameter `metadata` to control the amount of service metadata added to the response.

- `$format=application/json;metadata=full` returns the full metadata.
- `$format=application/json;metadata=minimal` removes all metadata that can be calculated by the client, returning only: context, count (if requested) and nextLink (if applicable)
- `$format=application/json;metadata=none` only returns: nextLink (if applicable) and count (if requested)

9.9.3.12. Server driven pagination

Responses that include only a partial set of the items identified by the request URL SHALL contain a link that allows retrieving the next partial set of items. This link is called a nextLink; its representation is format-specific. The final partial set of items (the last page) 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.

A STA 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 disallow a change of format on requests for subsequent pages using the next link.

Note: Adapted from OASIS OData-Part1, Section 11.2.6.7

REQUIREMENT 21: FORMAT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/pagination`
INCLUDED IN	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
STATEMENT	A service SHALL support pagination as described in section Clause 9.9.3.12.

9.10. Modifying Data

9.10.1. Introduction

This section describes how to create, update and delete Entities. Details on how to send create, update and delete requests and how responses are received are described in the protocol bindings sections.

The content type of data sent to a service must be `application/json`, unless otherwise specified.

REQUIREMENTS CLASS 4: MODIFYING DATA

IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud
OBLIGATION	requirement
TARGET TYPE	Interface Binding
CONFORMANCE CLASS	Conformance class A.7: http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/cud
NORMATIVE STATEMENTS	Requirement 22: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/advertisement Requirement 23: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/create Requirement 24: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/deep_insert Requirement 25: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/update Requirement 26: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/replace Requirement 27: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/deep_update Requirement 28: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/json_patch Requirement 29: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/return_value Requirement 30: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/update_relations Requirement 31: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/delete Requirement 32: http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/filtered_delete

REQUIREMENT 22: CREATE, UPDATE, DELETE

IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/advertisement
INCLUDED IN	Requirements class 4: http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud
STATEMENT	A service that supports data modification SHALL advertise the requirements class http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud in its service document.

9.10.2. Creating Entities

Entities can be created with a create request to an URI pattern that addresses an entity set (Clause 9.6.4 or Clause 9.6.9). The content of the "create" request is the JSON encoded representation of the entity to be created.

If the inline representation contains a value for a computed attribute (i.e., `id`), the service ignores that value when creating the entity.

When the URI pattern is of the related entity set type (Clause 9.6.9), the created entity is automatically linked to the parent entity of the entity set the create request is sent to.

Relations to existing entities can be created by providing the entity-id (selfLink) of the existing entities. If a conflicting value for a relation exists in the URI and the content of the request, the value from the URI takes priority.

```
{
  "name": "Oven temperature",
  "description": "This is a datastream measuring the air temperature in an
oven.",
  "resultType": {
    "type": "Quantity",
    "label": "Oven temperature",
    "definition": "ObservedProperties(1)",
    "uom": { "code": "Cel", "label": "degree Celsius", "symbol": "°C" }
  },
  "Sensor": {"@id": "Sensors(2)"}
}
```

Listing 65 — Sending a create request to `v2.0/Things(5)/Datastreams` with the given content will create a Datastream, linked to Thing 5, Sensor 2 and the ObservedProperty that matches the definition in the resultType.

REQUIREMENT 23: CREATING ENTITIES	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/create`
INCLUDED IN	Requirements class 4: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud`
STATEMENT	A service that supports data modification SHALL support creating entities as described in section Clause 9.10.2.

9.10.3. Deep Insert

A "create" request can have related entities embedded in-line, to be created alongside the main entity. Such a "create" request is referred to as a "deep insert".

On success, the service creates all entities and relates them. On failure, the service does not create any of the entities.

```
{
  "description": "This an oven with a temperature datastream.",
  "name": "oven",
  "Locations": [
    {
      "name": "CCIT",
      "description": "Calgary Centre for Innovative Technologies",
      "encodingType": "application/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.
",
      "resultType": {
        "type": "Quantity",
        "label": "Oven temperature",
        "definition": "ObservedProperties(1)",
        "uom": { "code": "Cel", "label": "degree Celsius", "symbol": "°C" }
      },
      "Sensor": {
        "name": "DS18B20",
        "description": "DS18B20 is an air temperature sensor…",
        "encodingType": "application/pdf",
        "metadata": "http://datasheets.maxim-ic.com/en/ds/DS18B20.pdf"
      }
    }
  ]
}
```

Listing 66 — Sending a "create" request to `v2.0/Things` with the given content will create a Thing with its Location and a Datastream with an associated Sensor. The Datastream will automatically be linked to the ObservedProperty that matches the definition in the resultType.

REQUIREMENT 24: DEEP INSERT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/deep_insert`
INCLUDED IN	Requirements class 4: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud`
STATEMENT	A service that supports data modification SHALL support creating entities as described in section Clause 9.10.3.

9.10.4. Updating Entities

Entities can be updated with an update request to an URI pattern that addresses a single entity (Clause 9.6.5 or Clause 9.6.8).

An update request must contain only those updatable attributes of the entity that are to be changed. Attributes that are not present in the update request are not changed, though server-generated attributes may be updated by the server as a result of the change request.

For even finer grained update control, or updates that are safe from concurrent modification by other users, see Clause 9.10.7.

```
{
  "name": "My Updated Oven"
}
```

Listing 67 — Sending an update request to `v2.0/Things(1)` with the given content will update the name of Thing 1, and leave all other fields unchanged.

REQUIREMENT 25: UPDATING ENTITIES	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/update`
INCLUDED IN	Requirements class 4: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud`
STATEMENT	A service that supports data modification SHALL support updating entities as described in section Clause 9.10.4.

9.10.5. Replacing Entities

Entities can be replaced with a replace request to an URI pattern that addresses a single entity (Clause 9.6.5 or Clause 9.6.8).

A “replace” request must contain as content the complete entity to be replaced. Entity attributes that are not present in the replace request are removed from the entity. Navigation attributes are not affected.

```
{
  "name": "My Updated Oven"
}
```

Listing 68 — Sending a replace request to `v2.0/Things(1)` with the given content will update the name of Thing 1, and remove all other fields.

REQUIREMENT 26: REPLACING ENTITIES	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/replace
INCLUDED IN	Requirements class 4: http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud
STATEMENT	If a service advertises this requirement in the service document then the service SHALL support replacing entities as described in section Clause 9.10.5.

9.10.6. Deep Update

The content of an update request may contain nested entities to be updated or created and entity references to be linked, that specify the full set of entities to be linked.

On success, the server makes all the changes. On failure, the server makes none of the changes.

The server must ensure data integrity and return an error if the update breaks integrity. For example, an update request that specifies fewer Datastreams on a Thing than the Thing currently has cannot be valid. This update would remove Datastreams from the Thing, but a Datastream MUST have a Thing. To remove a Datastream from a Thing, either the Datastream must be explicitly deleted, or linked to a different Thing.

```
{
  "name": "My Updated Oven",
  "Locations": [
    {
      "name": "New Location of my Oven",
      "description": "It fits much better here",
      "encodingType": "application/geo+json",
      "location": {
        "type": "Point",
        "coordinates": [-114.061,51.051]
      }
    },
    {
      "@id": "Locations(1)"
    }
  ]
}
```

Listing 69 — Sending an update request to `v2.0/Things(1)` with the given content will update the name of Thing 1, create a new Location, and set the Locations of the Thing to the newly created Location and existing Location 1, removing any old Location(s).

REQUIREMENT 27: DEEP UPDATE	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/deep_update
INCLUDED IN	Requirements class 4: http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud

REQUIREMENT 27: DEEP UPDATE	
STATEMENT	If a service advertises this requirement in the service document then the service SHALL support the deep updating of entities as described in section Clause 9.10.6.

9.10.7. JSON-Patch

Services MAY additionally support JSON PATCH format IETF RFC 6902 to express a sequence of operations to apply to an entity.

Entities can be updated with a json-patch request to an URI pattern that addresses a single entity (Clause 9.6.5 or Clause 9.6.8).

The content-type of the data in a json-patch request is `application/json-patch+json`.

The paths in the json-patch request must not contain navigation attributes.

```
[
  { "op": "test", "path": "/properties/status", "value": "inactive" },
  { "op": "replace", "path": "/properties/status", "value": "active" }
]
```

Listing 70 — Sending a json-patch request to `v2.0/Things(1)` with the given content will set the value of `properties/status` to `active` only if the current value of `properties/status` to `inactive` and return a `Conflict` error otherwise.

REQUIREMENT 28: JSON-PATCH	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/json_patch`
INCLUDED IN	Requirements class 4: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud`
STATEMENT	If a service advertises this requirement in the service document then the service SHALL support the updating of entities with JSON-Patch requests as described in section Clause 9.10.7.

9.10.8. Return Value

The returned value for an entity modification request depends on the `prefer` request parameter, with the possible values `return=minimal` (default) and `return=representation`.

If the `prefer` request parameter is not present, or does not contain a value for `return`, or has `return=minimal` then, for create requests, the entity-id (selfLink) of the created entity is returned as a value, and no content is returned. For update requests no content is returned.

If the `prefer` request parameter has `return=representation`, then the created or updated resource is returned, optionally taking `$expand`, `$select` and `$format` query options into account.

REQUIREMENT 29: RETURN VALUE	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/return_value
INCLUDED IN	Requirements class 4: http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud
STATEMENT	A service that supports data modification SHALL return data to modification requests as described in section Clause 9.10.8.

9.10.9. Modifying Relations

Single-valued navigation attributes (Clause 9.6.8) can be updated by sending a replace request to the association link (Clause 9.6.10). The "replace" request must contain an entity reference to the single entity that should be the new target of the navigation attribute.

```
replace http://host/service/Datastreams(1)/ProximateFeatureOfInterest/$ref
  {"@id": "Features(2)"}
```

Listing 71 — Example of a request setting the ProximateFeatureOfInterest of `Datastream 1` to `Feature 2`.

Single-valued navigation attributes can be un-linked by sending a delete request to the association link. This will not remove the entities, only remove the link between the entities.

```
delete http://host/service/Datastreams(1)/ProximateFeatureOfInterest/$ref
```

Listing 72 — Example of a request removing the ProximateFeatureOfInterest from `Datastream 1`.

EntitySet navigation attributes (Clause 9.6.9) can be extended with a new link to an existing entity by sending a create request to the association link of the navigation attribute. The "create" request contains an entity reference to the single entity to be added to the set of related entities. If the two entities are already related, the operation has no effect and returns a success.

```
create http://host/service/Features(1)/FeatureTypes/$ref
  {"@id": "FeatureTypes(3)"}
```

Listing 73 — Example of a request adding `FeatureType 3` as FeatureType of `Feature 1`.

The complete list of relations in an EntitySet-type navigation attribute can be replaced by sending a replace request to the association link of the navigation attribute. The replace request must contain a list of entity references, like the one returned by a read request to the association link of the navigation attribute.

```
replace http://host/service/Features(1)/FeatureTypes/$ref
  {"value": [
    {"@id": "FeatureTypes(1)"},
    {"@id": "FeatureTypes(2)"}
```

```
]}
```

Listing 74 — Example changing the set of `FeatureTypes` of `Feature 1` to contain exactly `FeatureType 1` and `FeatureType 2`.

A single relation can be removed from an EntitySet-type navigation attribute by either sending a delete request to the association link of the relation, or by sending a delete request to the association link of the navigation attribute, with an `$id` parameter specifying the self-link (absolute or relative) of the target Entity.

```
delete http://host/service/Features(1)/FeatureTypes(2)/$ref
delete http://host/service/Features(1)/FeatureTypes/$ref?$id=../../
FeatureTypes(2)
```

Listing 75 — Examples of the two ways to remove `FeatureType 2` from an EntitySet-typed navigation attribute `FeatureTypes` of `Feature 1`.

All relations can be removed from an EntitySet-type navigation attribute by sending a delete request to the association link of the navigation attribute.

```
delete http://host/service/Features(1)/FeatureTypes/$ref
```

Listing 76 — Example removing all `FeatureTypes` from `Feature 1`.

For any change of relations data integrity MUST be maintained. If a request breaks data integrity, then an INVALID REQUEST error is returned. On a successful change, the server returns an empty response.

REQUIREMENT 30: MODIFYING RELATIONS	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/update_relations`
INCLUDED IN	Requirements class 4: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud`
STATEMENT	A service that supports data modification SHALL support relation-modification requests as described in section Clause 9.10.9.

9.10.10. Deleting Entities

Entities can be removed from a service by sending a delete request to an URI pattern that addresses a single entity (Clause 9.6.5 or Clause 9.6.8).

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

The server must ensure data integrity and delete any entities that depend on the entity being deleted. For example, deleting a Thing would also delete the HistoricalLocations and Datastreams that depend on it, and the Observations in those Datastreams, but not the Locations of the Thing, since Locations can exist without a relation to a Thing.

REQUIREMENT 31: DELETING ENTITITES	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/delete
INCLUDED IN	Requirements class 4: http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud
STATEMENT	A service that supports data modification SHALL support delete requests as described in section Clause 9.10.10.

9.10.11. Deleting Entities by Filter

Sending individual delete requests for each entity to be deleted is very inefficient when many entities need to be deleted. For example for maintenance, when all Observations older than a certain date need to be deleted, a client would first have to send a read request to find all entities older than the threshold data, and then sent a delete request for each of the returned entities. To make deleting a large number of entities more efficient, a server may implement filtered-delete capability.

When filtered-delete is supported, a client can send a delete request to an URI pattern that addresses an entity set (Clause 9.6.4 or Clause 9.6.9) with the $filter option (Clause 9.9.3.10). The server will delete all entities from the set that match the filter.

REQUIREMENT 32: DELETING ENTITIES BY FILTER	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/filtered_delete
INCLUDED IN	Requirements class 4: http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud
STATEMENT	If a service advertises this requirement in the service document then the service SHALL support the mass-deleting of entities as described in section Clause 9.10.11.

10

HTTP INTERFACE BINDINGS

10 HTTP INTERFACE BINDINGS

10.1. Introduction

The HTTP Bindings specify how the abstract REST-API specified in the OGC SensorThings API is implemented over the HTTP protocol.

REQUIREMENTS CLASS 5: HTTP INTERFACE BINDINGS	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http`
OBLIGATION	requirement
TARGET TYPE	Interface Binding
CONFORMANCE CLASS	Conformance class A.1: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/binding/http`
NORMATIVE STATEMENTS	Requirement 33: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/advertisement` Requirement 34: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/request_response`

10.2. HTTP Binding Advertisement

To help a client find the HTTP endpoints of a STA service, the endpoints of the service are documented in the `serverSettings` object of the service documents of all request-response bindings, as described in Clause 9.9.2.2. If the service supports the HTTP interface binding, then the `serverSettings` object SHALL contain an attribute of type Object, with the name

`http://www.opengis.net/spec/sensorthings/2.0/req/binding/http`

This Object SHALL contain an attribute named `endpoints` of type Array. The JSON Array `endpoints` SHALL hold a list of URL Schemes that can be used to connect to the HTTP service.

REQUIREMENT 33: BINDING ADVERTISEMENT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/advertisement`
INCLUDED IN	Requirements class 5: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http`
STATEMENT	If a service implements any HTTP bindings, then the HTTP endpoints of the service SHALL be advertised in the service documents of all implemented request-response bindings, as described in Clause 10.2.

10.3. Request / Response

The actions that can be executed on resources, as defined in the Clause 9 are mapped to HTTP Methods as defined in table Table 30.

Table 30 — Mapping of abstract request types to HTTP Methods.

REQUEST TYPE	HTTP METHOD
Read	GET
Create	POST
Update	PATCH
Replace	PUT
Delete	DELETE
Execute	POST

Content for Create, Update, Replace or Execute requests is passed in the body of the HTTP request.

Request parameters (such as `prefer`, `content-type` and `location`) are passed as Headers.

HTTP HEAD and OPTIONS requests are treated as normal GET requests, but don't return data. If a GET request with the same URL and headers would return an error, the HEAD or OPTION requests also returns this error.

REQUIREMENT 34: REQUEST / RESPONSE	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/request_response
INCLUDED IN	Requirements class 5: http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http
PREREQUISITE	Requirements class 2: http://www.opengis.net/spec/sensorthings/2.0/req-class/api/abstract
STATEMENT	If a service advertises this requirement in the service document then the service SHALL map the request types as defined in the abstract STA API to HTTP methods as described in Clause 10.3.

10.4. WebSub Topic Discovery

This section defines the requirements for a STA HTTP interface supporting topic discovery as an extension to W3C REC-websub-20180123. With supporting WebSub Topic Discovery, asynchronous messaging over HTTP(S) can be established that allow any client to receive notification data from a STA endpoint based on self-defined topics including ODATA query parameters. The use of self-defined notification topics is a great improvement over PubSub, where topics are static and published by providers. With support for WebSub Topic Discovery, any client can determine if a particular topic — expressed as a STA HTTP URL — can be used for subscribing to future notifications.

NOTE: As defined in W3C REC-websub-20180123, building an Asynchronous Messaging System requires a WebSub Hub implementation that accepts subscriptions for self-discovered topics, subscribe to the STA MQTT broker and distribute the actual notifications to subscribed clients.

The `Discovery` requirements class is implemented at the STA service — HTTP interface to support the W3C WebSub discovery protocol.

REQUIREMENTS CLASS 6: DISCOVERY	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.2: http://www.opengis.net/spec/sensorthings/2.0/conf-class/binding/http/websub
PREREQUISITE	Requirement 34: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/request_response

REQUIREMENTS CLASS 6: DISCOVERY	
NORMATIVE STATEMENTS	Requirement 35: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/link-hub Requirement 36: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/link-self Requirement 37: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/link-help Requirement 38: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/odata-support Requirement 39: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/odata-discovery Requirement 40: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/odata-blacklisting Requirement 41: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/topics-discovery Requirement 42: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/topics-blacklisting

10.4.1. Link Header 'hub'

According to W3C REC-websub-20180123, discovery always returns `<Link>; rel="hub"` in response to HTTP GET and HEAD requests.

REQUIREMENT 35	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/link-hub
INCLUDED IN	Requirements class 6: http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub
A	An implementation SHALL return the `<Link>; rel="hub"` HTTP header in response to HTTP GET and HEAD requests to indicate support for WebSub.

10.4.2. Link Header 'self'

According to W3C REC-websub-20180123, discovery returns `<Link>; rel="self"` for a URL that is suitable for subscription.

REQUIREMENT 36	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/link-self

REQUIREMENT 36	
INCLUDED IN	Requirements class 6: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub`
A	An implementation SHALL return the `<Link>; rel="self"` HTTP header in response to HTTP GET and HEAD requests to indicate the URL can be used for subscription.
B	An implementation SHALL not return the `<Link>; rel="self"` HTTP header in response to HTTP GET and HEAD requests if the request URL either contains a denied topic or at least one denied ODATA option. In this case, the implementation SHALL return the `<Link>; rel="help"`.
C	An implementation SHALL determine if a request URL is allowed for subscription by first converting the URL into an MQTT topic and test whether the topic is included in the `topics_denied`. If so, the implementation SHALL not return the `<Link>; rel="self"` but the `<Link>; rel="help"` HTTP header in response to a GET or HEAD request.
D	An implementation SHALL isolate the query string from the request URL and check if any of the ODATA options is listed in the `odata_denied`. If so, the implementation SHALL not return the `<Link>; rel="self"` but the `<Link>; rel="help"` HTTP header in response to a GET or HEAD request.

10.4.3. Link Header 'help'

Returning the `<Link>; rel="help"` header (specific to STA-WebSub) indicates the cause why the requested URL is not suitable for subscription.

REQUIREMENT 37	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/link-help`
INCLUDED IN	Requirements class 6: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub`
A	An implementation SHALL provide a URL for obtaining help (why the request URL cannot be used for subscription) when returning the `<Link>; rel="help"` header.
B	An implementation SHALL use the `<Link>; rel="self"` and `<Link>; rel="help"` HTTP header mutually exclusive.

10.4.4. ODATA query options support

REQUIREMENT 38	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/odata-support`

REQUIREMENT 38	
INCLUDED IN	Requirements class 6: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub`
A	An implementation SHALL allow the discovery such that the associated MQTT topic is compliant to OGC SensorThings API v2.0 requirement regarding updates via MQTT with the following extensions: • URI patterns that addresses an entity set (Clause 9.6.4, Clause 9.6.9) can be extended with a ? followed by any valid combination of ODATA options that the server support, as defined in Clause 11.4.2, Clause 11.4.3 and Clause 11.4.4. • URI patterns that addresses an entity (Clause 9.6.5, Clause 9.6.8) can be extended with a ? followed by any valid combination of ODATA options that the server support, as defined in Clause 11.4.2, Clause 11.4.3. • URI pattern that addresses a single entity attribute (Clause 9.6.6).

10.4.5. ODATA support advertisement

REQUIREMENT 39	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/odata-discovery`
INCLUDED IN	Requirements class 6: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub`
A	An implementation SHALL advertise the support for ODATA query parameter by publishing the string `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub` in the `serverSettings` section of the root page.

10.4.6. ODATA Blacklisting

A request URL may contain query parameters (after the `?`) presenting ODATA options. An implementation may deny discovery (and thereby later subscription) for certain ODATA options like `$expand` or `$filter`.

REQUIREMENT 40	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/odata-blacklisting`
INCLUDED IN	Requirements class 6: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub`
A	An implementation SHALL advertise the existence of denied ODATA options by adding the following key to the root page: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub` that includes the key `odata_denied`. This key SHALL contain a JSON array for all denied ODATA options.

REQUIREMENT 40	
B	In case no ODATA options are denied,an implementation SHALL advertise the non-existence of denied ODATA options by adding the following key to the root page: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub that includes the key odata_denied. This key SHALL contain an empty JSON array.

```
{
    "serverSettings": {
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "odata_denied": <JSON Array of denied ODATA options>
        }
    }
}
```

Listing 77 — Example blacklisting structure for ODATA options.

```
{
    "serverSettings": {
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "odata_denied": ["$expand", "$skip", "$top", "$filter"]
        }
    }
}
```

Listing 78 — Example of a root page snippet indicating restrictions for `$expand`, `$skip`, `$top` and `$filter`

```
{
    "serverSettings": {
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "odata_denied": []
        }
    }
}
```

Listing 79 — Example of a root page snippet indicating no restrictions for ODATA options

10.4.7. Topics Blacklisting

The support for STA-WebSub can be determined from the root page as part of the `Discovery` conformance class.

REQUIREMENT 41	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/topics-discovery
INCLUDED IN	Requirements class 6: http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub

REQUIREMENT 41

A	An implementation SHALL advertise the support for topics blacklisting by publishing the string http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub in the serverSettings section of the root page that includes the key topics_denied.

```
{
    "serverSettings": {
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": <JSON Array of denied topics>
        }
    }
}
```

Listing 80 — Example blacklisting structure for topics

REQUIREMENT 42

IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/topics-blacklisting
INCLUDED IN	Requirements class 6: http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub
A	A STA topic starts with an URI patterns that addresses an entity set (Clause 9.6.4, Clause 9.6.9).
B	An implementation SHALL advertise the existence of denied topics by adding the following key to the root page: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub that includes the key topics_denied. This key SHALL contain a JSON array listing all denied topics.
C	In case no topics are denied, an implementation SHALL advertise the non-existence of denied topics by adding the following key to the root page: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub that includes the key topics_denied. This key SHALL contain an empty JSON array.

```
{
    "serverSettings": {
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": ["v2.0/Observations", "v2.0/Datastreams('very
chatty')/Observations"]
        }
    }
}
```

Listing 81 — Example of a root page snippet indicating deny for the topics `v2.0/Observations` and `v2.0/Datastreams('very chatty')/Observations`

```
{
    "serverSettings": {
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": []
        }
```

```
        }
    }
```

Listing 82 — Example of a root page snippet indicating no deny for topics

11

MQTT INTERFACE BINDINGS

11 MQTT INTERFACE BINDINGS

11.1. Introduction

The MQTT Bindings specify how to access a STA endpoint using the MQTT 5.0 protocol (OASIS mqtt-v5.0). This binding consists of three parts that can be implemented independently: Clause 11.3, Clause 11.4 and Clause 11.5.

REQUIREMENTS CLASS 7: MQTT INTERFACE BINDINGS	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/mqtt`
OBLIGATION	requirement
TARGET TYPE	Interface Binding
CONFORMANCE CLASS	Conformance class A.3: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/binding/mqtt`
NORMATIVE STATEMENTS	Requirement 43: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/advertisement` Requirement 44: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/request_response` Requirement 45: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub` Requirement 49: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/simple_create`

11.2. MQTT Binding Advertisement

To help a client find the MQTT endpoints of a STA service, the endpoints of the service are documented in the `serverSettings` object of the service documents of all request-response bindings, as described in Clause 9.9.2.2. If the service supports any MQTT interface bindings, then the `serverSettings` object SHALL contain a attribute of type Object, with the name

`http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt`

This Object SHALL contain a attribute named `endpoints` of type Array. The JSON Array `endpoints` SHALL hold a list of URL Schemes that can be used to connect to the MQTT service

REQUIREMENT 43: BINDING ADVERTISEMENT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/advertisement`
INCLUDED IN	Requirements class 7: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/mqtt`
STATEMENT	If a service implements any MQTT bindings, then the MQTT endpoints of the service SHALL be advertised in the service documents of all implemented request-response bindings, as described in Clause 11.2.

11.3. Request / Response

MQTT 5.0 formalizes a request-response pattern over the MQTT protocol (OASIS mqtt-v5.0, Section 4.10) that allows the use of the REST API over the MQTT protocol. See Listing 83 for a basic overview of the request-response pattern in MQTT 5.0.

MQTT 5.0 also defines User Properties (OASIS mqtt-v5.0, Section 3.3.2.3.7) that can be used to pass application-defined key-value pairs with a published message.

```
@startuml
  skinparam responseMessageBelowArrow true
  participant Client as client
  participant "MQTT5 Broker" as broker
  participant RequestHandler as rh

  == Setup RequestHandler ==
  rh -> broker: CONNECT
  rh <-- broker: CONNACK

  rh -> broker: SUBSCRIBE v2.0/$request
  rh <-- broker: ok

  == Setup Client ==

  client -> broker: CONNECT\nrequestResponseInformation=true
  client <-- broker: CONNACK\nResponseInformation:responses/userXyz

  client -> broker: SUBSCRIBE responses/userXyz
  client <-- broker: ok

  ...

  == Request-Response ==
  client -> broker: PUBLISH v2.0/$request\ncorrelationData=123xyz
\nresponseTopic=responses/userXyz
  client <-- broker: ok
```

```
  broker -> rh: PUBLISH v2.0/$request\ncorrelationData=123xyz\nresponseTopic=
responses/userXyz
  broker <-- rh: ok

  rh -> rh: handle request

  rh -> broker: PUBLISH responses/userXyz\ncorrelationData=123xyz\nrequest
response
  rh <-- broker: ok

  broker -> client: PUBLISH responses/userXyz\ncorrelationData=123xyz\nrequest
response
  broker <-- client: ok
@enduml
```

Listing 83 — MQTT 5 request-response pattern

Using the basic request-response pattern, and the User Properties of MQTT 5.0, the abstract STA API defined in Clause 9 can be implemented over the MQTT protocol.

- Requests are sent to the topic `v2.0/$request`.
- The URL, including query parameters, for the request is passed in the User Attribute named `url`. The URL does not need to be URL-Encoded.
- The request type (Clause 9.7) is passed in the User Property named `type`.
- Request and response parameters (such as `prefer` and `location`) are passed as User Properties.
- The status code of a response is passed in the User Attribute named `status`.
- The content type of a message is passed in the MQTT 5 property `Content Type` (OASIS mqtt-v5.0, Section 3.3.2.3.9)
- Content is sent as the payload of the message.
- The request must contain a response topic that follows the `Response Information` passed by the broker.
- The request may contain correlation data to help the client associate the response to the request.
- All URLs pointing to the STA API start with the version of the STA API and contain no scheme, host or port and no preceding slash.

To minimize the size of the returned payload, by default the JSON format is used with `metadata=none` (Clause 9.9.3.11.2) and clients are advised to use the $select query parameter.

```
Topic: v2.0/$request
ResponseTopic: responses/userxyz
CorrelationData: 42
User Properties:
  url: v2.0
  type: read
```

```
Payload: <empty>
```

Listing 84 — Message requesting the service document

```
Topic: responses/userxyz
CorrelationData: 42
ContentType: application/json;charset=UTF-8
User Properties:
  status: 200
Payload: <the service document>
```

Listing 85 — Response message to the previous read request

```
Topic: v2.0/$request
ResponseTopic: responses/userxyz
CorrelationData: 43
User Properties:
  url: v2.0/Things?$top=1&$orderby=id
  type: read
Payload: <empty>
```

Listing 86 — Message requesting the first thing, ordered by id

```
Topic: responses/userxyz
CorrelationData: 43
ContentType: application/json;charset=UTF-8
User Properties:
  status: 200
Payload:
{
  "value": [
    {
      "id": 1,
      "name": "Oven",
      "description": "This thing is an oven.",
      "properties": {
        "owner": "Ulrike Schmidt",
        "color": "Black"
      }
    }
  ],
  "@nextLink": "v2.0/Things?$top=1&$skip=1&$orderby=id"
}
```

Listing 87 — Response message to the previous read request

```
Topic: v2.0/$request
ResponseTopic: responses/userxyz
ContentType: application/json
User Properties:
  url: v2.0/Things
  type: create
Payload:
{
  "id": 1,
  "name": "Oven",
  "description": "This thing is an oven.",
  "properties": {
    "owner": "Ulrike Schmidt",
    "color": "Black"
  }
```

```
}
```

Listing 88 — Message creating a new thing

```
Topic: responses/userxyz
User Properties:
  status: 201
  location: v2.0/Things(1)
Payload: <empty>
```

Listing 89 — Response message to the previous create request

REQUIREMENT 44: REQUEST / RESPONSE	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/request_response`
INCLUDED IN	Requirements class 7: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/mqtt`
PREREQUISITE	Requirements class 2: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/abstract`
STATEMENT	If a service advertises this requirement in the service document then the service SHALL implement the abstract API as described in Clause 11.3.

11.4. Publish / Subscribe

11.4.1. General

MQTT subscriptions are allowed on EntitySets (Clause 9.6.4, Clause 9.6.9), individual entities (Clause 9.6.5, Clause 9.6.8), and individual entity attributes (Clause 9.6.6, Clause 9.6.7).

To get all updates for an EntitySet or Entity, clients can subscribe to the topic that matches the plain URL pattern, without service root, and without a preceding slash (/).

When an entity or attribute is created, updated/replaced or deleted, that is relevant for the subscription, the service returns a complete JSON representation of the affected entity in the payload of the message.

The action that caused the notification is specified in the User Property named `type`, with a value of `create`, `update` or `delete`. Replace actions are published with the type `update`.

```
v2.0/Datastreams(4)/Observations

User Properties:
  type: create
Payload:
{
```

```
  "@id": "v2.0/Observations(123)",
  "id": 123,
  "result": 45,
  "phenonmenonTime": "2015-02-05T17:00:00Z"
}
```

Listing 90 — Topic to use for receiving all updates for the Observations of a specific Datastream, and a typical message notifying about the creation of a new Observation

Since OData URLs are recursive, MQTT wildcards (# and +) are not allowed in subscriptions in the STA topic tree.

Like HTTP URLs, MQTT topics in the OGC SensorThings API can be extended with query options. Query options are separated from the topic with a question mark character (?) and separated using an ampersand (&). Query options are listed below.

REQUIREMENT 45: MQTT PUBLISH / SUBSCRIBE	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub`
INCLUDED IN	Requirements class 7: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/mqtt`
NORMATIVE STATEMENTS	Requirement 46: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/select` Requirement 47: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/expand` Requirement 48: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/filter`
STATEMENT	If a service advertises this requirement in the service document then the service SHALL implement the publish-subscribe API as described in Clause 11.4.

11.4.2. $select

The '$select' query option (Clause 9.9.3.3) can be added to topics to reduce the amount of data that is sent.

```
v2.0/Datastreams(4)/Observations?$select=phenomenonTime,result
```

Listing 91 — Topic to use for receiving all updates for the Observations of a specific Datastream, limited to only the phenomenonTime and result attributes

REQUIREMENT 46: MQTT PUBLISH / SUBSCRIBE SELECT	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/select`

REQUIREMENT 46: MQTT PUBLISH / SUBSCRIBE SELECT	
STATEMENT	If a service advertises Requirement 46: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/select` in the service document then the service SHALL implement support for the `$select` option as described in Clause 11.4.2.

11.4.3. $expand

The `$expand` query option (Clause 9.9.3.5) can be added to topics to indicate that certain entities related to the created or updated entity should be sent. For example, this can be useful when dealing with moving sensors, to receive new Observations together with their ProximateFeatureOfInterest.

```
v2.0/Datastreams(4)/Observations?$expand=ProximateFeatureOfInterest
```

Listing 92 — Topic to use for receiving all updates for the Observations of a specific Datastream

REQUIREMENT 47: MQTT PUBLISH / SUBSCRIBE EXPAND	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/expand`
STATEMENT	If a service advertises Requirement 47: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/expand` in the service document then the service SHALL implement support for the `$expand` option as described in Clause 11.4.3.

11.4.4. $filter

The $filter query option (Clause 9.9.3.10) can be used to only receive notification of certain changes.

REQUIREMENT 48: MQTT PUBLISH / SUBSCRIBE FILTER	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/filter`
STATEMENT	If a service advertises Requirement 48: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/filter` in the service document then the service SHALL implement support for the `$filter` option as described in Clause 11.4.4.

11.5. Simple Create

Since simple clients, like sensors, may not be capable of handling the full MQTT request/response pattern, the simple-create pattern allows for a simpler way to create entities. Entities can be created by sending a publish request to the topic of an EntitySet (Clause 9.6.4, Clause 9.6.9), without service root, and without a preceding slash (/), followed by `/create`.

```
Topic: v2.0/Datastreams(4)/Observations/create
Payload:
{
  "result": 42,
  "phenomenonTime": {
    "start": "2017-11-12T13:00:00Z"
  }
}
```

Listing 93 — Publish message used to create an Observation in Datastream 4.

More advanced clients are advised to use the MQTT Request/Response pattern to create entities, since it allows the server to return a success or failure response on entity creation.

REQUIREMENT 49: MQTT SIMPLE CREATE	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/simple_create`
INCLUDED IN	Requirements class 7: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/mqtt`
STATEMENT	If a service advertises Requirement 49: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/simple_create` in the service document then the service SHALL implement support simple entity creation as described in Clause 11.5.

12

OGC API – COMMON INTERFACE BINDINGS

12 OGC API — COMMON INTERFACE BINDINGS

12.1. General

The OGC API — Common Bindings specify how to combine implementations of requirements classes from the OGC SensorThings API Standard with those from other OGC API Standards.

This section does not specify a specific version of the OGC API — Common Standard to use. The exact version that an API implementation uses can be found by inspecting the conformance declaration of the API.

12.2. Landing Page

The OGC API — Common (OGC 19-072) Standard specifies several elements for the landing page of an OGC API implementation instance. None of the elements specified conflict with any elements specified by the OGC SensorThings API Standard, and can thus be added to the Service Document (Clause 9.9.2.2) for OGC API — Common compatibility.

The requirements class from the OGC API — Common Standard that specified the landing page requirements is:

http://www.opengis.net/spec/ogcapi-common-1/1.0/req/landing-page

The OGC API — Common Standard lists how to link to API definitions, and how conformance classes can be declared. An example of the elements to add to the Service Document to make it a valid OGC API — Common landing page can be found below.

```
{
  "title": "Service Title",
  "description": "This is the description of the service",
  "attribution": "Suggested attribution text, can include HTML-links.",

  "Links": [
    {
      "rel": "http://www.opengis.net/def/rel/ogc/1.0/conformance",
      "type": "application/json",
      "title": "Conformance Declaration",
      "href": "<Service Root>/v2.0/conformance"
    },
    {
      "rel": "service-desc",
      "type": "application/vnd.oai.openapi+json;version=3.0",
      "title": "OpenAPI 3.0 definition for this endpoint.",
```

```
      "href": "<Service Root>/v2.0/api"
    },
    {
      "rel": "service-desc",
      "type": "application/json",
      "title": "OData CSDL for this endpoint.",
      "href": "<Service Root>/v2.0/$metadata"
    }
  ]
}
```

Listing 94 — Items added to the service document to make it a valid OGC API — Common landing page. The attributes title, description and attribution are recommended but not required.

12.3. OpenAPI Definition

The OpenAPI Specification is a popular way to describe HTTP-Based APIs. An Open API document lists all the url paths, and the inputs and outputs of requests made to those paths. The STA API, being based on OData, cannot be fully described by OpenAPI, but the most common API features can be captured. The OpenAPI definition can be generated from the service metadata document (Clause 9.6.3)

How the API definition for an implementation should be made available is specified in section 9.2 of The OGC API — Common (OGC 19-072) Standard.

13

MEDIA TYPES FOR ANY DATA ENCODING(S)

13 MEDIA TYPES FOR ANY DATA ENCODING(S)

The SensorThings API references the following media types:

- application/json
- application/json-patch+json
- application/geo+json
- application/geopose+json
- text/plain

This OGC Standard uses Link Header as defined in IETF RFC 5988 and
W3C REC-websub-20180123. In particular, the following HTTP Link headers are used:

- `rel="hub"` as defined in https://www.w3.org/TR/websub/#x4-discovery
- `rel="self"` as defined in https://www.w3.org/TR/websub/#x4-discovery
- `rel="help"` as defined in https://www.iana.org/assignments/link-relations/link-relations.xhtml

A

ANNEX A (INFORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE (NORMATIVE)

A ANNEX A (INFORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE (NORMATIVE)

A.1. Conformance Class: HTTP

CONFORMANCE CLASS A.1	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf-class/binding/http`
REQUIREMENTS CLASS	Requirements class 5: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http`
PREREQUISITE	Conformance class A.4: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/datamodel/core`
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.1: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/advertisement` Abstract test A.2: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/request_response`

ABSTRACT TEST A.1	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/advertisement`
REQUIREMENT	Requirement 33: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/advertisement`
TEST PURPOSE	Validate the binding advertisement in the service document.
TEST METHOD	For each request_response binding: 1. Construct a path for the service document. 2. Issue a Read request on that path.

ABSTRACT TEST A.1	
	3. Validate that the http binding is advertised, and endpoints are set.

ABSTRACT TEST A.2	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/request_response
REQUIREMENT	Requirement 34: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/request_response
PREREQUISITE	Conformance class A.5: http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/abstract
TEST PURPOSE	Validate that the request types defined in the abstract API are mapped to the correct HTTP methods.
TEST METHOD	1. Execute a GET request on the URL for the service document. 2. Validate that a service document is returned.

A.2. Conformance Class: WebSub

CONFORMANCE CLASS A.2	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/conf-class/binding/http/websub
REQUIREMENTS CLASS	Requirements class 6: http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/http/websub
PREREQUISITE	Conformance class A.1: http://www.opengis.net/spec/sensorthings/2.0/conf-class/binding/http
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.3: http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/link-hub Abstract test A.5: http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/link-help Abstract test A.4: http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/link-self Abstract test A.6: http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/odata-support Abstract test A.7: http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/odata-discovery

CONFORMANCE CLASS A.2

Abstract test A.8: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/odata-blacklisting`
Abstract test A.9: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/topics-discovery`
Abstract test A.10: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/topics-blacklisting`

ABSTRACT TEST A.3

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/link-hub`
REQUIREMENT	Requirement 35: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/link-hub`
TEST PURPOSE	Validate that the STA-WebSub discovery is supported via HTTP GET.
TEST METHOD	1. Construct a topic URL to be used for subscription. 2. Issue a HTTP HEAD and GET request on that topic URL. 3. Validate that the request method was accepted and that the response headers include the Link headers `hub`.
DESCRIPTION	**NOTE:** The Link header `hub` must always be returned even for a topic URL that is not allowed for subscription. But for this test, it is sufficient to test that the HTTP method is supported and for that the `hub` header is returned.

ABSTRACT TEST A.4

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/link-self`
REQUIREMENT	Requirement 36: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/link-self`
TEST PURPOSE	Validate that the WebSub discovery returns the link headers `hub` and `self` for a topic URL that is **allowed** for subscription.
TEST METHOD	1. Construct a topic URL that is **allowed** for subscription. A topic URL does not include a query string. 2. Issue a HTTP HEAD and GET request on that topic URL. 3. Validate that the response includes the Link header `hub` and `self`. 4. validate that the Link header `help` is not present!

ABSTRACT TEST A.4

DESCRIPTION	**NOTE:** To determine which topics **are allowed**, examine the root page and check the `topics_denied` array.

ABSTRACT TEST A.5

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/link-help`
REQUIREMENT	Requirement 37: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/link-help`
TEST PURPOSE	Validate that the WebSub discovery returns the link headers `hub` and `help` for a topic URL **not allowed** for subscription.
TEST METHOD	1. Construct a topic URL that is **not allowed** for subscription. 2. Issue a HTTP HEAD and GET request on that topic URL. 3. Validate that the response includes the Link header `hub` and `help`. In particular validate that the Link header `self` is not present!
DESCRIPTION	**NOTE:** To determine which topic URLs **are not allowed**, examine the root page. The cause for 'not allowed' SHALL be based on `topics_denied` or `odata_denied`.

```
{
    "serverSettings": {
        "conformance": {
            "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/
websub"
        },
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": ["v1.1/Observations"],
            "odata_denied": ["$expand"]
        }
    }
}
```

Listing A.1 — Example for topics and ODATA restrictions

```
GET /sta/v1.1/Observations HTTP/1.1
Host: sta-websub.secd.eu

HTTP/1.1 200 Ok
Content-type: application/json
Link: <http://hub.sta-websub.secd.eu>; rel="hub"
Link: <http://sta-websub.secd.eu/sta/v1.1/help#topic_denied>; rel="help"
```

Listing A.2 — Example HTTP request and response for a URL including the denied topic `v1.1/Observations`

```
GET /sta/v1.1/Things?$expand=Locations HTTP/1.1
Host: sta-websub.secd.eu

HTTP/1.1 200 Ok
Content-type: application/json
Link: <http://hub.sta-websub.secd.eu>; rel="hub"
Link: <http://sta-websub.secd.eu/sta/v1.1/help#odata_option_denied>; rel="help"
```

Listing A.3 — Example HTTP request and response for a URL including the denied ODATA option $expand

ABSTRACT TEST A.6

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/odata-support`
REQUIREMENT	Requirement 38: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/odata-support`
TEST PURPOSE	Validate that the SensorThings implementation supports the use of ODATA options.
TEST METHOD	1. Construct SensorThings URLs for each valid and implemented ODATA option(s). 2. Issue a HTTP HEAD and GET request on each URL. 3. Validate the returned content to not contain errors.

ABSTRACT TEST A.7

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/odata-discovery`
REQUIREMENT	Requirement 39: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/odata-discovery`
TEST PURPOSE	Validate that the disallowed ODATA options are advertised on the root page.
TEST METHOD	1. Construct a URL to the root page. 2. Issue a HTTP HEAD and GET request on that URL. 3. Validate the contents of the returned document to include the implemented Conformance Class(es) in the `conformance` section of the root page: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub` 4. Validate the contents of the returned document to include the JSON object `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub` in the `serverSettings` including the key `odata_denied`. 5. Validate that the value of the key `odata_denied` is a JSON array that is either empty (“[]”) or contains a list of denied ODATA options.

```
{
    "serverSettings": {
        "conformance": {
```

```
            "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/
websub"
        },
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": [],
            "odata_denied": []
        }
    }
}
```

Listing A.4 — Example root page snippet advertising no denied ODATA options

```
{
    "serverSettings": {
        "conformance": {
            "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/
websub"
        },
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": [],
            "odata_denied": ["$expand"]
        }
    }
}
```

Listing A.5 — Example root page snippet advertising $expand as a denied ODATA option

ABSTRACT TEST A.8

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/odata-blacklisting`
REQUIREMENT	Requirement 40: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/odata-blacklisting`
TEST PURPOSE	Validate that the disallowed ODATA options do not return the HTTP link header `rel="self"`.
TEST METHOD	1. Construct multiple SensorThings URLs each including a supported ODATA option that is listed under the `odata_denied`. 2. Issue HTTP HEAD and GET requests on each URL. 3. Validate the contents of the returned HTTP header to not include the link header `rel="self"` but the link header `rel="help"`.

```
{
    "serverSettings": {
        "conformance": {
            "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/
websub"
        },
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": [],
            "odata_denied": ["$expand"]
        }
    }
```

```
}
```

Listing A.6 — Example root page with denied ODATA option $expand

```
GET /sta/v1.1/Things?$expand=Locations HTTP/1.1
Host: sta-websub.secd.eu

HTTP/1.1 200 Ok
Content-type: application/json
Link: <http://hub.sta-websub.secd.eu>; rel="hub"
Link: <http://sta-websub.secd.eu/sta/v1.1/help#odata_option_denied>; rel="help"
```

Listing A.7 — Example HTTP response for a URL including the denied ODATA option $expand

ABSTRACT TEST A.9	
IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/topics-discovery
REQUIREMENT	Requirement 41: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/topics-discovery
TEST PURPOSE	Validate that the implemented STA-WebSub Conformance Class Discovery is listed on the root page.
TEST METHOD	1. Construct a URL to the root page. 2. Issue a HTTP HEAD and GET request on that URL. 3. Validate the contents of the returned document to include the implemented Conformance Class(es) in the conformance section of the root page: http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub 4. Validate the contents of the returned document to include the JSON object http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub in the serverSettings including the key topics_denied. 5. Validate that the value of the key topics_denied is a JSON array that is either empty ("[]") or contains a list of denied topics.

```
{
    "serverSettings": {
        "conformance": {
            "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/
websub"
        },
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": [],
            "odata_denied": []
        }
    }
}
```

Listing A.8 — Example root page snippet advertising no denied ODATA options

```
{
    "serverSettings": {
        "conformance": {
            "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/
websub"
```

```
        },
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": ["v1.1/Observations"],
            "odata_denied": []
        }
    }
}
```

Listing A.9 — Example root page snippet advertising `v1.1/Observations` as a denied topic

ABSTRACT TEST A.10	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/http/websub/topics-blacklisting`
REQUIREMENT	Requirement 42: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub/topics-blacklisting`
TEST PURPOSE	Validate that the denied topics as advertised on the root page are actually denied for discovery.
TEST METHOD	1. Construct a URL using a denied topic. 2. Issue a HTTP HEAD and GET request on that URL. 3. Validate the returned link headers to contain the `link rel=help"` and **not** `link rel="self"`.

```
{
    "serverSettings": {
        "conformance": {
            "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/
websub"
        },
        "http://www.opengis.net/spec/sensorthings/2.0/req/binding/http/websub":
 {
            "topics_denied": ["v1.1/Datastreams(4711)"],
            "odata_denied": []
        }
    }
}
```

Listing A.10 — Example root page snippet advertising `v1.1/Datastreams(4711)` as a denied topic

```
GET /sta/v1.1/Datastreams(4711) HTTP/1.1
Host: sta-websub.secd.eu

HTTP/1.1 200 Ok
Content-type: application/json
Link: <http://hub.sta-websub.secd.eu>; rel="hub"
Link: <http://sta-websub.secd.eu/sta/v1.1/help#topic_denied>; rel="help"
```

Listing A.11 — Example HTTP response for a URL including the denied topic `v1.1/Datastreams(4711)`

A.3. Conformance Class: MQTT

CONFORMANCE CLASS A.3	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf-class/binding/mqtt`
REQUIREMENTS CLASS	Requirements class 7: `http://www.opengis.net/spec/sensorthings/2.0/req-class/binding/mqtt`
PREREQUISITE	Conformance class A.4: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/datamodel/core`
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.11: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/advertisement` Abstract test A.12: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/request_response` Abstract test A.13: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/pub_sub` Abstract test A.14: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/pub_sub/select` Abstract test A.15: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/pub_sub/expand` Abstract test A.16: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/pub_sub/filter` Abstract test A.17: `http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/simple_create`

ABSTRACT TEST A.11	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/advertisement`
REQUIREMENT	Requirement 43: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/advertisement`
TEST PURPOSE	Validate the MQTT advertisement in the service documents of all request-response bindings.
TEST METHOD	For each request-response binding: 1. Construct a path for the service document. 2. Issue a Read request on that path. 3. Validate that the MQTT binding is advertised, and endpoints are set.

ABSTRACT TEST A.12

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/request_response`
REQUIREMENT	Requirement 44: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/request_response`
PREREQUISITE	Conformance class A.5: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/abstract`
TEST PURPOSE	Validate that the MQTT interface implements the request-response API.
TEST METHOD	1. execute an MQTT read request for the path for the service document. 2. Validate that a service document is returned. 3. Validate that the service document contains the request-response requirement.

ABSTRACT TEST A.13

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/pub_sub`
REQUIREMENT	Requirement 45: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub`
TEST PURPOSE	Validate that the MQTT interface implements the publish-subscribe bindings described in Clause 11.4.
TEST METHOD	1. For each path that addresses an entity set a) Subscribe to the path with MQTT Subscribe, b) create a new entity in the subscribed entity set, c) Check if a correct JSON representation of the newly created entity through MQTT is received. 2. For each path that addresses an entity set a) Subscribe to the path with MQTT Subscribe, b) update an existing entity in the subscribed entity set, c) Check if a complete JSON representation of the updated entity through MQTT is received. 3. For each path that addresses an entity a) Subscribe to the path with MQTT Subscribe, b) update the entity with PATCH, c) Check if the JSON object of the updated entity is received. 4. For a selection of paths that addresses an entity property a) Subscribe to the path with MQTT Subscribe, b) update the property with PATCH, c) Check if the JSON object of the updated property is received.

ABSTRACT TEST A.14

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/pub_sub/select`
REQUIREMENT	Requirement 46: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/select`
TEST PURPOSE	Validate that the MQTT interface implements the $select parameter.
TEST METHOD	1. For each path that addresses an entity set a) Subscribe to the path with MQTT Subscribe, selecting a subset of the properties of the entity type using `$select`, b) create a new entity in the subscribed entity set, c) Check if the JSON representation of the newly created entity, received through MQTT, only contains the selected properties. 2. For each path that addresses an entity set a) Subscribe to the path with MQTT Subscribe, selecting a subset of the properties of the entity type using `$select`, b) update an existing entity in the subscribed entity set, c) Check if the JSON representation of the updated entity, received through MQTT, only contains the selected properties. 3. For each path that addresses an entity a) Subscribe to the path with MQTT Subscribe, selecting a subset of the properties of the entity type using `$select`, b) update the entity with PATCH, c) Check if the JSON representation of the updated entity, received through MQTT, only contains the selected properties.

ABSTRACT TEST A.15

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/pub_sub/expand`
REQUIREMENT	Requirement 47: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/expand`
TEST PURPOSE	Validate that the MQTT interface implements the $expand parameter.
TEST METHOD	1. For a variety of paths that address an entity or entity set a) Subscribe to the path with MQTT Subscribe, expanding one or more navigation properties using `$expand`, b) create a new entity in the subscribed entity set or update the subscribed entity. c) Check if the JSON representation of the entity, received through MQTT, contains the expanded relation.

ABSTRACT TEST A.16	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/pub_sub/filter`
REQUIREMENT	Requirement 48: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/pub_sub/filter`
TEST PURPOSE	Validate that the MQTT interface implements the $filter parameter.
TEST METHOD	1. For each path that addresses an entity set a) Subscribe to the path with MQTT Subscribe, using a filter i) create a new entity, that matches the filter, in the subscribed entity set, A) Check if a correct JSON representation of the newly created entity through MQTT is received. ii) create a new entity, that does not match the filter, in the subscribed entity set, A) Check if no JSON representation of the newly created entity is received.

ABSTRACT TEST A.17	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/binding/mqtt/simple_create`
REQUIREMENT	Requirement 49: `http://www.opengis.net/spec/sensorthings/2.0/req/binding/mqtt/simple_create`
TEST PURPOSE	Validate that the MQTT interface implements MQTT Simple Create.
TEST METHOD	1. For each path that addresses an entity set a) Send an MQTT Publish message, containing a valid entity representation for the entity set, to the topic of the entity set with `/create` appended. b) Check if the new entity is created.

A.4. Conformance Class: Data Model

CONFORMANCE CLASS A.4	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf-class/datamodel/core`
REQUIREMENTS CLASS	Requirements class 1: `http://www.opengis.net/spec/sensorthings/2.0/req-class/datamodel/core`

CONFORMANCE CLASS A.4

CONFORMANCE TESTS	Abstract test A.18: http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/thing Abstract test A.19: http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/location Abstract test A.20: http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/historical-location Abstract test A.21: http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/sensor Abstract test A.22: http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/datastream Abstract test A.23: http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/observed-property Abstract test A.24: http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/feature Abstract test A.25: http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/feature-type Abstract test A.26: http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/observation

ABSTRACT TEST A.18

IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/thing
REQUIREMENT	Requirement 1: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/thing
TEST PURPOSE	Validate that the API Definition document can be retrieved from the expected location.
TEST METHOD	For each request_response binding: 1. Request a list of Things 2. Check if each Thing has all the mandatory Entity Attributes 3. Check if each Thing has all the mandatory Navigation Attributes

ABSTRACT TEST A.19

IDENTIFIER	http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/location
REQUIREMENT	Requirement 2: http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/location
TEST PURPOSE	Validate that Entities of this EntityType have the required attributes and relations.
TEST METHOD	For each request_response binding:

ABSTRACT TEST A.19

1. Request a list of Locations
2. Check if each Location has all the mandatory Entity Attributes
3. Check if each Location has all the mandatory Navigation Attributes

ABSTRACT TEST A.20

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/historical-location`
REQUIREMENT	Requirement 3: `http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/historical-location`
TEST PURPOSE	Validate that Entities of this EntityType have the required attributes and relations.
TEST METHOD	For each request_response binding: 1. Request a list of HistoricalLocations 2. Check if each HistoricalLocation has all the mandatory Entity Attributes 3. Check if each HistoricalLocation has all the mandatory Navigation Attributes

ABSTRACT TEST A.21

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/sensor`
REQUIREMENT	Requirement 5: `http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/sensor`
TEST PURPOSE	Validate that Entities of this EntityType have the required attributes and relations.
TEST METHOD	For each request_response binding: 1. Request a list of Sensors 2. Check if each Sensor has all the mandatory Entity Attributes 3. Check if each Sensor has all the mandatory Navigation Attributes

ABSTRACT TEST A.22

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/datastream`
REQUIREMENT	Requirement 4: `http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/datastream`
TEST PURPOSE	Validate that Entities of this EntityType have the required attributes and relations.

ABSTRACT TEST A.22

TEST METHOD	For each request_response binding: 1. Request a list of Datastreams 2. Check if each Datastream has all the mandatory Entity Attributes 3. Check if each Datastream has all the mandatory Navigation Attributes 4. Check if the linked ObservedProperties match those referenced in the resultType attribute.

ABSTRACT TEST A.23

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/observed-property`
REQUIREMENT	Requirement 6: `http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/observed-property`
TEST PURPOSE	Validate that Entities of this EntityType have the required attributes and relations.
TEST METHOD	For each request_response binding: 1. Request a list of ObservedProperties 2. Check if each ObservedProperty has all the mandatory Entity Attributes 3. Check if each ObservedProperty has all the mandatory Navigation Attributes

ABSTRACT TEST A.24

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/feature`
REQUIREMENT	Requirement 8: `http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/feature`
TEST PURPOSE	Validate that Entities of this EntityType have the required attributes and relations.
TEST METHOD	For each request_response binding: 1. Request a list of Features 2. Check if each Feature has all the mandatory Entity Attributes 3. Check if each Feature has all the mandatory Navigation Attributes

ABSTRACT TEST A.25

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/feature-type`

ABSTRACT TEST A.25	
REQUIREMENT	Requirement 9: `http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/feature-type`
TEST PURPOSE	Validate that Entities of this EntityType have the required attributes and relations.
TEST METHOD	For each request_response binding: 1. Request a list of FeatureTypes 2. Check if each FeatureType has all the mandatory Entity Attributes 3. Check if each FeatureType has all the mandatory Navigation Attributes

ABSTRACT TEST A.26	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/datamodel/core/observation`
REQUIREMENT	Requirement 7: `http://www.opengis.net/spec/sensorthings/2.0/req/datamodel/core/observation`
TEST PURPOSE	Validate that Entities of this EntityType have the required attributes and relations.
TEST METHOD	For each request_response binding: 1. Request a list of Observations 2. Check if each Observation has all the mandatory Entity Attributes 3. Check if each Observation has all the mandatory Navigation Attributes 4. Check if the content of the result attribute matches the structure defined in the resultType attribute of the linked Datastream

A.5. Conformance Class: Abstract API

CONFORMANCE CLASS A.5	
IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/abstract`
REQUIREMENTS CLASS	Requirements class 2: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/abstract`
CONFORMANCE TESTS	Conformance class A.6: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/read`

CONFORMANCE CLASS A.5

	Conformance class A.7: `http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/cud`

CONFORMANCE CLASS A.6

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/read`
REQUIREMENTS CLASS	Requirements class 3: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/read`
CONFORMANCE TESTS	Abstract test A.27: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/paths` Abstract test A.28: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/precedence` Abstract test A.29: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/select` Abstract test A.30: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/select_distinct` Abstract test A.31: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/expand` Abstract test A.32: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/top_skip` Abstract test A.33: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/count` Abstract test A.34: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/orderby` Abstract test A.35: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/filter` Abstract test A.36: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/format` Abstract test A.37: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/pagination`

ABSTRACT TEST A.27

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/paths`
REQUIREMENT	Requirement 10: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/paths`
TEST PURPOSE	Validate that all the specified paths can be read, for the implemented data model.
TEST METHOD	1. For each binding that supports request-response a) For each URL pattern defined in Clause 9.9.2

ABSTRACT TEST A.27

i) Execute a read request to the URL with no query options.

ii) Test if the server returns the appropriate response.

ABSTRACT TEST A.28

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/precedence`
REQUIREMENT	Requirement 11: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/precedence`
TEST PURPOSE	Validate that the implementation uses the correct operator precedence.
TEST METHOD	1. For each binding that supports request-response a) Send a query with the $filter query option for each combination of two built-in filter operators with adjacent operator precedence. b) Check if the server returns appropriate result as defined in this specification.

ABSTRACT TEST A.29

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/select`
REQUIREMENT	Requirement 12: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/select`
TEST PURPOSE	Validate that the implementation correctly implements the $select query option.
TEST METHOD	1. For each binding that supports request-response a) For each URL pattern defined in Clause 9.9.2 i) Execute read requests with a variety of $select query options to the URL. ii) Test if the server returns the appropriate response.

ABSTRACT TEST A.30

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/select_distinct`
REQUIREMENT	Requirement 13: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/select_distinct`
TEST PURPOSE	Validate that, if the query advertises select-distinct capability, it is implemented correctly.
TEST METHOD	1. For each binding that supports request-response and advertises select-distinct capability a) For each URL pattern defined in Clause 9.9.2 that returns a list of entities

ABSTRACT TEST A.30

i) Execute read requests with a an appropriate $select=distinct: query option to the URL.

ii) Test if the server returns the appropriate response.

ABSTRACT TEST A.31

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/expand`
REQUIREMENT	Requirement 14: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/expand`
TEST PURPOSE	Validate that the implementation correctly implements the $expand query option.
TEST METHOD	1. For each binding that supports request-response a) For various URL patterns defined in Clause 9.9.2 i) Execute read requests with a variety of $expand query options to the URL. A) Including nested expands and expands with query options. ii) Test if the server returns the appropriate response.

ABSTRACT TEST A.32

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/top_skip`
REQUIREMENTS	Requirement 15: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/top` Requirement 16: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/skip`
TEST PURPOSE	Validate that the implementation correctly implements the Top and $skip query options.
TEST METHOD	1. For each binding that supports request-response a) For various URL patterns defined in Clause 9.9.2 that return a list of entities i) Execute read requests with $top=10 and $skip=0. ii) Execute read requests with $top=2 and $skip=1. iii) Test if the server returns items two and three in the second request, when compared to the first request.

ABSTRACT TEST A.33

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/count`

ABSTRACT TEST A.33

REQUIREMENT	Requirement 17: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/count`
TEST PURPOSE	Validate that the implementation correctly implements the $count query option.
TEST METHOD	1. For each binding that supports request-response a) For various URL patterns defined in Clause 9.9.2 that return a list of entities i) Execute read requests with $count=false, ii) Test if the server correctly omits a count in the response. iii) Execute read requests with $count=true, iv) Test if the response correctly contains a count.

ABSTRACT TEST A.34

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/orderby`
REQUIREMENT	Requirement 18: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/orderby`
TEST PURPOSE	Validate that the implementation correctly implements the $orderby query option.
TEST METHOD	1. For each binding that supports request-response a) For various URL patterns defined in Clause 9.9.2 that return a list of entities i) Execute read requests with a $orderby containing a single attribute. A) Validate that the returned entities are correctly ordered, in ascending order, by the given attribute. ii) Execute read requests with a $orderby containing a single attribute with the `desc` modifier. A) Validate that the returned entities are correctly ordered, in descending order, by the given attribute. iii) Execute read requests with a $orderby containing a single attribute with the `asc` modifier. A) Validate that the returned entities are correctly ordered, in ascending order, by the given attribute. iv) Execute read requests with a $orderby containing two attributes. A) Validate that the returned entities are correctly ordered, in ascending order, by the given attributes.

ABSTRACT TEST A.35

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/filter`

ABSTRACT TEST A.35

REQUIREMENT	Requirement 19: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/filter`
TEST PURPOSE	Validate that the implementation correctly implements the $filter query option.
TEST METHOD	1. For each binding that supports request-response a) For various URL patterns defined in Clause 9.9.2 that return a list of entities i) Execute various read requests containing filter functions ii) Validate that only entities matching the filter are returned.

ABSTRACT TEST A.36

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/format`
REQUIREMENT	Requirement 20: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/format`
TEST PURPOSE	Validate that the implementation correctly implements the $format query option.
TEST METHOD	1. For each binding that supports request-response a) For various URL patterns defined in Clause 9.9.2 i) Execute a request with the request parameter `$format=application/json;metadata=full` ii) Validate that all metadata is correctly returned. iii) Execute a request with the request parameter `$format=application/json;metadata=minimal` iv) Validate that only context, count (if requested) and nextLink (if applicable) are returned. v) Execute a request with the request parameter `$format=application/json;metadata=none` vi) Validate that only nextLink (if applicable) and count (if requested) are returned.

ABSTRACT TEST A.37

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/read/options/pagination`
REQUIREMENT	Requirement 21: `http://www.opengis.net/spec/sensorthings/2.0/req/api/read/options/pagination`
TEST PURPOSE	Validate that the implementation correctly implements server-driven pagination.
TEST METHOD	1. For each binding that supports request-response a) For various URL patterns defined in Clause 9.9.2 that return a list of entities

ABSTRACT TEST A.37

i) Execute read requests with $top=10 and $skip=0.

ii) Execute read requests with $top=2 and $skip=0.

iii) Validate that a nextLink is present

iv) Execute a read request on the nextLink

v) Validate that the server returns items three and four in this last request, when compared to the first request.

CONFORMANCE CLASS A.7

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf-class/api/cud`
REQUIREMENTS CLASS	Requirements class 4: `http://www.opengis.net/spec/sensorthings/2.0/req-class/api/cud`
CONFORMANCE TESTS	Abstract test A.38: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/advertisement` Abstract test A.39: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/create` Abstract test A.40: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/deep_insert` Abstract test A.41: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/update` Abstract test A.42: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/replace` Abstract test A.43: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/deep_update` Abstract test A.44: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/json_patch` Abstract test A.45: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/update_relations` Abstract test A.46: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/delete` Abstract test A.47: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/filtered_delete` Conformance test A.7-8: `http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/return_value`

ABSTRACT TEST A.38

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/advertisement`

ABSTRACT TEST A.38

REQUIREMENT	Requirement 22: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/advertisement`
TEST PURPOSE	Validate that CUD capability is advertised
TEST METHOD	For each request_response binding: 1. Construct a path for the service document. 2. Issue a Read request on that path. 3. Validate that the CUD capability is advertised.

ABSTRACT TEST A.39

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/create`
REQUIREMENTS	Requirement 23: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/create` Requirement 29: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/return_value`
TEST PURPOSE	Validate that the create capability is correctly implemented.
TEST METHOD	1. For each SensorThings entity type creates an entity instance by linking to existing entities with a single request. a) Varying the value of the return option for the prefer request parameter. i) Check if the server responds as defined in this specification. ii) Check if the entity is correctly created, with the correct relations. 2. For each SensorThings entity type creates an entity instance that does not follow the integrity constraints of the data model (Clause 8) by linking to existing entities with a single request. a) Check if the server responds as defined in this specification. b) Check that the entity is not created. 3. Create a Location for a Thing entity a) Check if the Thing has a HistoricalLocation created by the service according to the Location entity. 4. Create a HistoricalLocation for a Thing entity, with a time that is later than the existing HistoricalLocations of the Thing. a) Check if the Location of the Thing was changed to the Location of the Historical Location.

ABSTRACT TEST A.40

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/deep_insert`

ABSTRACT TEST A.40

REQUIREMENTS	Requirement 24: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/deep_insert` Requirement 29: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/return_value`
TEST PURPOSE	Validate that the deep insert capability is correctly implemented.
TEST METHOD	1. For each SensorThings entity type creates an entity instance by following the integrity constraints of the data model (Clause 8) and creating the related entities with a single request (i.e., deep insert), a) Varying the value of the return option for the prefer request parameter, i) Check if the entity instance is successfully created. ii) Check if the service responds as defined in this specification. 2. 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), a) Check if the service fails the request without creating any entity within the deep insert request. b) Check if the service responds the appropriate status code. 3. For each SensorThings entity type issue an entity creation request that does not follow the integrity constraints of the data model (Clause 8) with deep insert, a) Check if the service fails the request without creating any entity within the deep insert request. b) Check if the service responds with the appropriate status code.

ABSTRACT TEST A.41

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/update`
REQUIREMENTS	Requirement 25: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/update` Requirement 29: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/return_value`
TEST PURPOSE	Validate that the update capability is correctly implemented.
TEST METHOD	1. For each SensorThings entity type send an update request, a) with updated values for varying attributes, b) varying the value of the return option for the prefer request parameter, i) check if the properties provided in the payload corresponding to updatable properties replace the value of the corresponding property in the entity. ii) check if the missing properties of the containing entity or complex property are not directly altered. iii) Check if the service responds as defined in this specification.

ABSTRACT TEST A.42

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/replace`
REQUIREMENTS	Requirement 26: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/replace` Requirement 29: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/return_value`
TEST PURPOSE	Validate that the replace capability, if advertised, is correctly implemented.
TEST METHOD	1. For each SensorThings entity type send a replace request, a) with updated values for varying attributes, i) check if the properties provided in the payload corresponding to updatable properties replace the value of the corresponding property in the entity. ii) check if the missing properties of the containing entity or complex property are removed. iii) Check if the service responds as defined in this specification.

ABSTRACT TEST A.43

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/deep_update`
REQUIREMENTS	Requirement 27: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/deep_update` Requirement 29: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/return_value`
TEST PURPOSE	Validate that the deep update capability, if advertised, is correctly implemented.
TEST METHOD	1. For each SensorThings entity type send an update request that contains related entities as inline content, a) check if the service updates the inline entities as specified.

ABSTRACT TEST A.44

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/json_patch`
REQUIREMENTS	Requirement 28: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/json_patch` Requirement 29: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/return_value`
TEST PURPOSE	Validate that the JSON-Patch capability, if advertised, is correctly implemented.
TEST METHOD	1. For each SensorThings entity type send a JSON-PATCH request, a) With various JSON-Patch operations,

ABSTRACT TEST A.44

i) check if the service updates the inline entities as specified.

ABSTRACT TEST A.45

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/update_relations`
REQUIREMENT	Requirement 30: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/update_relations`
TEST PURPOSE	Validate that relations can be updated as specified.
TEST METHOD	1. For each relation in the SensorThings data model, a) Send all relation-update requests appropriate for the relation, i) Check if the service correctly updates the relation, if the update does not violate the integrity requirements of the data model. ii) Check if the service correctly rejects the update, if the update violates the integrity requirements of the data model.

ABSTRACT TEST A.46

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/delete`
REQUIREMENT	Requirement 31: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/delete`
TEST PURPOSE	Validate that the deleting of entities is correctly implemented.
TEST METHOD	1. For each entity type in the SensorThings data model, a) Send a delete requests deleting an entity of the given type i) Check if the service correctly removes the entity. ii) Check if the service correctly removes any entities that depended on the deleted entity.

ABSTRACT TEST A.47

IDENTIFIER	`http://www.opengis.net/spec/sensorthings/2.0/conf/api/cud/filtered_delete`
REQUIREMENT	Requirement 32: `http://www.opengis.net/spec/sensorthings/2.0/req/api/cud/filtered_delete`
TEST PURPOSE	Validate that the filtered-delete capability, if advertised, is correctly implemented.
TEST METHOD	1. For each entity type in the SensorThings data model,

ABSTRACT TEST A.47

a) Send a filtered-delete requests deleting a specific sub-set of the entities of the given type,

   i) Check if the service correctly removes the entities.

   ii) Check if the service correctly removes any entities that depended on the deleted entities.

   iii) Check if the service does not delete any entities not matching the filter.

B

ANNEX B (INFORMATIVE) SHORT CSDL EXAMPLE WITH COMMENTS

ANNEX B (INFORMATIVE) SHORT CSDL EXAMPLE WITH COMMENTS

```
{
    "$Version": "4.01",  // OData version
    "$EntityContainer": "org.OGC.STA.SensorThingsV2",  // Namespace + . +
container name
    "org.OGC.STA": {         // Namespace
        "Datastream": {  // EntityType name
            "$Kind": "EntityType",  // Object is an EntityType
            "$Key": [    // Primary key of the EntityType
                "id"     // Primary key consists of the id field
            ],
            "id": {                  // id field, structural properties have
no $Kind
                "$Type": "Edm.Int64"  // Type of the id field
            },                        // Not nullable, but PrimaryKeys are auto
 generated
            "name": {},   // name field, Default $Type=Edm.String, not nullable
            "properties": {                // Property properties
                "$Type": "org.OGC.STA.Object",  // custom type defined in org.
OGC.STA namespace
                "$Nullable": true           // Nullable, thus Optional
            },
            "phenomenonTime": {               // Property phenomenonTime
                "$Type": "org.OGC.STA.TM_Period", // custom type defined in
org.OGC.STA namespace
                "$Nullable": true             // Nullable, thus Optional
            },
            "Observations": {                   // Property Observations
                "$Kind": "NavigationProperty",  // NavigationProperty
                "$Collection": true,            // links to a Set of Entities
                "$Partner": "Datastream",       // inverse link is called
Datastream
                "$Type": "org.OGC.STA.Observation", // Target is of type
Observation
                "$Nullable": true               // can be an empty set
            }
        },
        "Observation": {            // EntityType name
            "$Kind": "EntityType",  // Object is an EntityType
            "$Key": [               // Primary key of the EntityType
                "id"                // Primary key consists of the id field
            ],
            "id": {                   // id field, structural properties have
no $Kind
                "$Type": "Edm.Int64"  // Type of the id field
            },                        // Not nullable, but PrimaryKeys are auto
 generated
            "result": {
                "$Type": "Edm.Untyped"  // Untyped=can be anything
            },
            "Datastream": {                    // Property Datastream linking
to a single Entity
                "$Kind": "NavigationProperty", // is a NavigationProperty
```

```
                "$Partner": "Observations",    // inverse relation is called
Observations
                "$Type": "org.OGC.STA.Datastream"  //  links to a Datastream
            }                                  // No $Nullable, thus mandatory!
        },
        "Object": {                 // Custom type Object
            "$Kind": "ComplexType", // is a complex type
            "$OpenType": true       // Open, thus can hold user-defined
properties
        },
        "TM_Period": {                         // Custom type TM_Period
            "$Kind": "ComplexType",            // is a complex type
            "start": {                         // Has a structural property
named start
                "$Type": "Edm.DateTimeOffset"  // that is of type
DataTimeOffset
            },                                 // No $Nullable, thus mandatory!
            "end": {                           // Has a structural property
named end
                "$Type": "Edm.DateTimeOffset"  // that is of type
DataTimeOffset
            }
        },
        "SensorThingsV2": {             // The entity container defining the
base entity sets
            "$Kind": "EntityContainer",
            "Datastreams": {                    // The v2.0/Datastreams
entitySet
                "$Collection": true,            // is an entity set
                "$Type": "org.OGC.STA.Datastream",   // containing Datastreams
                "$NavigationPropertyBinding": {  // The navigationProperties
link to other top-level sets
                    "Observations": "Observations"  // v2.0/Datastreams(x)/
Observations are also in v2.0/Observations
                }
            },
            "Observations": {                   // The v2.0/Observations
entitySet
                "$Collection": true,            // is an entity set
                "$Type": "org.OGC.STA.Observation",  // containing Observations
                "$NavigationPropertyBinding": {
                    "Datastream": "Datastreams"  // v2.0/Observations(x)/
Datastream is also in v2.0/Datastreams
                }
            }
        }
    }
}
```

Listing B.1 — A shortened example CSDL definition with only (incomplete) Datastreams and Observations.

C

ANNEX C (INFORMATIVE) EXAMPLE SENSORTHINGS API CORE CSDL

ANNEX C
(INFORMATIVE)
EXAMPLE SENSORTHINGS API CORE CSDL


```
{
    "$Version": "4.01",
    "$EntityContainer": "org.OGC.STA.SensorThingsV2",
    "org.OGC.STA": {
        "Datastream": {
            "$Kind": "EntityType",
            "$Key": [
                "id"
            ],
            "id": {
                "$Type": "Edm.Int64"
            },
            "name": {},
            "description": {
                "$Nullable": true
            },
            "resultType": {
                "$Type": "org.OGC.STA.Object"
            },
            "properties": {
                "$Type": "org.OGC.STA.Object",
                "$Nullable": true
            },
            "observedArea": {
                "$Type": "Edm.Geometry",
                "$SRID": 4326,
                "$Nullable": true
            },
            "phenomenonTime": {
                "$Type": "org.OGC.STA.TM_Period",
                "$Nullable": true
            },
            "resultTime": {
                "$Type": "org.OGC.STA.TM_Period",
                "$Nullable": true
            },
            "Observations": {
                "$Kind": "NavigationProperty",
                "$Collection": true,
                "$Partner": "Datastream",
                "$Type": "org.OGC.STA.Observation",
                "$Nullable": true
            },
            "ObservedProperties": {
                "$Kind": "NavigationProperty",
                "$Collection": true,
                "$Partner": "Datastreams",
                "$Type": "org.OGC.STA.ObservedProperty",
                "$Nullable": true
            },
            "Sensor": {
                "$Kind": "NavigationProperty",
```

```
            "$Partner": "Datastreams",
            "$Type": "org.OGC.STA.Sensor"
        },
        "Thing": {
            "$Kind": "NavigationProperty",
            "$Partner": "Datastreams",
            "$Type": "org.OGC.STA.Thing"
        },
        "ultimateFeatureOfInterest": {
            "$Kind": "NavigationProperty",
            "$Partner": "Datastreams",
            "$Type": "org.OGC.STA.Feature",
            "$Nullable": true
        }
    },
    "Feature": {
        "$Kind": "EntityType",
        "$Key": [
            "id"
        ],
        "id": {
            "$Type": "Edm.Int64"
        },
        "name": {},
        "description": {
            "$Nullable": true
        },
        "encodingType": {
            "$Nullable": true
        },
        "feature": {
            "$Type": "org.OGC.STA.Geometry",
            "$Nullable": true
        },
        "properties": {
            "$Type": "org.OGC.STA.Object",
            "$Nullable": true
        },
        "Datastreams": {
            "$Kind": "NavigationProperty",
            "$Collection": true,
            "$Partner": "ultimateFeatureOfInterest",
            "$Type": "org.OGC.STA.Datastream",
            "$Nullable": true
        },
        "FeatureTypes": {
            "$Kind": "NavigationProperty",
            "$Collection": true,
            "$Partner": "Features",
            "$Type": "org.OGC.STA.FeatureType",
            "$Nullable": true
        },
        "Observations": {
            "$Kind": "NavigationProperty",
            "$Collection": true,
            "$Partner": "proximateFeatureOfInterest",
            "$Type": "org.OGC.STA.Observation",
            "$Nullable": true
        }
    },
    "FeatureType": {
        "$Kind": "EntityType",
        "$Key": [
```

```
                "id"
            ],
            "id": {
                "$Type": "Edm.Int64"
            },
            "name": {},
            "description": {
                "$Nullable": true
            },
            "definition": {
                "$Nullable": true
            },
            "properties": {
                "$Type": "org.OGC.STA.Object",
                "$Nullable": true
            },
            "Features": {
                "$Kind": "NavigationProperty",
                "$Collection": true,
                "$Partner": "FeatureTypes",
                "$Type": "org.OGC.STA.Feature",
                "$Nullable": true
            }
        },
        "HistoricalLocation": {
            "$Kind": "EntityType",
            "$Key": [
                "id"
            ],
            "id": {
                "$Type": "Edm.Int64"
            },
            "time": {
                "$Type": "Edm.DateTimeOffset"
            },
            "Locations": {
                "$Kind": "NavigationProperty",
                "$Collection": true,
                "$Partner": "HistoricalLocations",
                "$Type": "org.OGC.STA.Location",
                "$Nullable": true
            },
            "Thing": {
                "$Kind": "NavigationProperty",
                "$Partner": "HistoricalLocations",
                "$Type": "org.OGC.STA.Thing"
            }
        },
        "Location": {
            "$Kind": "EntityType",
            "$Key": [
                "id"
            ],
            "id": {
                "$Type": "Edm.Int64"
            },
            "name": {},
            "description": {
                "$Nullable": true
            },
            "encodingType": {
                "$Nullable": true
            },
```

```
            "location": {
                "$Type": "org.OGC.STA.Geometry",
                "$Nullable": true
            },
            "properties": {
                "$Type": "org.OGC.STA.Object",
                "$Nullable": true
            },
            "HistoricalLocations": {
                "$Kind": "NavigationProperty",
                "$Collection": true,
                "$Partner": "Locations",
                "$Type": "org.OGC.STA.HistoricalLocation",
                "$Nullable": true
            },
            "Things": {
                "$Kind": "NavigationProperty",
                "$Collection": true,
                "$Partner": "Locations",
                "$Type": "org.OGC.STA.Thing",
                "$Nullable": true
            }
        },
        "Observation": {
            "$Kind": "EntityType",
            "$Key": [
                "id"
            ],
            "id": {
                "$Type": "Edm.Int64"
            },
            "phenomenonTime": {
                "$Type": "org.OGC.STA.TM_Object",
                "$Nullable": true
            },
            "resultTime": {
                "$Type": "Edm.DateTimeOffset",
                "$Nullable": true
            },
            "validTime": {
                "$Type": "org.OGC.STA.TM_Period",
                "$Nullable": true
            },
            "properties": {
                "$Type": "org.OGC.STA.Object",
                "$Nullable": true
            },
            "result": {
                "$Type": "Edm.Untyped",
                "$Nullable": true
            },
            "Datastream": {
                "$Kind": "NavigationProperty",
                "$Partner": "Observations",
                "$Type": "org.OGC.STA.Datastream"
            },
            "proximateFeatureOfInterest": {
                "$Kind": "NavigationProperty",
                "$Partner": "Observations",
                "$Type": "org.OGC.STA.Feature",
                "$Nullable": true
            }
        },
```

```
    "ObservedProperty": {
        "$Kind": "EntityType",
        "$Key": [
            "id"
        ],
        "id": {
            "$Type": "Edm.Int64"
        },
        "definition": {},
        "name": {},
        "description": {
            "$Nullable": true
        },
        "properties": {
            "$Type": "org.OGC.STA.Object",
            "$Nullable": true
        },
        "Datastreams": {
            "$Kind": "NavigationProperty",
            "$Collection": true,
            "$Partner": "ObservedProperties",
            "$Type": "org.OGC.STA.Datastream",
            "$Nullable": true
        }
    },
    "Sensor": {
        "$Kind": "EntityType",
        "$Key": [
            "id"
        ],
        "id": {
            "$Type": "Edm.Int64"
        },
        "name": {},
        "description": {
            "$Nullable": true
        },
        "encodingType": {
            "$Nullable": true
        },
        "metadata": {
            "$Nullable": true
        },
        "properties": {
            "$Type": "org.OGC.STA.Object",
            "$Nullable": true
        },
        "Datastreams": {
            "$Kind": "NavigationProperty",
            "$Collection": true,
            "$Partner": "Sensor",
            "$Type": "org.OGC.STA.Datastream",
            "$Nullable": true
        }
    },
    "Thing": {
        "$Kind": "EntityType",
        "$Key": [
            "id"
        ],
        "id": {
            "$Type": "Edm.Int64"
        },
```

```
            "name": {},
            "description": {
                "$Nullable": true
            },
            "properties": {
                "$Type": "org.OGC.STA.Object",
                "$Nullable": true
            },
            "Datastreams": {
                "$Kind": "NavigationProperty",
                "$Collection": true,
                "$Partner": "Thing",
                "$Type": "org.OGC.STA.Datastream",
                "$Nullable": true
            },
            "HistoricalLocations": {
                "$Kind": "NavigationProperty",
                "$Collection": true,
                "$Partner": "Thing",
                "$Type": "org.OGC.STA.HistoricalLocation",
                "$Nullable": true
            },
            "Locations": {
                "$Kind": "NavigationProperty",
                "$Collection": true,
                "$Partner": "Things",
                "$Type": "org.OGC.STA.Location",
                "$Nullable": true
            }
        },
        "Geometry": {
            "$Kind": "TypeDefinition",
            "$UnderlyingType": "Edm.Geometry",
            "$SRID": 4326,
            "@Core.Description": "A Free Location object"
        },
        "Object": {
            "$Kind": "ComplexType",
            "$OpenType": true,
            "@Core.Description": "A free object that can contain anything"
        },
        "TM_Object": {
            "$Kind": "ComplexType",
            "@Core.Description": "An ISO time instant or time interval.",
            "start": {
                "$Type": "Edm.DateTimeOffset"
            },
            "end": {
                "$Type": "Edm.DateTimeOffset",
                "$Nullable": true
            }
        },
        "TM_Period": {
            "$Kind": "ComplexType",
            "@Core.Description": "An ISO time interval.",
            "start": {
                "$Type": "Edm.DateTimeOffset"
            },
            "end": {
                "$Type": "Edm.DateTimeOffset"
            }
        },
        "SensorThingsV2": {
```

```
"$Kind": "EntityContainer",
"Datastreams": {
    "$Collection": true,
    "$Type": "org.OGC.STA.Datastream",
    "$NavigationPropertyBinding": {
        "ultimateFeatureOfInterest": "Features",
        "Observations": "Observations",
        "ObservedProperties": "ObservedProperties",
        "Thing": "Things",
        "Sensor": "Sensors"
    }
},
"Features": {
    "$Collection": true,
    "$Type": "org.OGC.STA.Feature",
    "$NavigationPropertyBinding": {
        "Datastreams": "Datastreams",
        "FeatureTypes": "FeatureTypes",
        "Observations": "Observations"
    }
},
"FeatureTypes": {
    "$Collection": true,
    "$Type": "org.OGC.STA.FeatureType",
    "$NavigationPropertyBinding": {
        "Features": "Features"
    }
},
"HistoricalLocations": {
    "$Collection": true,
    "$Type": "org.OGC.STA.HistoricalLocation",
    "$NavigationPropertyBinding": {
        "Locations": "Locations",
        "Thing": "Things"
    }
},
"Locations": {
    "$Collection": true,
    "$Type": "org.OGC.STA.Location",
    "$NavigationPropertyBinding": {
        "Things": "Things",
        "HistoricalLocations": "HistoricalLocations"
    }
},
"Observations": {
    "$Collection": true,
    "$Type": "org.OGC.STA.Observation",
    "$NavigationPropertyBinding": {
        "Datastream": "Datastreams",
        "proximateFeatureOfInterest": "Features"
    }
},
"ObservedProperties": {
    "$Collection": true,
    "$Type": "org.OGC.STA.ObservedProperty",
    "$NavigationPropertyBinding": {
        "Datastreams": "Datastreams"
    }
},
"Sensors": {
    "$Collection": true,
    "$Type": "org.OGC.STA.Sensor",
    "$NavigationPropertyBinding": {
```

```
                    "Datastreams": "Datastreams"
                }
            },
            "Things": {
                "$Collection": true,
                "$Type": "org.OGC.STA.Thing",
                "$NavigationPropertyBinding": {
                    "Datastreams": "Datastreams",
                    "Locations": "Locations",
                    "HistoricalLocations": "HistoricalLocations"
                }
            }
        }
    }
}
```

Listing C.1 — A CSDL example for the SensorThings API Core

D

ANNEX D (INFORMATIVE) REVISION HISTORY

ANNEX D (INFORMATIVE) REVISION HISTORY

Table D.1 — Revision History

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2023 — 2025	0.1	Hylke van der Schaaf and others	all	Complete reimagining of the Standard.
2025-11-15	0.2	Hylke van der Schaaf	all	Addressed comments from OAB review.
2026-01-16	0.3	Hylke van der Schaaf	all	Improved the consistency of the use of name of the standard
2026-02-05	0.4	Hylke van der Schaaf	Clause 7.7.3, Clause 8	Changed cardinality of Datastream-UltimateFeatureOfInterest to many-to-many and renamed to UltimateFeaturesOfInterest
2026-02-05	0.4	Hylke van der Schaaf	Clause 11.4	Fixed subscription example missing the `@id` attribute
2026-08-31	0.4	Hylke van der Schaaf	Clause 9.10.9	Fixed missing `s` in `Features(1)` in examples