OGC® DOCUMENT: 24-032

External identifier of this OGC® document: http://www.opengis.net/doc/{doc-type}/{standard}/{m.n}

Open Geospatial Consortium

OGC SENSORTHINGS API EXTENSION: WEBSUB 1.0

STANDARD
Implementation

DRAFT

Version: 0.2
Submission Date: 2024-07-03
Approval Date: 2024-12-11
Publication Date: 2029-12-31
Editor: Andreas Matheus,

CONTENTS

I. ABSTRACT vi

II. KEYWORDS vi

III. PREFACE vii

IV. SECURITY CONSIDERATIONS viii

V. SUBMITTING ORGANIZATIONS ix

VI. SUBMITTERS ix

PREFACE 2

CONFORMANCE 4

[_DISCOVERY_CONFORMANCE_CLASS_MANDATORY]. Discovery Conformance Class (mandatory) 4

[_ODATA_CONFORMANCE_CLASS_OPTIONAL]. ODATA Conformance Class (optional) 6

[_SUBSCRIPTION_CONFORMANCE_CLASS_HUB_MANDATORY]. Subscription Conformance Class — Hub (mandatory) 7

[_NOTIFICATION_CONFORMANCE_CLASS_HUB_MANDATORY]. Notification Conformance Class — Hub (mandatory) 7

[_LANDING_PAGE_SENSORTHINGS_API_MANDATORY]. Landing Page SensorThings API (mandatory) 8

[_API_KEY_CONFORMANCE_CLASS_OPTIONAL]. API-Key Conformance Class (optional) 8

1. SCOPE 10

2. CONFORMANCE 12

3. NORMATIVE REFERENCES 14

4. TERMS AND DEFINITIONS 16

5. CONVENTIONS 18

5.1. Identifiers 18

6. WEBSUB EXTENSION 20

6.1. Introduction to W3C WebSub 20

6.2. Overview 21

6.3. Discovery, Subscription and Notification 22

6.4. Why one should use STA-WebSub 24

7. TOPICS FOR DISCUSSION 26
7.1. Discussion of the implementation options for the ODATA query handling. 26
7.2. Discussion of X-Hub-Signature and Api-Key 27
7.3. Discussion of Hub Authentication 27

8. APPENDIX: ODATA HANDLING IMPLEMENTED IN THE HUB 29
8.1. Subscription 30
8.2. Webhook Authentication 30

9. CLAUSE CONTAINING NORMATIVE MATERIAL 32
9.1. Requirement Class A or Requirement A Example 32

10. MEDIA TYPES FOR ANY DATA ENCODING(S) 35

ANNEX A (INFORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE (NORMATIVE) 37
A.1. Conformance Class A 37

ANNEX B (INFORMATIVE) TITLE 40

ANNEX C (INFORMATIVE) REVISION HISTORY 42

LIST OF TABLES

Table 1 33
Table C.1 42

LIST OF FIGURES

Figure 1 — W3C WebSub Flow Diagram 21
Figure 2 — STA-WebSub Flow Diagram 22
Figure 3 — Discovery returns <Link/>; rel="self" 5
Figure 4 — Discovery does not return <Link/>; rel="self" but <Link/>; rel="help" 5
Figure 5 — Discovery returns <Link/>; rel="self" 6
Figure 6 — Discovery returns <Link/>; rel="self" 7
Figure 7 — MQTT notification delivery by the STA-WebSub Hub 8
Figure 8 29

LIST OF RECOMMENDATIONS

REQUIREMENTS CLASS 1 32
REQUIREMENT 1 32
REQUIREMENT 2 33

I ABSTRACT

STA-WebSub — SensorThings API extension WebSub — defines an additional SensorThings API capability that allows the distribution of updates via HTTP(S) using Webhook as defined in the W3C WebSub Recommendation. A subscriber can fetch base data from a SensorThings API service and then can use the same identical URL to subscribe for updates. This allows any subscriber to prevent polling a SensorThings API service, as any updates — according to the URL used — get submitted to the subscriber's Webhook when the update event is triggered. The use of this WebSub extension is also developer friendly, as subscribers only need to setup a W3C WebSub compliant Webhook using HTTP(S). The SensorThings API MQTT protocol is not exposed to the subscriber; it remains internal between the SensorThings API service and the associated STA-WebSub Hub(s). Any SensorThings API service, supporting MQTT topic pattern to include an ODATA query, allows the subscriber to subscribe for updates by defining actual triggering conditions using `$filter`. Using the ODATA query also supports the subscriber to get exactly that data and structured as it is fit for purpose using `$select` and `$expand`.

The use of STA-WebSub improves the flexible use a SensorThings API service building asynchronous workflows. The ability to define trigger conditions and to specify the event data structure that is pushed over HTTP(S) POST enables a fit-for-purpose processing of events using workflows. From a subscriber's point of view, only HTTP requests have to be implemented (HTTP GET for subscription and HTTP POST for receiving event data); the use of the MQTT protocol is used between the SensorThings Service and the Hub only. This strengthen the ability to take load off the MQTT broker as only the associated Hub may subscribe to topics. Also, access control can be implemented in the discovery functionality which controls the subscription based on business or access control policies.

II KEYWORDS

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

ogcdoc, OGC document, OGC Standard, API, SensorThings, WebSub

III PREFACE

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

Any MQTT based event system is susceptible to denial of service attacks when to many clients subscribe to topics with high frequency update (`chatty topics`). For example, subscription to the topic `v1.1/Observations` brings the danger that the SensorThings API service cannot deliver all events to all subscribed clients when the update frequency gets too high. One needs to keep in mind that the MQTT broker is also responsible for receiving observations from very many sensors.

Mitigation to this attack vector exist by disallowing to subscribe to `chatty topics` but in case of the `v1.1/Observations` topic, this is not a good idea as most of the interesting updates happen on this topic! The other mitigation may be to reduce the number of subscribing clients.

This WebSub extension mitigates the denial of service risk allowing only associates Hubs to subscribe to MQTT topics and which topics are supported for subscription can be controlled via the discovery logic. Also, the distribution of the update notifications is not done by the SensorThings service itself; The Hub instance(s) does(do) that.

Any implementation that is compliant with the `ODATA` conformance class, is particularly susceptible to denial of service. To prevent that an attacker can achieve a denial of service attack by subscribing with a complex ODATA query, an implementation of the conformance class `ODATA` should carefully analyze the query before accepting the subscription request. The SensorThings API STA-WebSub discovery logic should validate the subscription carefully.

Any SensorThings API service should deny MQTT subscription requests that do not origin from an associated Hub. This guarantees that an attacker cannot simply bypass the Hub and swamp the SensorThings API service directly with bogus subscriptions.

The proper network separation of SensorThings API service and Hub — e.g. in a fast private network — allows to establish a clever load distribution between a SensorThings API service and as many hubs as needed. This self-adaptivity to scaling mitigates the likelihood for denial of service attacks even with many subscribed clients and complex topics.

In any case, the Hub must validate a subscription with the STA-WebSub Service. This can be achieved via the regular discovery request (HTTP Head request with the topic URL).

To prevent bogus subscriptions, a Hub may require authentication for the subscribe / unsubscribe API.

V SUBMITTING ORGANIZATIONS

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

- Secure Dimensions GmbH
- Centre de Recerca Ecològica i Aplicacions Forestals (CREAF)
- Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.

VI SUBMITTERS

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

NAME	REPRESENTING	OGC MEMBER
Andreas Matheus	Secure Dimensions GmbH	Yes
Joan Maso	Centre de Recerca Ecològica i Aplicacions Forestals (CREAF)	Yes
Hylke van der Schaaf	Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.	Yes

PREFACE

PREFACE

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.

CONFORMANCE

CONFORMANCE

This extension defines the following conformance classes:

[_discovery_conformance_class_mandatory]. Discovery Conformance Class (mandatory)

The `Discovery` conformance class (mandatory) is implemented at the SensorThings API service — HTTP interface and supports the W3C WebSub discovery protocol.

A conformant SensorThings service must return the discovery `<Link/>; rel="hub"` to indicate support for STA-WebSub.

The implementation of this conformance class may include policies that regulate for which MQTT topics a Hub may subscribe for. The implementation may therefore return the `<Link/>; rel="self"` as HTTP response headers for a HTTP GET or HEAD request if the associated MQTT topic is accepted as MQTT subscription. In the case where the discovery policy denies a subscription, the `<Link/>; rel="self"` HTTP header will be missing in the response. Instead, the `<Link/>; rel="help"` HTTP header is returned to inform the user or calling implementation why the subscription is not possible.

This conformance class allows the use of subscriptions to `hub.topic` URLs. The transformation into a MQTT topic must be compliant to SensorThings API v1.1 requirement regarding updates via MQTT (section 14.2 — https://docs.ogc.org/is/18-088/18-088.html#req-receive-updates-via-mqtt-receive-updates). A subscription may be based on one of the following topic patterns **excluding** ODATA commands:

- `SERVICE_VERSION/RESOURCE_PATH/COLLECTION_NAME` as defined in 14.2.1
- `SERVICE_VERSION/RESOURCE_PATH_TO_AN_ENTITY` as defined in 14.2.2
- `SERVICE_VERSION/RESOURCE_PATH_TO_AN_ENTITY/PROPERTY_NAME` as defined in 14.2.3

The following sequence diagram illustrates the discovery for the URL `http://localhost/sta/v1.1/Observations`

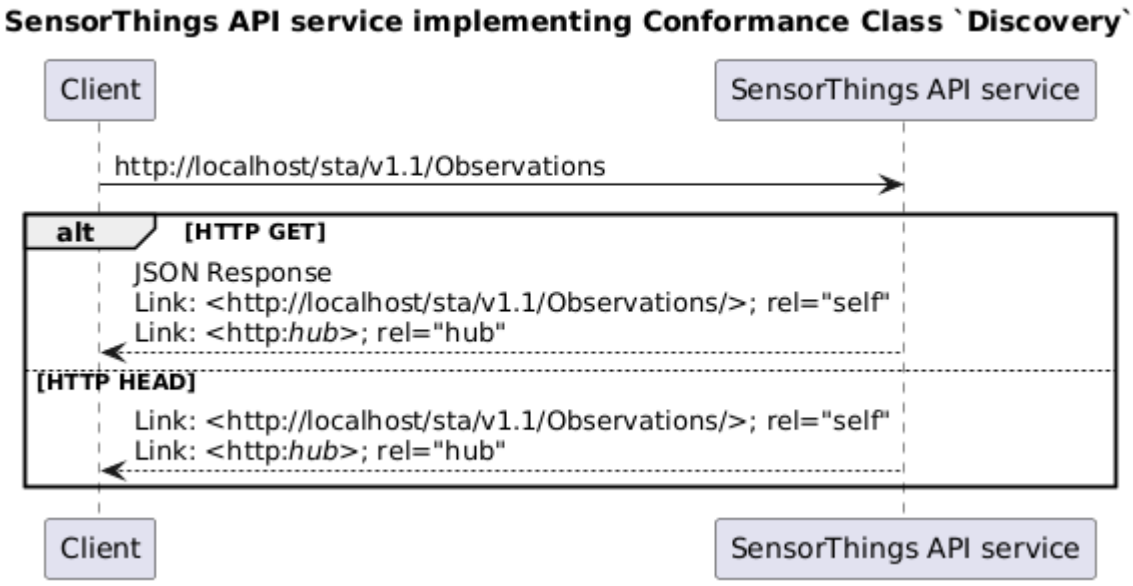

Figure 3 — Discovery returns `<Link/>; rel="self"`

The following sequence diagram illustrates the discovery for the URL `http://localhost/sta/v1.1/Observations?$filter=Datastream/unitOfMeasurement/definition eq 'https://qudt.org/vocab/unit/DEG_C' and Datastream/ObservedProperty/definition eq 'http://vocabs.lter-europe.net/EnvThes/22035`

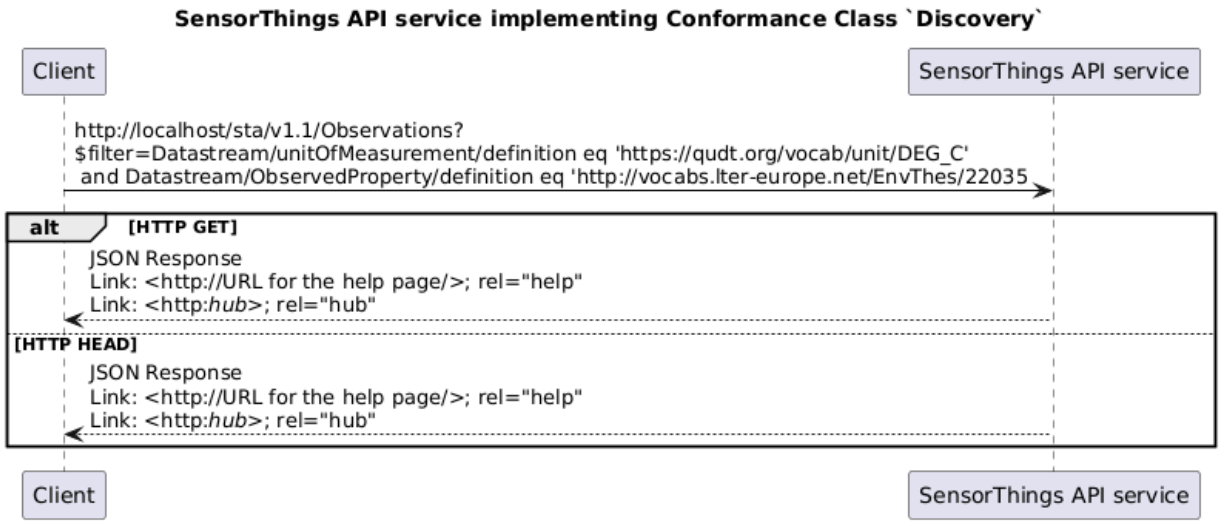

Figure 4 — Discovery does not return `<Link/>; rel="self"` but `<Link/>; rel="help"`

[_odata_conformance_class_optional]. ODATA Conformance Class (optional)

This conformance class extends the SensorThings API v1.1 requirement regarding updates via MQTT (section 14.2 — https://docs.ogc.org/is/18-088/18-088.html#req-receive-updates-via-mqtt-receive-updates) and allows the use of ODATA commands as expressed in the HTTP query part.

- `SERVICE_VERSION/RESOURCE_PATH/COLLECTION_NAME` as defined in 14.2.1 can be extended with a `?` followed by any valid ODATA commands — e.g. `v1.1/Datastreams(1)/Observations?$filter=result gt 30`
- `SERVICE_VERSION/RESOURCE_PATH_TO_AN_ENTITY` as defined in 14.2.2 can be extended with a `?` followed by any valid ODATA commands — e.g. `v1.1/Observations?$select=result`

The use of MQTT topics that include ODATA commands like `$filter`, `$select` and `$expand` may trigger that the discovery response does not include the `<Link/>; rel="self"`.

The following sequence diagram illustrates the discovery for the URL `http://localhost/sta/v1.1/Observations?$filter=Datastream/unitOfMeasurement/definition eq 'https://qudt.org/vocab/unit/DEG_C' and Datastream/ObservedProperty/definition eq 'http://vocabs.lter-europe.net/EnvThes/22035`

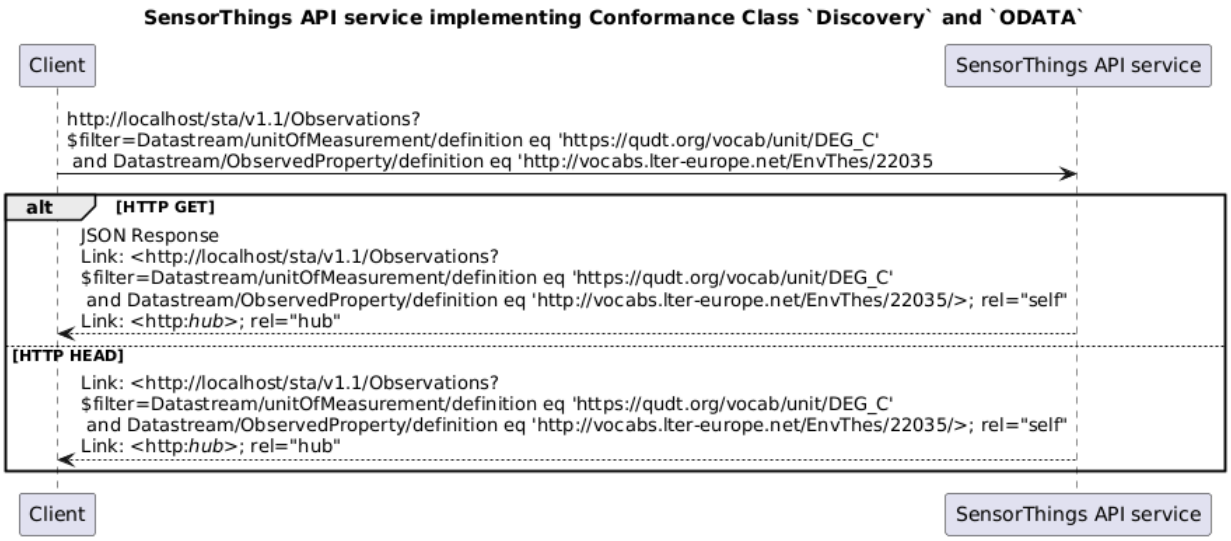

Figure 5 — Discovery returns `<Link/>; rel="self"`

[_subscription_conformance_class_hub_mandatory]. Su
Conformance Class — Hub (mandatory)

The STA-WebSub Hub, associated to a SensorThings API service, supports the W3C subscribe / unsubscribe protocol and transforms a subscription topic URL (`hub.topic`) into a MQTT topic. A compliant implementation must know how to transform the absolute HTTP(S) URL into the corresponding MQTT topic for the associated SensorThings API service.

The Hub uses the MQTT protocol to subscribe and unsubscribe with the MQTT broker of the associated SensorThings API service.

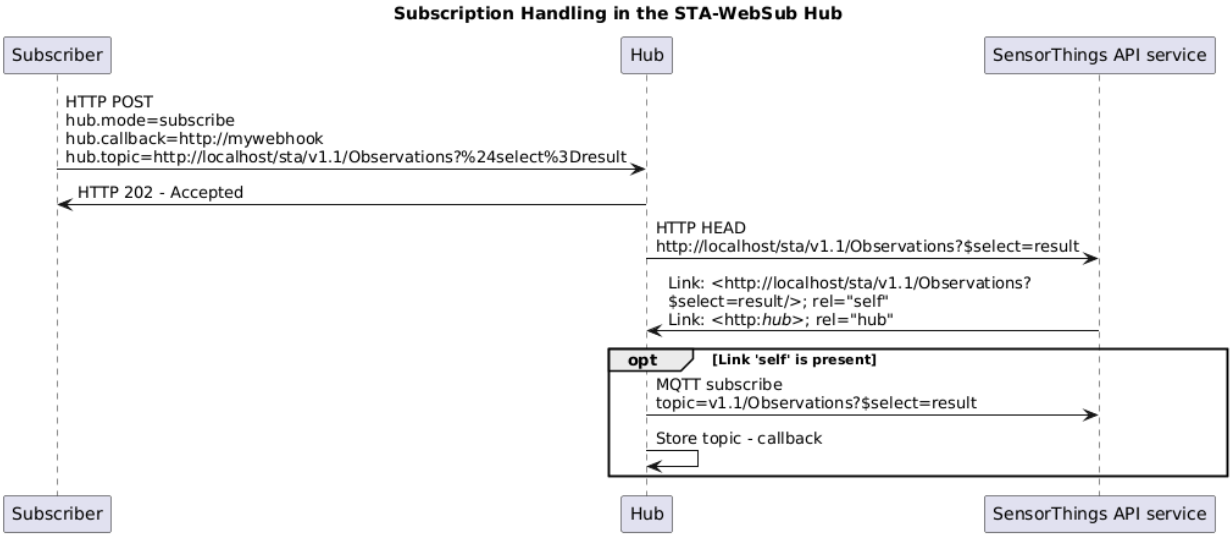

Figure 6 — Discovery returns `<Link/>; rel="self"`

[_notification_conformance_class_hub_mandatory]. No
Conformance Class — Hub (mandatory)

The STA-WebSub Hub supports receiving MQTT notifications from the MQTT broker associated with the SensorThings API service. A compliant implementation distributes the notification content to the subscribed subscribers using HTTP(S) as defined by the W3C WebSub Recommendation.

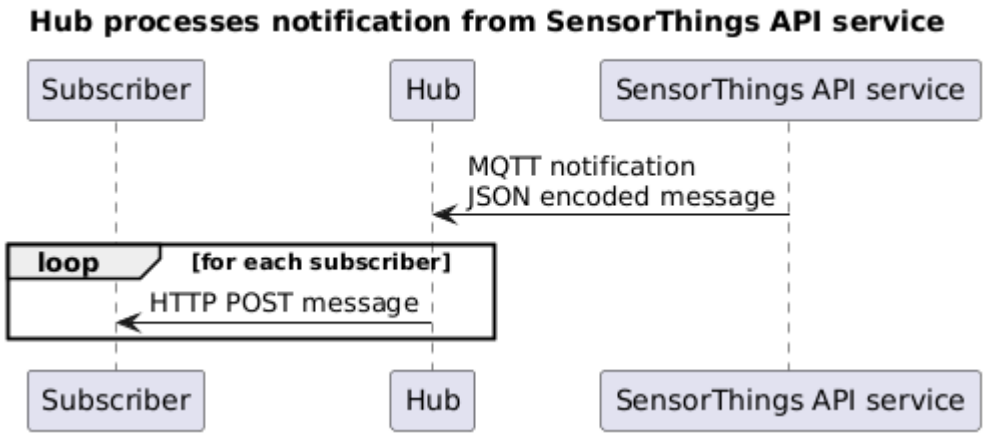

Figure 7 — MQTT notification delivery by the STA-WebSub Hub

[_landing_page_sensorthings_api_mandatory]. Landing Page SensorThings API (mandatory)

The SensorThings API service landing page must list all implemented conformance classes.

[_api_key_conformance_class_optional]. API-Key Conformance Class (optional)

The `API-Key` conformance class (optional) allows that a subscriber protects a HTTP(S) Webhook with an HTTP `API-Key` or `X-API-Key` header. This standard defines this ability in addition to the subscription parameters defined in W3C WebSub. The subscription parameter `webhook.api_key` can be used to trigger that the Hub includes the HTTP header `API-Key` and the or `webhook.x_api_key` can be used to trigger that the Hub includes the HTTP header `X-API-Key`. A subscription renewal may also update the API-Key.

1 SCOPE

1 SCOPE

The W3C WebSub Recommendation does not specify the protocol between a Hub and a Publisher (here the SensorThings API service) — see §6 for details.

This Standard defines how to adopt the W3C WebSub Recommendation with an OGC SensorThings API service. In particular this Standard defines the protocol between a Hub and the MQTT broker that is part of the SensorThings API service.

2

CONFORMANCE

2 CONFORMANCE

This standard defines XXXX.

Requirements for N standardization target types are considered:

- AAAA
- BBBB

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

In order to conform to this OGC® interface standard, a software implementation shall choose to implement:

- Any one of the conformance levels specified in Annex A (normative).
- Any one of the Distributed Computing Platform profiles specified in Annexes TBD through TBD (normative).

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: 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.

WebSub, January 23, 2018. https://www.w3.org/TR/websub

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.

This document uses the terms defined in Sub-clause 5.3 of [OGC06-121r9], which is based on the ISO/IEC Directives, Part 2, Rules for the structure and drafting of International Standards. In particular, the word “shall” (not “must”) is the verb form used to indicate a requirement to be strictly followed to conform to this standard.

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

4.1. example term

term used for exemplary purposes

Note 1 to entry: An example note.

Example Here’s an example of an example term.

[**SOURCE:**]

5

CONVENTIONS

5 CONVENTIONS

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

5.1. Identifiers

The normative provisions in this standard are denoted by the URI

`http://www.opengis.net/spec/{standard}/{m.n}`

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

6

WEBSUB EXTENSION

6 WEBSUB EXTENSION

The SensorThings API WebSub Extension as defined in this OGC Standard is based on the W3C WebSub Recommendation. This Standard leverages the WebSub Recommendation, §6 openness to connect the Hub with the MQTT broker of the SensorThings API service. The STA-WebSub does not modify the protocol defined in the W3C Recommendation. Therefore, any W3C WebSub compliant `Subscriber` does work with STA-WebSub without modifications.

A compliant implementation consists of three parts:

- STA-WebSub SensorThings service: The `Discovery` protocol must be implemented. This should (but not necessarily must) be implemented as an extension to the SensorThings API service. In any way (internal or external), the discovery functionality controls the subscription capabilities by returning the WebSub HTTP response headers.
- STA-WebSub Hub: The `Subscription` protocol must be implemented as part of the Hub. The implemented logic transforms a HTTP request URL for a SensorThings API service into an MQTT topic (for that same service). Also, the implementation must support to subscribe to MQTT for the topic.
- STA-WebSub Hub: The `Notification` of events takes place via MQTT. A STA-WebSub Hub must therefore 'listen' to MQTT events received from the associated SensorThings API service and distribute the event data to all subscribers using HTTP(S) POST.

An implementation of the STA-WebSub extension is available as open source:

- The STA-WebSub Hub is available as open source from Github: https://github.com/securedimensions/WebSub-Hub
- The `Discovery` protocol is implemented as a plugin to the SensorThings API service FROST-Server: https://github.com/securedimensions/FROST-Server-WebSub
- The FROST-Server is available as open source from Github: https://github.com/FraunhoferIOSB/FROST-Server

6.1. Introduction to W3C WebSub

WebSub is a W3C Recommendation that defines a protocol how event triggered notifications are discovered, subscribed to and delivered. The recommendation defines three roles: The Subscriber is the actor that wants to receive data updates; the Publisher is the service that offers event triggered data notifications; the Hub is the service that accepts subscriptions from Subscribers about topics and delivers the data received from the Publisher that is associated with a subscribed topic.

The following figure illustrates the basic functioning of the W3C WebSub Recommendation.

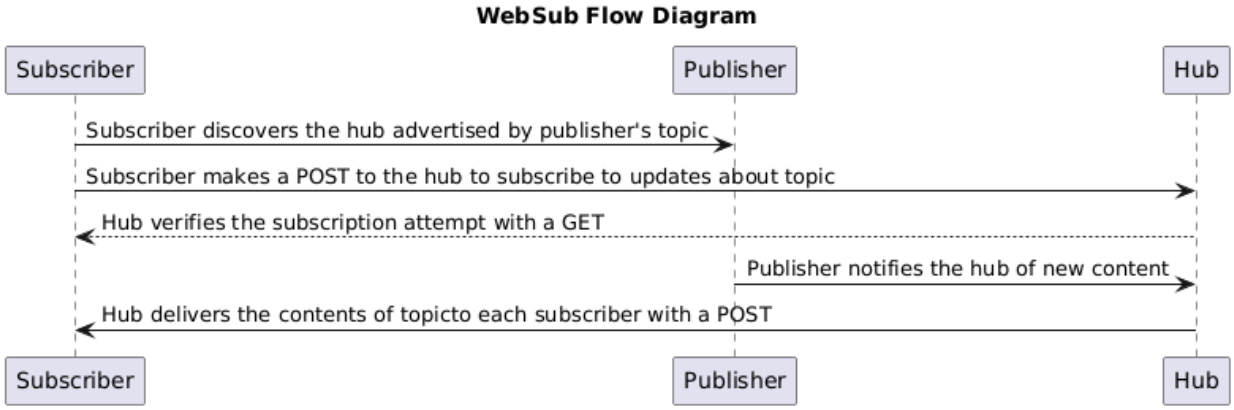

Figure 1 — W3C WebSub Flow Diagram

Even though the W3C WebSub Recommendation defines the protocol details for discovery (Subscriber → Publisher) and content distribution (Hub → Subscriber), the protocol how the Publisher and Hub communicate is deliberately unspecified (see §6 WebSub Recommendation). This openness is used by this Standard to connect a W3C WebSub compliant Hub with the OGC SensorThings API compliant service implementation.

6.2. Overview

STA-WebSub — SensorThings API extension WebSub — defines additional SensorThings API capabilities that allow the propagation of updates via HTTP(S) using Webhook as defined in W3C WebSub Recommendation. A subscriber can fetch base data from a compliant SensorThings API service and use the same identical URL to subscribe for updates. The use of the WebSub extension allows a subscriber to prevent polling a SensorThings API service to detect data changes. Updates — according to the W3C WebSub topic URL — get pushed to the subscriber's Webhook using HTTP POST when the update event is triggered. The use of this WebSub extension is also developer friendly, as subscribers only need to setup a W3C WebSub compliant Webhook using HTTP(S). The SensorThings API MQTT protocol is not exposed to the subscriber; it remains internal between the SensorThings API service and the associated Hub(s). Any implementation of the WebSub extension must support a MQTT topic pattern as defined in SensorThings API v1.0 or v1.1 extended by an ODATA query. The functionality to process MQTT topics with ODATA query can be implemented in the SensorThings API service or the STA-WebSub Hub. This feature entitles the subscriber to subscribe for updates by defining actual triggering conditions using the ODATA `$filter`. Using the ODATA `$select`, `$expand` support the subscriber to fix exactly the data structure that gets pushed to the Webhook of the Subscriber. The ability to craft notification conditions in combination to compose exactly the data structure needed for a notification is a powerful ability to trigger fit for purpose workflow executions connected to a subscriber's Webhook.

The following high-level flow diagram illustrates the protocol adoption for the STA-WebSub implementation.

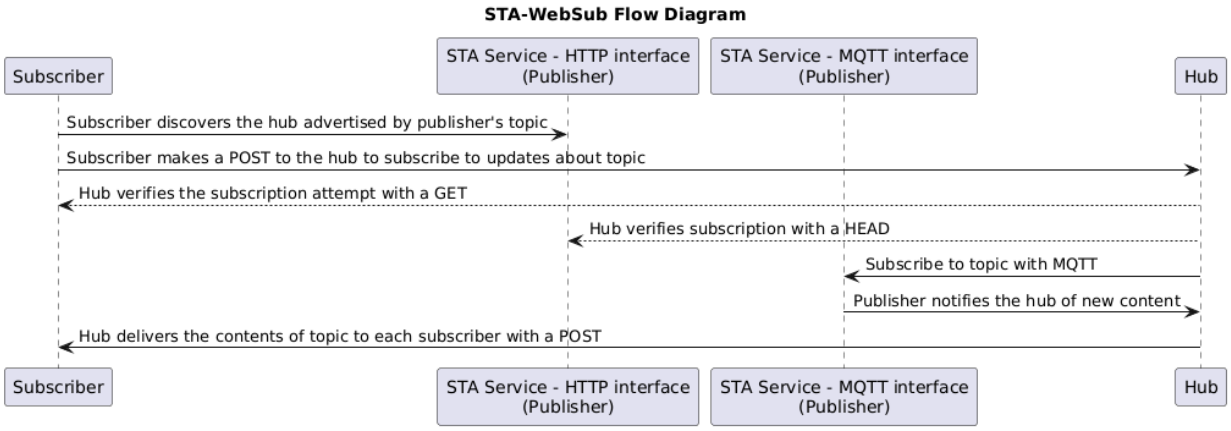

Figure 2 — STA-WebSub Flow Diagram

As illustrated in the flow diagram above, an implementation of this Standard can take under consideration to separate the SensorThings API service into an HTTP and MQTT part. The HTTP part, aka the `Publisher` must support the W3C WebSub `Discovery` protocol. The Hub must implement the MQTT protocol to subscribe for topics and receive notifications. In addition, the Hub must implement the logic to transform a topic URL into a SensorThings MQTT topic.

6.3. Discovery, Subscription and Notification

For adopting the W3C WebSub protocol for a SensorThings API service, certain functional requirements are to be defined regarding discovery, subscription and notification.

6.3.1. Discovery

A STA-WebSub compliant SensorThings API Publisher supports the W3C WebSub discovery by adding the `Link` headers `rel="hub"` and `rel="self"` to the HTTP response. The W3C WebSub Recommendation further requires that the `Link` headers are returned either as HTTP response headers or inline to a XHTML encoded response. As the typical response encoding for SensorThings API is not XHTML — it usually is either JSON or GeoJSON — the WebSub extension requires that the `Link` headers are returned as HTTP response headers. The W3C WebSub Recommendation foresees that these discovery `Link` s will be returned on a HTTP GET or HEAD request. To meet the HTTP method requirement for discovery as specified in the W3C WebSub Recommendation, the WebSub extension requires that a SensorThings API implementation supports the HTTP HEAD method in addition to the already supported HTTP GET method.

6.3.2. Subscriptions

A compliant STA-WebSub Hub, as advertized by a compliant SensorThings API service is capable to transform the W3C WebSub `hub.topic` expressed as a HTTP(S) URL into a MQTT topic pattern accepted by the MQTT broker associated with the SensorThings API service. As the SensorThings API service and the Hub are working in close relation, this transformation should not be too difficult.

In case the Hub receives an unsubscribe request from the subscriber, the Hub should verify the intent and unsubscribe from the SensorThings API service for the corresponding MQTT topic.

6.3.3. Notifications

Once the Hub has subscribed to a MQTT topic, it awaits MQTT notifications from the MQTT broker of the SensorThings API service. In case a message arrives, the Hub delivers the message to the subscribed callback URLs (the subscribers' Webhooks).

Example: The Hub must support the subscriber to determine authenticity of the received message. As defined by the W3C WebSub recommendation, the subscription to a topic may include the `hub.secret`. In this case the Hub adds the HMAC HTTP response header `X-Hub-Signature` to the HTTP(S) POST headers as defined in the W3C WebSub Recommendation when delivering the message to the Webhook.
The Hub should follow closely all security considerations outlined in the W3C WebSub Recommendation.

6.3.4. Discovery Error Handling

A STA-WebSub Hub may receive a subscription via the `hub.topic` that transforms to a MQTT topic which may not be accepted by the SensorThings API MQTT broker. Which topic URLs are accepted is deployment specific. For example, the subscription to `/Observations` may produce a too high load. Also, a URL may include ODATA commands like `$expand` or `$filter` that are not supported. In these cases, the discovery response will not include the `Link rel="self"` header. This behavior is perfectly compliant with the WebSub Recommendation. But any caller (user or service/process) may wonder why.

The W3C Recommendation does not specify any error handling. To be exact, the fact that no `Link rel="self"` is returned is not an error. Therefore, the use of HTTP 4xx status codes is not appropriate. However, there should be guidance why the link header is missing.

This STA-WebSub Standard introduces the use of the `Link rel="help"` header. The URL for this relationship points to a (static) help page that explains why the discovery response is missing the `Link rel="self"` header.

6.4. Why one should use STA-WebSub

According to the SensorThings API v1.1 Standard, any notification will origin the MQTT broker of the SensorThings API service. The STA-WebSub extension as defined in this OGC Standard supports the distribution of the MQTT events via the HTTP(S). Therefore, it is not the MQTT broker that sends the events 1→n, it is the Hub that does that. This separation of duty brings important improvements regarding use, security and scalability:

- The use of the STA-WebSub extension makes the MQTT protocol internal between the MQTT broker of the SensorThings API service and the associated Hub(s). This allows to restrict MQTT subscriptions origin the associated STA-WebSub Hub. Also, the use of discovery policies allow to implement flexible and fine grained access control regarding the subscriptions done by the Hub.
- The fact that the MQTT protocol is internal between the Hub and the SensorThings Service simplifies the use for subscribers to well-known infrastructure patterns like Webhook, essentially using a W3C WebSub compliant HTTP(S) endpoint listening for GET and POST requests.
- The separation of duty for sending update 'messages' to subscribers between the SensorThings API and the Hub improves scalability. The SensorThings API service only delivers the topic updates to associated Hub(s) using MQTT. The Hub(s) then optionally processes and distributes the 'response' or 'message' to subscribers using well understood cloud-scaling code stacks.
- The ability that subscribers can determine the notification conditions (i.e. using `$filter`) and the data structure of the notification (i.e. using `$select` and `$expand`) improves the usability over predefined MQTTP topics. How flexible a subscriber can get is controlled by the discovery functionality.

7

TOPICS FOR DISCUSSION

7 TOPICS FOR DISCUSSION

7.1. Discussion of the implementation options for the ODATA query handling.

Any compliant implementation of the functional unit SensorThings API and Hub service must be capable to accept subscriptions for topics, expressed as a HTTP(S) URL that may contain an ODATA query. For each `rel="self"` link exposed by the SensorThings API service, the hub implementation knows how to transform a subscription topic received via the `hub.topic` parameter into a MQTT topic pattern.

For any topic pattern that includes an ODATA query, either the SensorThings API service or the Hub must implement the functionality to apply the ODATA query. If arguments exist that disallow to upgrade the core functionality of a SensorThings API deployment to accept a MQTT topic including an ODATA query, the Hub must implement the ODATA query processing. In such an implementation, the Hub may receive a subscription where the `hub.topic` parameter value is a URL including an ODATA query like `http://localhost/sta/v1.1/Observations?v1.1/Observations?$filter=Datastream/unitOfMeasurement/definition eq 'https://qudt.org/vocab/unit/DEG_C' and Datastream/ObservedProperty/definition eq 'http://vocabs.lter-europe.net/EnvThes/22035' and result gt 30`. As the SensorThings API service would not accept an MQTT subscription for the topic `v1.1/Observations?v1.1/Observations?$filter=Datastream/unitOfMeasurement/definition eq 'https://qudt.org/vocab/unit/DEG_C' and Datastream/ObservedProperty/definition eq 'http://vocabs.lter-europe.net/EnvThes/22035' and result gt 30`, the Hub must remove the ODATA query (`$filter=...`) and subscribe for the topic `v1.1/Observations`. Upon receiving a notification for this topic, the Hub could issue a HTTP(S) request to the SensorThings API service using the `@iot.selfLink` from the notification and attach the stored ODATA query. In this example, the Hub would issue a HTTP(S) GET request with a URL similar to this `http://localhost/sta/v1.1/Observations(4711)?v1.1/Observations?$filter=Datastream/unitOfMeasurement/definition eq 'https://qudt.org/vocab/unit/DEG_C' and Datastream/ObservedProperty/definition eq 'http://vocabs.lter-europe.net/EnvThes/22035' and result gt 30`, assuming the notification was for `Observations(4711)`. In case the response is empty (`[]` indicates a false positive notification) or does not contain a `selfLink`, the Hub must not distribute the response to the HTTP(S) request. In case the response includes an `@iot.selfLink`, the Hub must distribute the response to the subscribed Webhook(s).

It is clear that such an implementation of a Hub may cause a lot of unnecessary network traffic between the Hub and the SensorThings API service as well as HTTP(S) requests by the Hub to apply the ODATA query. In particular subscriptions to `v1.1/Observations` are likely to be very chatty. It may therefore be a good idea to carefully judge if the implementation of the ODATA query handling gets implemented in the Hub.

7.2. Discussion of X-Hub-Signature and Api-Key

One fundamental (runtime) functionality for a WebSub Hub is the ability to deliver a MQTT notification message to all subscribers. To do this, an implementation would probably cache the message and then use HTTP POST to all webhooks for all subscribers.

The W3C WebSub Recommendation defines the use of `X-Hub-Signature` that enables the ability to check the authenticity of a received message (subscriber can check on the Webhook if the content was sent by the expected hub). The use of HMAC signatures introduces only a small burden to the hub's CPU when calculating the HMAC value. Also adding the HMAC to the `X-Hub-Signature` header does also not introduce any burden on Hub side.

However, the validation of the HMAC requires that a subscriber's Webhook receives the entire message. In case a fraudulent message was received, it must be received in full which may consume quite an amount of resources at the subscriber's side. This vulnerability to DoS attacks by consuming too many resources via high-frequency of fraudulent messages can be eliminated if the subscriber's Webhook is protected by an `X-Api-Key`. Like the use of `hub.secret` to share the secret for HMAC generation, a subscription could include an additional parameter, i.e. `callback.x_api_key` that must be used by the Hub when POSTing the notification message. The use of `X-Api-Key` would prevent that fraudulent messages get processed in the first place.

7.3. Discussion of Hub Authentication

A Hub implementation may restrict the use of the subscribe/unsubscribe API to authenticated users. The definition of the capabilities of a multi-tenant Hub is outside of this Standard.

8

APPENDIX: ODATA HANDLING IMPLEMENTED IN THE HUB

8 APPENDIX: ODATA HANDLING IMPLEMENTED IN THE HUB

A SensorThings implementation of the `ODATA` conformance class that does **not** support subscriptions where the topic pattern includes an ODATA query returns a `<Link>; rel="self"` the Hub must remove the ODATA query to create a valid topic pattern.

The following sequence diagram illustrates the subscription for the URL `http://localhost/sta/v1.1/Observations?$filter=Datastream/unitOfMeasurement/definition eq 'https://qudt.org/vocab/unit/DEG_C' and Datastream/ObservedProperty/definition eq 'http://vocabs.lter-europe.net/EnvThes/22035` assuming a subscription including the ODATA query is **not** supported.

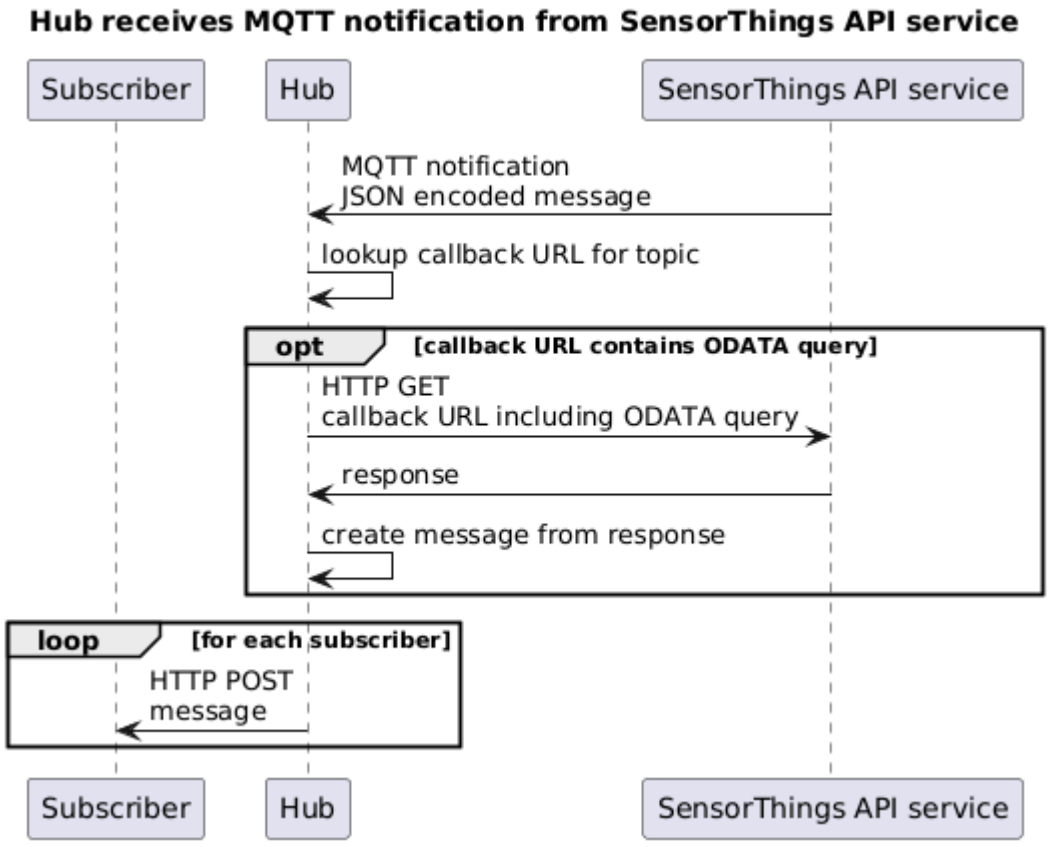

Figure 8

8.1. Subscription

As defined in the W3C WebSub Recommendation, a subscriber sends a subscription request to the Hub using the URL from the `<Link/> rel="hub"`. The subscription topic (`hub.topic`) is identical to the URL returned with the `<Link/> rel="self"`.

Upon receiving the subscription request, the Hub should validate the intend of the subscriber as defined in the W3C WebSub Recommendation. Upon success, the Hub would subscribe to the SensorThings API service via MQTT using one of the supported topic patterns as introduced above.

To prevent bogus subscription attempts, the Hub should require authentication and validate the intend of the subscriber as defined in the W3C WebSub Recommendation. This essentially requires a HTTP GET request to the subscriber's Webhook URL. If that is cleared, the Hub should subscribe with the SensorThings API service and act on the callback to evaluate if the subscription was successful. Any blacklisted topic patterns may cause the MQTT subscription to fail.

8.2. Webhook Authentication

The W3C WebSub Recommendation defines the subscription parameter `hub.secret` to support HMACing of messages pushed to subscribed Webhook(s). Each receiving Webhook can use the HMAC to validate origin and integrity of the received message leveraging the shared secret. The use of HMAC however does not prevent that the Webhook gets executed for bogus messages coming in not origin the expected Hub. An attacker may leverage the 'openness' of the Webhook and push large messages with bogus HMAC to cause a denial of service on the Webhook or achieve execution with rogue data.

The use of the HTTP Request header `X-API-Key` is a common practice to protect a service endpoint from unwanted execution. But, the W3C WebSub Recommendation does not define the option to submit the API-Key value with a subscription request.

The WebSub Extension Conformance Class `Authentication` defines the use of the subscription parameters `hub.api_key` and `hub.x_api_key`. The former is translated in the HTTP request header `API-Key` and the latter into `X-API-Key` with messages POSTed to the Webhook. A Hub that does not implement the `API-Key` conformance class should reject a subscription including the `hub.api_key` or `hub.x_api_key` as it does not make sense to POST messages excluding the associated HTTP header (the Webhook would presumably return HTTP status code 401).

9

CLAUSE CONTAINING NORMATIVE MATERIAL

9 CLAUSE CONTAINING NORMATIVE MATERIAL

Paragraph

9.1. Requirement Class A or Requirement A Example

Paragraph – intro text for the requirement class.

Use the following table for Requirements Classes.

REQUIREMENTS CLASS 1	
OBLIGATION	requirement
DESCRIPTION	Requirements Class • http://www.example.org/req/blah • urn:iso:ts:iso:19139:clause:6
NORMATIVE STATEMENTS	Requirement 1-1 Requirement 1-2

9.1.1. Requirement 1

Paragraph — intro text for the requirement.

Use the following table for Requirements, number sequentially.

REQUIREMENT 1	
OBLIGATION	requirement
STATEMENT	Requirement 'shall' statement

Dictionary tables for requirements can be added as necessary. Modify the following example as needed.

Table 1

NAMES	DEFINITION	DATA TYPES AND VALUES	MULTIPLICITY AND USE
name 1	definition of name 1	float	One or more (mandatory)
name 2	definition of name 2	character string type, not empty	Zero or one (optional)
name 3	definition of name 3	GML:: Point PropertyType	One (mandatory)

9.1.2. Requirement 2

Paragraph — intro text for the requirement.

Use the following table for Requirements, number sequentially.

REQUIREMENT 2	
LABEL	/req/req-class-a/req-name-2
CONDITIONS	1. The process input value is specified in-line in an execute request. 2. The process input is defined as an object according to its schema.
A	The server SHALL support process input values encoded as qualified values.
B	The value of the `value` key SHALL be an *object* instance.

10

MEDIA TYPES FOR ANY DATA ENCODING(S)

10 MEDIA TYPES FOR ANY DATA ENCODING(S)

A section describing the MIME-types to be used is mandatory for any standard involving data encodings. If no suitable MIME type exists in http://www.iana.org/assignments/media-types/index.html then this section may be used to define a new MIME type for registration with IANA.

A

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

A

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

NOTE: Ensure that there is a conformance class for each requirements class and a test for each requirement (identified by requirement name and number)

A.1. Conformance Class A

Example

label	http://www.opengis.net/spec/name-of-standard/1.0/conf/example1
subject	Requirements Class “example1”
classification	Target Type:Web API

A.1.1. Example 1

ABSTRACT TEST A.1	
SUBJECT	/req/req-class-a/req-name-1
LABEL	/conf/core/api-definition-op
TEST PURPOSE	Validate that the API Definition document can be retrieved from the expected location.
TEST METHOD	1. Construct a path for the API Definition document that ends with `/api`. 2. Issue a HTTP GET request on that path 3. Validate the contents of the returned document using test /conf/core/api-definition-success.

A.1.2. Example 2

ABSTRACT TEST A.2	
SUBJECT	/req/req-class-a/req-name-2
LABEL	/conf/core/http
TEST PURPOSE	Validate that the resource paths advertised through the API conform with HTTP 1.1 and, where appropriate, TLS.
TEST METHOD	1. All compliance tests SHALL be configured to use the HTTP 1.1 protocol exclusively. 2. For APIs which support HTTPS, all compliance tests SHALL be configured to use HTTP over TLS (RFC 2818) with their HTTP 1.1 protocol.

B

ANNEX B (INFORMATIVE) TITLE

B

ANNEX B (INFORMATIVE) TITLE

NOTE: Place other Annex material in sequential annexes beginning with “B” and leave final two annexes for the Revision History and Bibliography

C

ANNEX C (INFORMATIVE) REVISION HISTORY

C

ANNEX C (INFORMATIVE) REVISION HISTORY

Table C.1

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2016-04-28	0.1	G. Editor	all	initial version

BIBLIOGRAPHY

BIBLIOGRAPHY