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: 0.3
Submission Date: 2024-07-03
Approval Date: 2024-12-18
Publication Date: 2029-12-31
Editor: Andreas Matheus,

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF RECOMMENDATIONS

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.

The W3C WebSub recommends that the subscriber's callback URL is 'not easily guessable' and that the callback URL is refreshed whenever a subscription is renewed. Following this recommendation makes the use of other authentication mechanisms, i.e. the use of api-key unnecessary. But, for computing environments were is it is not possible or too difficult to follow this recommendation, the STA-WebSub Standard introduces the use of `API-Key` or `X-API-Key` to protect the callback endpoint URL. Likewise the refreshing of the callback URL, it is strongly recommended that a subscriber updates the api-key value with a strong random value each time a subscription is renewed. Leveraging an api-key over HTTPS callback URLs may be preferred over the use of `X-Hub-Signature` when the message size is large. The clear advantage is that neither the Hub nor the Subscriber must calculate a hash over the message. The use of api-key

acts like a secret that only the subscription hub knows — so it proofs authenticity — and HTTPS ensures message integrity which is equivalent to HMAC.

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.

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 two conformance classes.

Requirements for one standardization target type are considered:

- SensorThings API v1.0
- SensorThings API v1.1

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).

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

STA-WEBSUB EXTENSION

6 STA-WEBSUB EXTENSION

The STA-WebSub — SensorThings API WebSub Extension — as defined in this OGC Standard is based on the W3C WebSub Recommendation. A SensorThings API implementation supports two protocols: The HTTP interface and the MQTT interface. STA-WebSub leverages the W3C WebSub openness (see §6 W3C WebSub for details) for specifying the protocol between the Hub and the Publisher using MQTT. In addition it specifies that the SensorThings API HTTP interface implementation supports the W3C WebSub discovery requirements.

A system for STA-WebSub consists of to parts:

- SensorThings HTTP interface:
  - The W3C discovery protocol must be implemented.
  - The STA-WebSub error handling must be implemented.
- Hub:
  - The subscription functionality must be extended to transform a W3C topic URL into a SensorThings MQTT topic and the Hub must use the MQTT protocol to subscribe/unsubscribe for topics.
  - The notification of distribution events takes place via MQTT. A STA-WebSub Hub must therefore listen to MQTT events received from the associated MQTT broker and distribute the event content to all subscribers using HTTP(S) POST.

An implementation of a STA-WebSub system is available as open source:

- The STA-WebSub Hub is available as open source from Github: https://github.com/securedimensions/WebSub-Hub
- The W3C discovery protocol extension for the SensorThings API service is implemented as a plugin to FROST-Server:
  - STA-WebSub plugin: https://github.com/securedimensions/FROST-Server-WebSub
  - The FROST-Server: https://github.com/FraunhoferIOSB/FROST-Server

6.1. Introduction to W3C WebSub

NOTE: It is recommended to first read the W3C WebSub Recommendation.

WebSub is a W3C Recommendation that addresses how to implement asynchronous processing over the Web. First, the discovery protocol allows a Subscriber to check with a Publisher if self-defined topics can be used for subscription. Second, the subscription protocol defines how a Subscriber can interact with a hub to register a topic for receiving notifications afterwards.

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

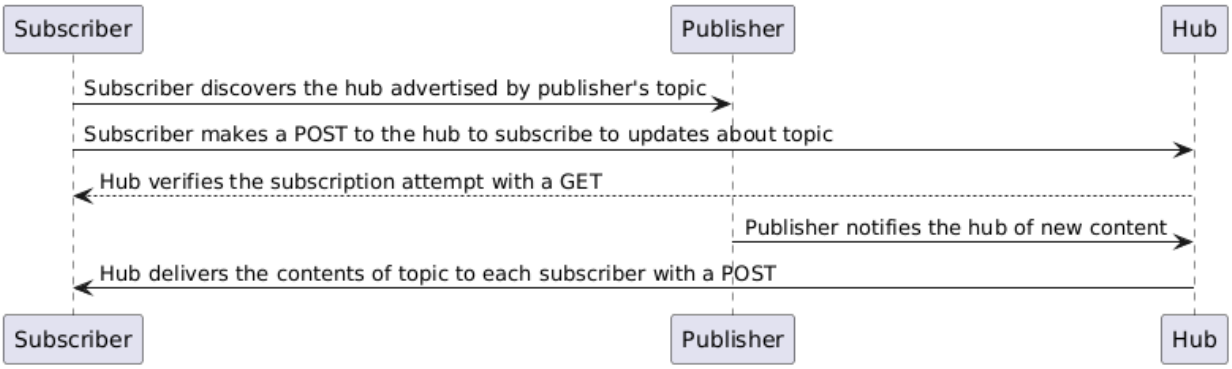


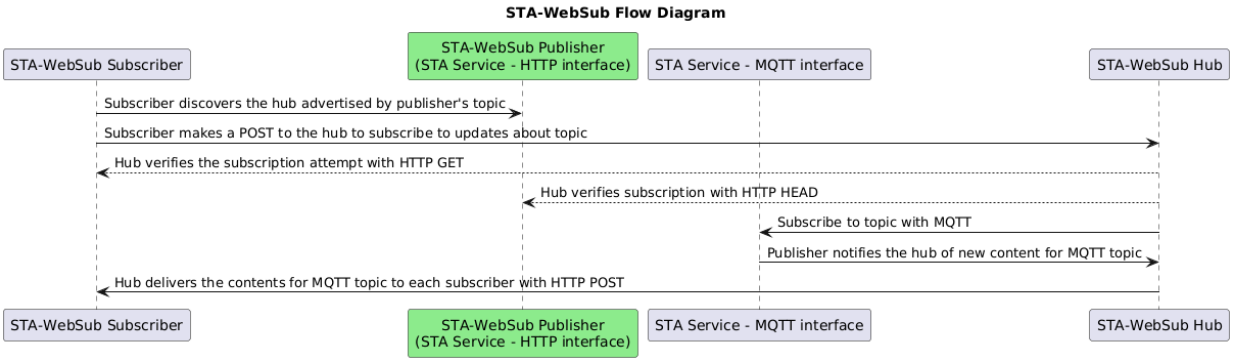
Figure 1 — W3C WebSub Flow Diagram

The notification protocol is deliberately left unspecified (see WebSub §6 and *"Publisher notifies the hub of new content"* as illustrated above) so that basically any protocol can be used between a publisher and a hub. This OGC STA-WebSub Standard leverages this openness to couple the publisher and the hub using MQTT.

6.2. STA-WebSub Overview

The STA-WebSub — SensorThings API extension WebSub — defines additional capabilities in the context of SensorThings API that allow the propagation of updates via HTTP(S) using callback URLs as defined in W3C WebSub Recommendation. The notification event and content is based on MQTT as defined in the SensorThings API Standard.

To make STA-WebSub work, the subscriber, publisher and hub need to implement certain functionalities. In order to understand the requirements better, the following high-level flow diagram illustrates the additions for STA-WebSub:



NOTE: As illustrated in the STA-WebSub flow diagram above, this Standard has implications to the subscriber, the implementations of the publisher and the hub. But only the part of the publisher is considered normative here, as the SensorThings API service is the standardization target. To ensure a working solution, the Appendices B and C define the implementation specific aspects to adapt a W3C Hub and Subscriber for Web-Sub.

Figure 2 — STA-WebSub Flow Diagram

The use of the STA-WebSub extension allows a subscriber to prevent polling a SensorThings API service for detecting data changes. Updates, received by the hub via MQTT, get pushed to the subscriber's callback URL when the update event is triggered. The callback URL can be setup as a W3C WebSub compliant Webhook using HTTP POST.

The use of W3C WebSub does not predefine topics. Instead, the subscriber can discover if a topic (URL) can be used for subscription. This flexibility is important for STA-Web to support the subscriber for defining (i) the actual trigger for a change event and (ii) the data structure for the received content. 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. A SensorThings API service implementation that supports ODATA commands with MQTT topics offers the subscriber to use `$filter` for specifying the event trigger and `$select` and `$expand` to specify the content data structure. 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.

NOTE: The support for `$select` is mandatory for a SensorThings API service implementation. However, the support for `$filter` and `$expand` is optional.

6.3. Benefits of 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.

6.4. Introduction to Discovery, Subscription and Notification

For extending the W3C WebSub protocol to support STA-WebSub, certain functional requirements are to be defined regarding discovery, subscription and notification.

6.4.1. Discovery

A STA-WebSub compliant SensorThings API HTTP interface (aka publisher) supports the W3C WebSub discovery by adding the `Link` headers `rel="hub"` and `rel="self"` to the HTTP response. W3C WebSub 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 STA-WebSub extension requires that the `Link` headers are returned as HTTP response headers. The W3C WebSub foresees further that these discovery `Link` s will be returned on a HTTP GET or HEAD request. To meet the HTTP method requirement for discovery, the STA-WebSub extension requires that a publisher implementation supports the HTTP HEAD method in addition to the already supported HTTP GET method.

6.4.2. Discovery Error Handling

A STA-WebSub hub may receive a subscription request 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 or the use of not supported / not allowed ODATA commands like `$expand` or `$filter` may cause that no `Link rel="self"` is returned. The missing `Link rel="self"` header implies that subscription for such a topic URL is not possible. This behavior is perfectly compliant with the WebSub Recommendation. But any subscriber (user or service/process) may wonder why the discovery response does not include the `Link rel="self"` header.

Technically, the fact that no `Link rel="self"` is returned is not an error. Therefore, the use of HTTP 4xx status codes is not appropriate. Also, the W3C WebSub does not specify any error handling. But to support a subscriber, there should be guidance why the link header `rel="self"` is missing.

This STA-WebSub Standard introduces the use of the `Link rel="help"` header. The URL for this relationship must point to a (static) help page that explains why a subscription to the topic URL is not possible.

6.4.3. Subscriptions

A STA-WebSub Hub 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.

To prevent that a subscription to an unsupported MQTT is possible, the Hub must use the discovery protocol and deny the request if the discovery response does not include the `Link rel="self"` header.

In case the Hub receives an unsubscribe request from the subscriber, the Hub must verify the intent with the subscriber and unsubscribe from the MQTT topic with the associated SensorThings API service.

6.4.4. 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 of a notification event (topic + content), the Hub delivers the content to the subscribed callback URLs (the subscribers' Webhooks).

6.5. X-Api-Key and X-Hub-Signature

One fundamental (runtime) functionality for a WebSub Hub is the ability to deliver a notification content to all subscribers.

The W3C WebSub offers the use of HMAC so that subscribers can validate the authenticity of received messages; basically check that the message was received from the expected hub. Leveraging HMAC also enables the subscriber to validate message integrity which is important when using non secure communication — so HTTP instead of HTTP**S**. The W3C WebSub supports the transmission of the HMAC using the HTTP header `X-Hub-Signature`.

The use of HMAC signatures however introduces a burden to the hub's and subscriber's CPU when calculating the HMAC value. It also requires caching of the message at the hub and the subscriber. This prevents that the subscriber can stream the received message directly into connected workflow processing. But, on the hub side, the caching does not introduce a resource

burden as an implementation would probably cache the message anyway before distributing it to all subscribers.

But because the validation of the HMAC requires that a subscriber receives the entire message, this introduces a denial-of-service vulnerability. Assuming that an attacker would send large fraudulent messages with high frequency (many messages in parallel), the subscriber must process the entire message to determine authenticity.

The W3C WebSub recommends that a callback URL is random and gets refreshed by the subscriber when a subscription is updated. This procedure gains equivalent protection of the callback endpoint to the use of api-key. Therefore, when following this procedure and the callback operates over HTTPS, the use of HMAC is not necessary.

However, when the generation or refreshing of callback URLs is not possible or too difficult, STA-WebSub offers the use of api-key authentication. The use of api-key compensates the use of random callback URLs. And, api-key in combination with HTTP**S** callback URLs is equivalent to the use of HMAC in terms of authenticity and integrity but it is not necessary to calculate HMAC on the hub side and validate on the subscriber side. This reduces the burden of resources on the hub and subscriber side. This is important for hub deployments on the edge.

Like the use of `hub.secret` to share the secret for HMAC generation, a STA-WebSub subscription may include the parameter `hub.api_key` or `hub.x_api_key` to trigger api-key management at the hub. These parameters trigger that the hub adds the HTTP header `Api-Key` or `X-Api-Key` when distributing the content to the subscriber.

7

STA-WEBSUB REQUIREMENTS (NORMATIVE)

7 STA-WEBSUB REQUIREMENTS (NORMATIVE)

A STA-WebSub system requires that the SensorThings API HTTP interface supports the discovery protocol; the hub supports the transformation of topic URLs into MQTT topics, use MQTT for subscription and notifications, and accepts the subscription with api-key; the subscriber supports the use the api-key and the evaluation of the `Link rel="help"`.

The normative section of this OGC Standard is only concerned with the discovery aspect as the standardization target is the SensorThings API implementation. But, to provide guidance for implementing a STA-WebSub system, the Appendices B and C define the requirements for a Subscriber and a Hub.

The STA-WebSub extension defines the following requirements classes:

- Discovery Conformance Class (mandatory)

7.1. 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.

An implementation MUST support discovery via HTTP GET **and** HEAD methods.

An implementation MUST return the `<Link/> rel="hub"` HTTP header to indicate support for WebSub.

An implementation MUST return the `<Link/> rel="self"` HTTP header to indicate the ability for subscription.

An implementation MUST omit the `<Link/> rel="self"` HTTP header and instead return the `<Link/> rel="help"` HTTP header to indicate the cause why the request URL is not suitable for subscription.

NOTE 1: An implementation may use 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.

An implementation of the STA-WebSub Hub MUST transform the HTTP (discovery) URL into a MQTT topic by removing the SensorThings API `baseUrl`. For example, a deployment with the `baseUrl=http://localhost:8080/mysta`, the discovery URL `http://localhost:8080/mysta/v1.1/Observations` results in the MQTT topic `v1.1/Observations`.

An implementation of this conformance class MUST limit the discovery such that the associated MQTT topic is compliant to OGC 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] with the following extensions:

- `SERVICE_VERSION/RESOURCE_PATH/COLLECTION_NAME` as defined in 14.2.1 can be extended with a `?` followed by any valid combination of ODATA commands — e.g. `v1.1/Datastreams(1)/Observations?$filter=result gt 30`
- `SERVICE_VERSION/RESOURCE_PATH_TO_AN_ENTITY` as defined in 14.2.2 can be extended with a `?` followed by any valid combination of ODATA commands — e.g. `v1.1/Observations?$select=result`
- `SERVICE_VERSION/RESOURCE_PATH_TO_AN_ENTITY/PROPERTY_NAME` as defined in 14.2.3

NOTE 2: Compliance with this conformance class does not allow that the HTTP URL includes an ODATA query!

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

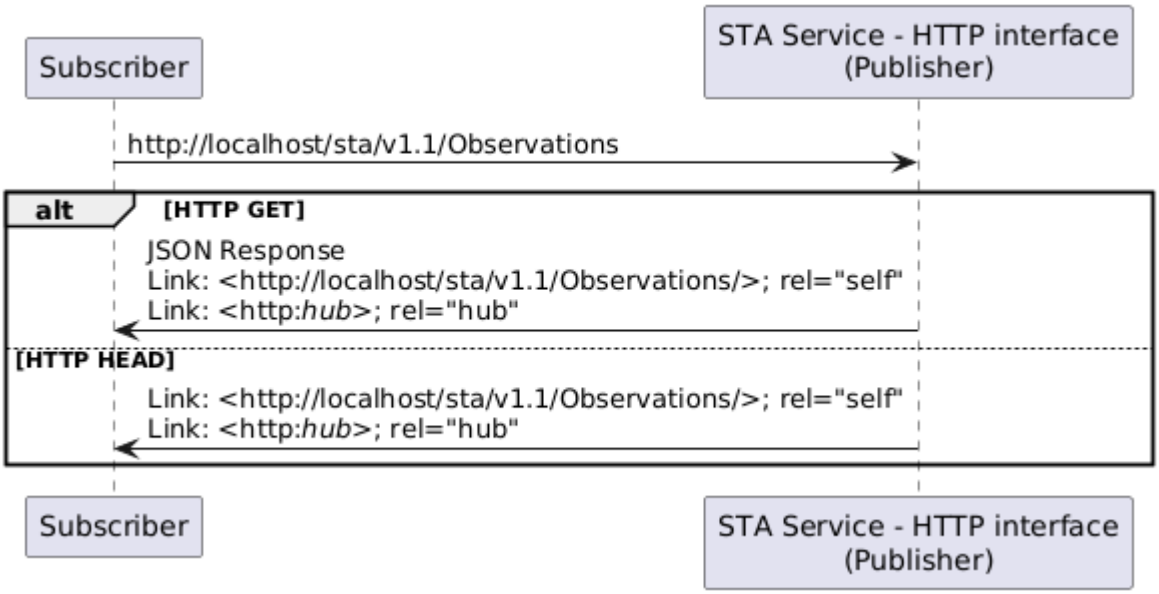


Figure 3 — Discovery with URL allowed for Subscription

The following sequence diagram illustrates the discovery for the non-compliant URL `http://localhost/sta/v1.1/Observations?$expand=FeatureOfInterest`

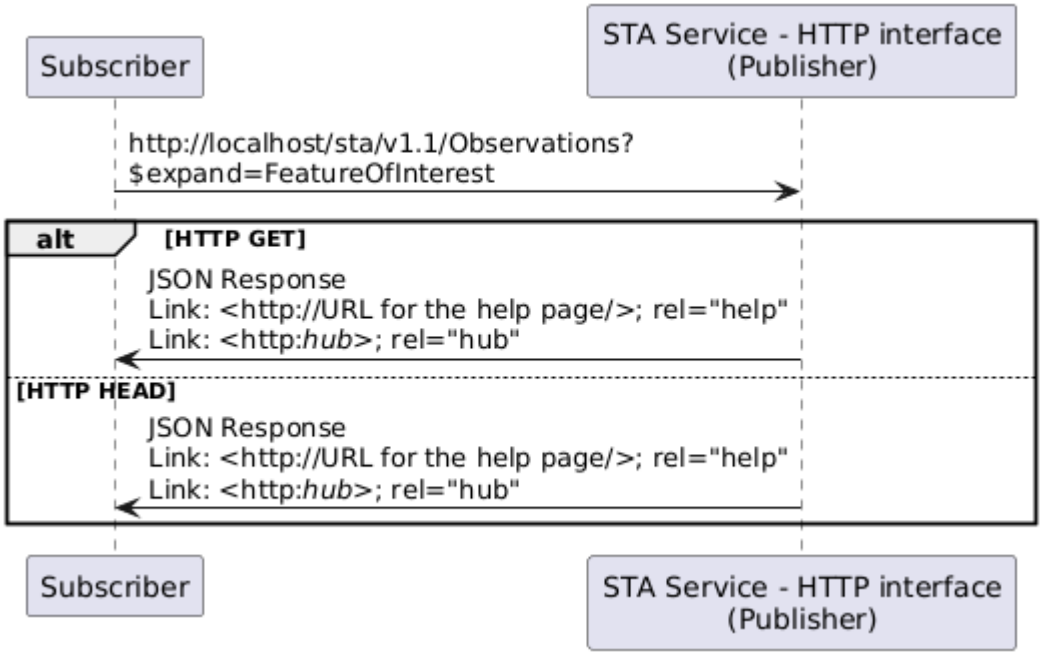


Figure 4 — Discovery with URL not allowed for Subscription

An implementation MUST advertize the accepted ODATA commands on the OGC Landing Page as a space separated string. For example, the following advertizement indicates support for `$select` and `$filter`:

```
"conformance": {
    ...
    "link:++http://www.opengis.net/spec/sensorthings-websub/1.0/conf/discovery++[]",
    "link:++http://www.opengis.net/spec/sensorthings-websub/1.0/conf/discovery/odata++[]"
},
"link:++http://www.opengis.net/spec/sensorthings-websub/1.0/conf/discovery/odata":++[] {
    "allowed": "$select $filter"
}
```

Listing 1

Expressing the limitations for the discovery on root topics or particular entity(ies) can become quite complex. An implementation MUST provide a description of the restrictions and advertize the link to that page on the Landing Page. The following snippet of the Landing Page is an example:

```
"conformance": {
    ...
    "link:++http://www.opengis.net/spec/sensorthings-websub/1.0/conf/discovery++[]",
    "link:++http://www.opengis.net/spec/sensorthings-websub/1.0/conf/discovery/policy++[]"
},
"link:++http://www.opengis.net/spec/sensorthings-websub/1.0/conf/discovery/policy":++[] {
    "href": "link:++http://localhost/myDiscoveryPolicy.html++[]"
```

}

Listing 2

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

7.1.1. 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)

7.1.2. 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.

8

MEDIA TYPES FOR ANY DATA ENCODING(S)

8 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) HUB REQUIREMENTS CLASS

B

ANNEX B
(INFORMATIVE)
HUB REQUIREMENTS CLASS

B.1. Subscription (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.

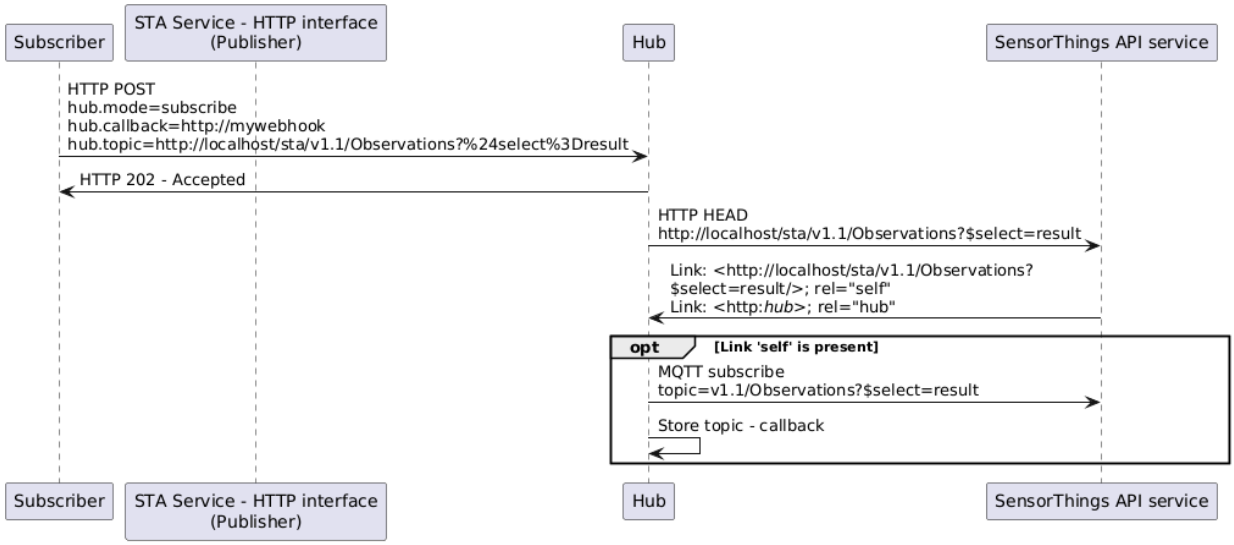


Figure B.1 — Subscription Handling in the STA-WebSub Hub

As defined in the W3C WebSub, 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. For a STA-WebSub Hub it is recommended to also check the subscription validity with the publisher using the discovery protocol. Upon success (the `Link`

`rel="self"` is included in both discovery responses), the Hub subscribes to the SensorThings API service via MQTT after transforming the topic URL into a MQTT topic.

It is recommended to use the HTTP HEAD method for the discovery requests to avoid unnecessary data transfer.

B.2. Notification (mandatory)

The STA-WebSub Hub supports receiving MQTT notifications from the MQTT broker associated with the SensorThings API service. An implementation MUST listen to notifications from the MQTT broker of the SensorThings API service. Once a notification is recevied, the implementation MUST distribute the notification content to all subscribers using HTTP(S) as defined by the W3C WebSub Recommendation.

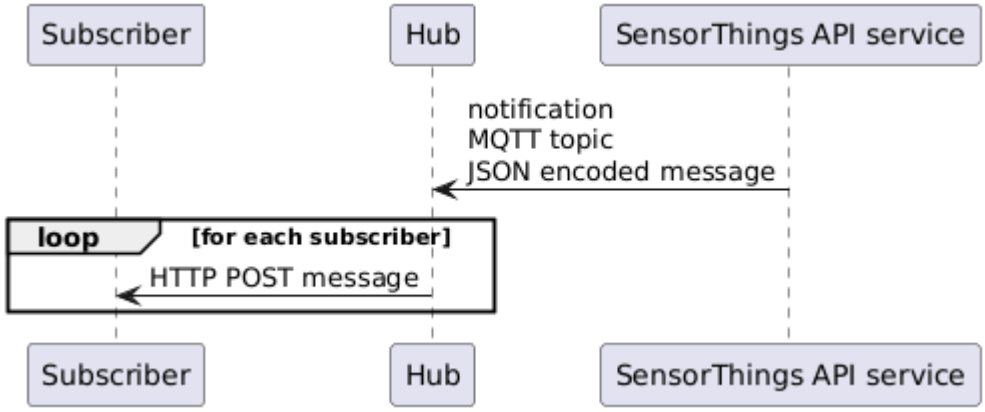


Figure B.2 — MQTT notification delivery by the STA-WebSub Hub

If the subscription was made with the `hub.api_key` or `hub.x_api_key`, the implementation MUST add the `Api-Key` or `X-Api-Key` to the HTTP request send to the subscriber's callback.

Likewise, the implementation MUST add the `X-Hub-Signature` header if the subscription was made with the `hub.secret` parameter.

B.3. Callback Authentication (mandatory)

This conformance class enables the STA-WebSub Hub supporting the use of the parameter `hub.api_key` or `hub.x_api_key` with a subscription request.

An implementation MUST accept the `hub.api_key` or `hub.x_api_key` with a subscription request. If both parameters (the `hub.api_key` **and** the `hub.x_api_key`) are used, the hub MUST deny the subscription and return a HTTP response status code 400.

- If the subscription request includes the `hub.api_key` parameter, the hub MUST add the HTTP header `Api-Key` to the request send to the subscriber's callback
- If the subscription request includes the `hub.x_api_key` parameter, the hub MUST add the HTTP header `X-Api-Key` to the request send to the subscriber's callback

For renewing a subscription (the hub receives a subscribe request for an existing subscription), the hub MUST update the api-key and use the new value for further message distribution to the subscriber.

The `hub.api_key` or `hub.x_api_key` parameter SHOULD only be specified when the request was made over HTTPS [[RFC2818]]. The parameter MUST be less than 200 bytes in length.

B.4. ODATA handling implemented in the Hub (optional)

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.

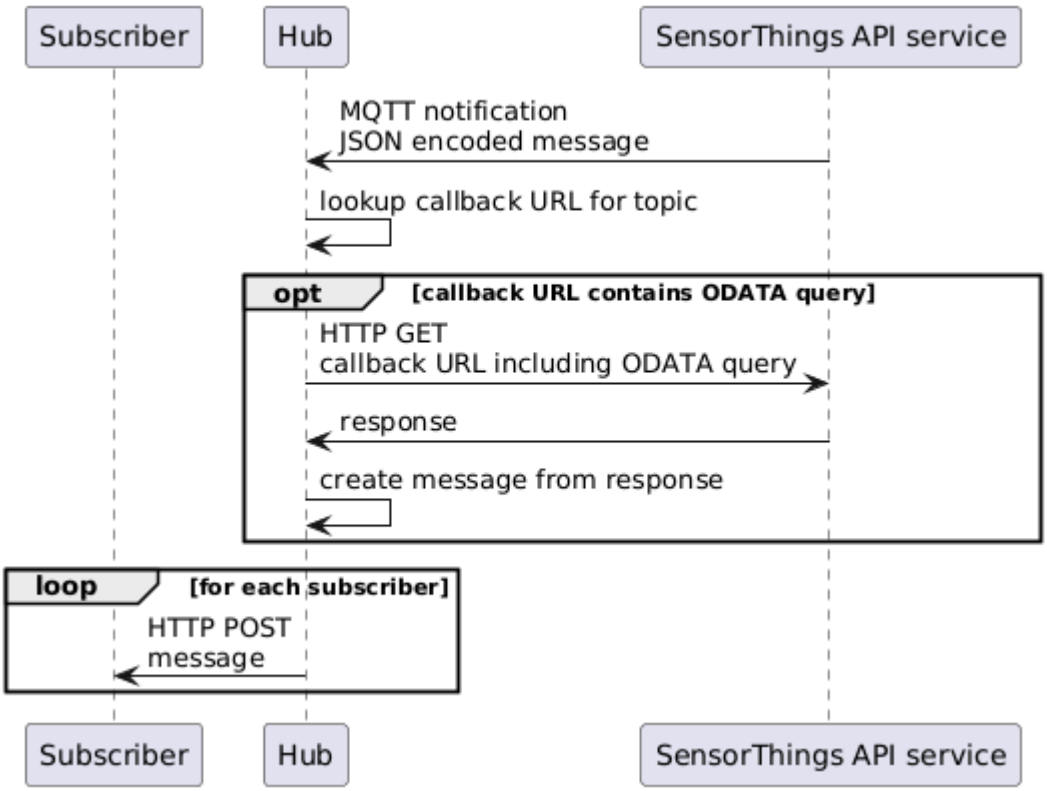


Figure B.3 — Hub receives MQTT notification from SensorThings API service

C

ANNEX C (INFORMATIVE) TITLE

C

ANNEX C (INFORMATIVE) TITLE

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

D

ANNEX D (INFORMATIVE) DISCUSSION OF THE IMPLEMENTATION OPTIONS FOR THE ODATA QUERY HANDLING.

ANNEX D (INFORMATIVE) 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.

E

ANNEX E (INFORMATIVE) REVISION HISTORY

ANNEX E (INFORMATIVE) REVISION HISTORY

Table E.1

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2024-10-08	0.1	Andreas Matheus	all	initial version
2024-12-11	0.2	Andreas Matheus	all	improvements based on implementations
2024-12-19	0.3	Andreas Matheus	all	restructuring to meet OGC standards structure

BIBLIOGRAPHY

BIBLIOGRAPHY