OGC® DOCUMENT: 22-022

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

OGC SENSORTHINGS API 1.1 EXTENSION: STAPLUS 1.0

STANDARD
Implementation

DRAFT

Version: 0.7
Submission Date: 2023-02-06
Approval Date: 2029-03-30
Publication Date: 2029-03-30
Editor: Andreas Matheus

Notice for Drafts: This document is not an OGC Standard. This document is distributed for review and comment. This document is subject to change without notice and may not be referred to as an OGC Standard.
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

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

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

LIST OF TABLES

LIST OF FIGURES

LIST OF RECOMMENDATIONS

I ABSTRACT

The OGC SensorThings API 1.1 Extension: STAplus 1.0 Standard specifies a backwards-compatible extension to the OGC SensorThings API Part 1: Sensing Version 1.1 (STA) Standard data model.

The motivation for specifying this STAplus extension is based on requirements from the Citizen Science community.

The dominant use for the OGC SensorThings API (STA) data model (and API) can be coined with the use case “single authority provides sensor readings to consumers”. However, in Citizen Science there are many contributors (citizens) that – together – create the big “picture” with their observations.

The STAplus extension is designed to support a model in which observations are owned by (different) users that may express the license for re-use. This part of the contribution is called the ownership concept. In addition to the ownership and license abilities, the extension supports expressing explicit relations between observations and creating group(s) of observations to containerize observations that belong together. Relations can be created among any individual observations or observations of a group to support performant Linked Data extraction and semantic queries, for example expressed in SPARQL.

The STAplus extension is believed to be an important contribution towards the realization of the FAIR principles as STAplus strengthens the “I” (Interoperability) through a common data model and API as well as the “R” (Re-usability) by allowing expressing standards-based queries that may consider licensing conditions which is relevant for reuse of other users’ observations.

The STAplus Data Model and Business Logic also enriches existing deployments as the extension can be seamlessly added and thereby offer new capabilities to create and manage the “big picture” with multi-user capabilities.

The key work for crafting this OGC Standard was undertaken in the Co-designed Citizen Observatories Services for the EOS-Cloud (Cos4Cloud) project, which received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement number 863463. Testing of this extension was done with data from the Framework biodiversity project, which received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement number 862731.

II KEYWORDS

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

ogcdoc, OGC document, API, SensorThings, STAplus

III PREFACE

STAplus — SensorThings API extension PLUS — defines a SensorThings data model extension to improve FAIR principles when exchanging sensor data including licensing and ownership information.

The STAplus extension is fully backwards compatible to the existing OGC OGC SensorThings API Part 1: Sensing Version 1.1 and thereby offers existing deployments to easily upgrade to STAplus. 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

A STAplus service implementation that supports the Create, Update or Delete of STAplus entities should also implement authentication and a “fit-for-purpose” business logic that enforces the required access conditions to ensure ownership of all entities, including SensorThings core entities.

The Business Logic makes the full power of STAplus useful! For example, sealed (closed) Groups — aka a set of observations that may include relations — could be used for interdisciplinary research by simply exchanging the group’s URL. But, it is essential to trust such a group which requires a verifiable group author and the business logic to support integrity of such a closed group.

As each business logic flow may be different from service to service deployment, the operator of the service should describe the business logic and make it available to developers that intend to use the STAplus deployment. The specification of a conformance class `Business Logic`, reflecting the semantics of the logic in a standardized fashion, is out of scope for this Standard but could be defined in an extension to STAplus.

Without an appropriate business logic, enforcing ownership and ensuring integrity in a multi-user scenario, it is possible that “junk” or “spam” is associated to a party without their knowledge. In such a multi-user CRUD access scenario, it would also be possible to “steal” an entire `Datastream` entity by simply updating the associated `Party` entity. Further, observations could be modified or even deleted without the creating user’s approval. While not specific to STAplus, these Business Logic considerations and concerns are identical for any deployed SensorThings API service.

The STAplus data model does not support to record provenance. If the business logic allows the update or even deletion of entities, there is no history of who did what and when. The data served from a service implementation is a snapshot in time which may cause pagination to produce unexpected results.

For implementations of the `Create`, `Update` and `Delete` conformance class it is paramount to check all user uploads for malicious code. Typical SQL injection checks are mandatory but also JavaScript code injection must be tackled. For example, it is likely that the value of an observation will be displayed on some HTML page. To prevent malicious code or virus injection, similar to cross-site-scripting attacks, pattern checking on any uploaded data should be accomplished.

V SUBMITTING ORGANIZATIONS

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

- Secure Dimensions
- CREAF
- Fraunhofer Gesellschaft

1

SCOPE

1 SCOPE

OGC STAplus provides an open standard-based and backwards-compatible data model extension to SensorThings API v1.1 by introducing ownership and licensing. Additional features such as grouping and relations can be used for enriching the re-use and linking of sensor data with semantical concepts to improve the FAIR principles.

2

CONFORMANCE

2 CONFORMANCE

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

2.1. Introduction

The STAplus 1.0 standard defines an extension to the OGC SensorThings API Part 1: Sensing Version 1.1 Standard by adding additional entities via the Data Model. Access to the STAplus entities via HTTP is defined in the Read, Update and Delete Requirements Classes. The use of MQTT for STAplus entities is defined in the MQTT[1] Requirements Class. A default storage-CRS is defined via the Default-CRS Requirements Class and additional geometry encodings are defined in the Geometry FG and Geometry WKT Requirements Class. The Authentication Requirements Class supports the unique identification of acting users. The Business Requirements Class supports the textual description of the implemented business logic.

2.2. STAplus 1.0 Conformance Classes

This OGC Standard defines one mandatory and several optional conformance classes.

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:

- The one mandatory conformance class.
- Any one of the optional conformance classes.
- Pass all applicable tests as defined in Annex A (normative)

In order to conform to this OGC® Standard, a software implementation SHALL implement the mandatory conformance class specified:

[1] MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT).

- **Core** (mandatory): This Conformance Class incorporates the capabilities to support the Read of STAplus entities via HTTP. It also includes the Storage-CRS Requirements Class to ensure interoperability with geometry encodings in the `feature` and `location` properties.
- **Create** (optional): This Conformance Class supports to create STAplus entities via HTTP.
- **Update** (optional): This Conformance Class supports to update STAplus entities via HTTP.
- **Delete** (optional): This Conformance Class supports to delete STAplus entities via HTTP.
- **Authentication** (optional): This Conformance Class supports user authentication and their unique identification.
- **Geometry-FG** (optional): This Conformance Class supports the alternative geometry encoding for the `feature` and `location` properties based on Features and their Geometries.
- **Geometry-WKT** (optional): This Conformance Class supports the alternative geometry encoding for the `feature` and `location` properties based on Well Known Text (WKT) as defined in Geographic information - Simple features access - Part 1: Common architecture.
- **MQTT Subscribe** (optional): This Conformance Class supports a client to receive STAplus entity changes via MQTT.
- **Business Logic** (optional): This Conformance Class supports that the implementation business logic is described in English text to help developers understand the details of CRUD access.

2.2.1. Conformance Class Core

The `Core` Conformance Class is defined as follows:

CONFORMANCE CLASS 1: CORE	
IDENTIFIER	`/conf/core`
REQUIREMENTS CLASS	Requirements class 1: `/req-class/entity-control-information` Requirements class 2: `/req-class/party` Requirements class 3: `/req-class/license` Requirements class 4: `/req-class/group` Requirements class 5: `/req-class/relation` Requirements class 6: `/req-class/project` Requirements class 7: `/req-class/read` Requirements class 14: `/req-class/storage-crs`
TARGET TYPE	Implementation
CONFORMANCE TESTS	Conformance test A.1: `/conf/core/common-control-information` Conformance test A.2: `/conf/core/entities`

CONFORMANCE CLASS 1: CORE	
	Conformance test A.3: `/conf/core/read` Conformance test A.4: `/conf/core/storage-crs/crs-definition` Conformance test A.5: `/conf/core/storage-crs/axis-order` Conformance test A.6: `/conf/core/storage-crs/media-type` Conformance test A.7: `/conf/core/storage-crs/processing`

2.2.2. Conformance Class Create

The `Create` Conformance Class is defined as follows:

CONFORMANCE CLASS 2: UPDATE	
IDENTIFIER	`/conf/create`
REQUIREMENTS CLASS	Requirements class 8: `/req-class/create`
PREREQUISITE	Conformance class 1: `/conf/core`
TARGET TYPE	Implementation
CONFORMANCE TEST	Conformance test A.8: `/conf/create/http`

2.2.3. Conformance Class Update

The `Update` Conformance Class is defined as follows:

CONFORMANCE CLASS 3: UPDATE	
IDENTIFIER	`/conf/update`
REQUIREMENTS CLASS	Requirements class 9: `/req-class/update`
PREREQUISITE	Conformance class 1: `/conf/core`
TARGET TYPE	Implementation
CONFORMANCE TESTS	Conformance test A.9: `/conf/update/put` Conformance test A.10: `/conf/update/patch`

2.2.4. Conformance Class Delete

The `Delete` Conformance Class is defined as follows:

CONFORMANCE CLASS 4: DELETE	
IDENTIFIER	`/conf/delete`
REQUIREMENTS CLASS	Requirements class 10: `/req-class/delete`
PREREQUISITE	Conformance class 1: `/conf/core`
TARGET TYPE	Implementation
CONFORMANCE TEST	Conformance test A.11: `/conf/delete/entity`

2.2.5. Conformance Class Authentication

The `Authentication` Conformance Class is defined as follows:

CONFORMANCE CLASS 5: AUTHENTICATION	
IDENTIFIER	`/conf/authentication`
REQUIREMENTS CLASS	Requirements class 13: `/req-class/authentication`
PREREQUISITE	Conformance class 1: `/conf/core`
TARGET TYPE	Implementation
CONFORMANCE TESTS	Conformance test A.12: `/conf/authentication/id` Conformance test A.13: `/conf/authentication/anon-personal-data-crud` Conformance test A.14: `/conf/authentication/own-personal-data-crud` Conformance test A.15: `/conf/authentication/other-personal-data-crud`

2.2.6. Conformance Class Geometry FG

The `Geometry FG` Conformance Class is defined as follows:

CONFORMANCE CLASS 6: GEOMETRY FG	
IDENTIFIER	`/conf/geometry-fg`
REQUIREMENTS CLASS	Requirements class 15: `/req-class/geometry-fg`
PREREQUISITE	Conformance class 1: `/conf/core`
TARGET TYPE	Implementation
CONFORMANCE TESTS	Conformance test A.18: `/conf/geometry-fg//media-type` Conformance test A.19: `/conf/geometry-fg/default-crs` Conformance test A.20: `/conf/geometry-fg/supported-crs` Conformance test A.21: `/conf/geometry-fg/crs-error` Conformance test A.22: `/conf/geometry-fg/processing` Conformance test A.23: `/conf/geometry-fg/out`

2.2.7. Conformance Class Geometry WKT

The `Geometry WKT` Conformance Class is defined as follows:

CONFORMANCE CLASS 7: GEOEMTRY WKT	
IDENTIFIER	`/conf/geometry-wkt`
REQUIREMENTS CLASS	Requirements class 16: `/req-class/geometry-wkt`
PREREQUISITE	Conformance class 1: `/conf/core`
TARGET TYPE	Implementation
CONFORMANCE TESTS	Conformance test A.24: `/conf/geometry-wkt/media-type` Conformance test A.25: `/conf/geometry-wkt/crs-definition` Conformance test A.26: `/conf/geometry-wkt/default-crs` Conformance test A.27: `/conf/geometry-wkt/supported-crs` Conformance test A.28: `/conf/geometry-wkt/crs-error` Conformance test A.29: `/conf/geometry-wkt/value` Conformance test A.30: `/conf/geometry-wkt/processing` Conformance test A.31: `/conf/geometry-wkt/out`

2.2.8. Conformance Class MQTT Subscribe

The `MQTT Subscribe` Conformance Class is defined as follows:

CONFORMANCE CLASS 8: MQTT SUBSCRIBE	
IDENTIFIER	`/conf/mqtt-subscribe`
REQUIREMENTS CLASS	Requirements class 11: `/req-class/mqtt-subscribe`
PREREQUISITE	Conformance class 1: `/conf/core`
TARGET TYPE	Implementation
CONFORMANCE TEST	Conformance test A.32: `/conf/mqtt-subscribe/definition`

2.2.9. Conformance Class Business Logic

The `Business Logic` Conformance Class is defined as follows:

CONFORMANCE CLASS 9: BUSINESS LOGIC	
IDENTIFIER	`/conf/business-logic`
REQUIREMENTS CLASS	Requirements class 12: `/req-class/business-logic`
PREREQUISITE	Conformance class 1: `/conf/core`
TARGET TYPE	Implementation
CONFORMANCE TESTS	Conformance test A.16: `/conf/business-logic/definition` Conformance test A.17: `/conf/business-logic/location`

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.

OGC SensorThings API Part 1: Sensing Version 1.1 (2020)

The GeoJSON Format, IETF (2016)

Geographic information — Simple features access — Part 1: Common architecture, ISO, 2004, https://portal.opengeospatial.org/files/?artifact_id=25355

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. Party

The entity in the data model that represents a user. A party can be associated to other entities of the data model to express ownership.

4.2. License

The entity in the data model that represents a well-defined license. Associating a license with a `Multi/Datastream` entity expresses re-use conditions for all observations associated with that `Multi/Datastream` entity.

4.3. Group

The entity in the data model that allows creation a collection of observations for a particular purpose.

4.4. Relation

The entity in the data model that allows expressing relationships between two observations or an observation and an external object.

4.5. Project

The entity in the data model that allows merging `Multi/Datastream` entities together that are required to achieve the objective represented by the project.

4.6. Ownership

The `Party` entity in the data model allows expressing the right of possession over entity instances. This is the requirement to operate STAplus with multi-user CRUD access.

5 CONVENTIONS

5 CONVENTIONS

This Clause 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/doc/is/sensorthings/1.1/staplus/1.0`

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

6

INTRODUCTION

6 INTRODUCTION

The STAplus 1.0 Standard defines an extension to the OGC SensorThings API named STAplus. This extension originally started with motivations and requirements from the Citizen Science community. However, the STAplus extension has wider applicability than just Citizen Science.

STAplus is a 100% backwards compatible extension to the STA V1.1 data model and as such can be added to existing STA deployments.

In addition to the data model extension, this Standard defines different concepts that support operating an implementation in a multi-user CRUD deployment.

6.1. Concept of Ownership

In Citizen Science, users participate in projects or campaigns, offered by different Citizen Science portals. These portals, typically operated by different entities, have one feature in common: Contributions, uploaded by users are associated with the user and the ownership does not change. There is an explicit relationship between the observation (contribution) and the user. Expressed as ownership, users can undertake certain actions on their resources.

With the SensorThings API (STA), observations are linked to a data stream that is linked to a sensor which belongs to a thing.The data model does not provide a class that supports explicitly linking a user (party) to an observation. This limits the use of the STA data model (and API) to a simple use case: One operator can create data objects and all other users have read-only access. This limits the options for applications to interact with the API.

Even though the STA v1.1 data model offers the generic use of “properties”, expressing ownership (user association) buried in properties is not wise. This also makes compliance with the European General Data Protection Regulation (GDPR) difficult for applications using STA. This is because it is unclear if properties store personal data and therefore fall under GDPR. This decreases interoperability tremendously as an application developer would need to know which attribute expresses ownership. Also, querying observations based on unstructured properties is extremely complex and difficult.

Therefore, the STAplus extension defines the class Party for expressing the association from a (Multi)Datastream to a user. All observations, generated by a Datastream instance belong to the associated party.

6.2. Improving F A I R

In general, there are many aspects when it comes to ensure reusability of (existing) data. Not only in the context of Citizen Science, one fundamental aspect is the licensing aspect. Very many users contribute to Citizen Science with the motivation to do meaningful things for the common good. At the same time, users like to be credited when it comes to re-use of their contribution(s). Therefore, most contributions in Citizen Science are freely accessible (open access) but the re-use is not simply "open". In order to get credited, users might associate a license like CC-BY (Creative Commons Attribution License). Even though the data is still freely accessible, there is a condition that must be followed as expressed in the license.

6.3. Creating Observation Bags

When contributing to Citizen Science, the actual observation is often a set or bag of individual observations that belong together — in other words belong to the same observation event. For example, a camera trap event consists of a picture, a textual observation expanding the likelihood of species prediction and sensor readings for environmental context (temperature, humidity, luminance, air pressure, GPS location). All of these (individual) observations were created at the same time/location and could therefore be grouped.

Another use case for applying grouping to existing observations is to create a package of observations for the purpose of building the fundamentals for research or to be used in workflows. For researching and later evaluation of a particular phenomenon, the same bag (or set) of observations can be exchanged via the grouping concept.

Also, over time a user community might link other observations and even provide cross links to other databases such as the Global Biodiversity Information Facility (GBIF)[2]. These can be semantically tagged with the `Relation` entity which is described later in this document.

The STAplus extension defines a flexible grouping concept by adding the `Group` entity to the STA data model.

6.4. Expressing Relations

For Citizen Science expressing relations between observations explicitly to support search based on these relations is important. The SensorThings data model does not support to express relationships.

Therefore, the STAplus extension introduces the `Relation` entity that supports creating generic "from – to" relations. The "to" can point to another observation or to an external object. This allows the generic expression of meaning, leveraging existing semantic concepts that exist elsewhere, for example Dublin or Darwin Core. This helps to reason how observations are related even if they were not observed in a same observation event, but instead were linked later to a particular community process (as linking to other databases).

[2] https://www.gbif.org/

The use of Relations as defined in STAplus can be applied to already existing SensorThings deployments to enrich the data towards semantics.

6.5. Data Model Extension

The STAplus data model extension allows expressing the following additional characteristics:

- People: The `Party` entity supports linking a user to a `Datastream` or `Group`
- License: The `License` entity supports expressing reuse conditions by linking a `License` to a `Datastream` and / or to a `Group`. A `License` on a `Datastream` has the result that all `Observation` entities of that `Datastream` (as well as entities of the `Thing`, `Sensor` and `ObservedProperty`) have the associated license. A `License` on a `Group` gives the bag or set of `Observations` (represented by the `Group`) a license for reuse. Note that still the license for each `Observation` must still be followed.
- Union: The `Group` entity supports packaging individual `Observations` as a bag or set either as deep copy or via linking
- Semantics: The `Relation` entity supports expressing relationships between `Observation` entities using the “from-to” type. It is also possible to create relations between `Observation` entities and external objects using the URI scheme. This allows in particular to express a relation to any entity of the database (via their external URI). `Relation` entities can exist on its own or be included into a Group to enrich a bag or set of `Observation` entities.
- Project: The `Project` entity is a container of `Datastream` and `MultiDatastream` entities that supports organizing a campaign or project and to provides metadata as well as legal information such as terms of use and a privacy statement, in case it is relevant.

7

STAPLUS ENTITY TYPE REQUIREMENTS

7 STAPLUS ENTITY TYPE REQUIREMENTS

`Party`, `License`, `Group`, `Relation` and `Project` are the STAplus entity types. The implemented entities SHALL be listed in the response to a GET request to the root URL as described in Sensing part.

An implementation SHALL implement all STAplus entities (with read access) as defined in the Core conformance class.

The STAplus entities are depicted in Figure 1 and Figure 2.

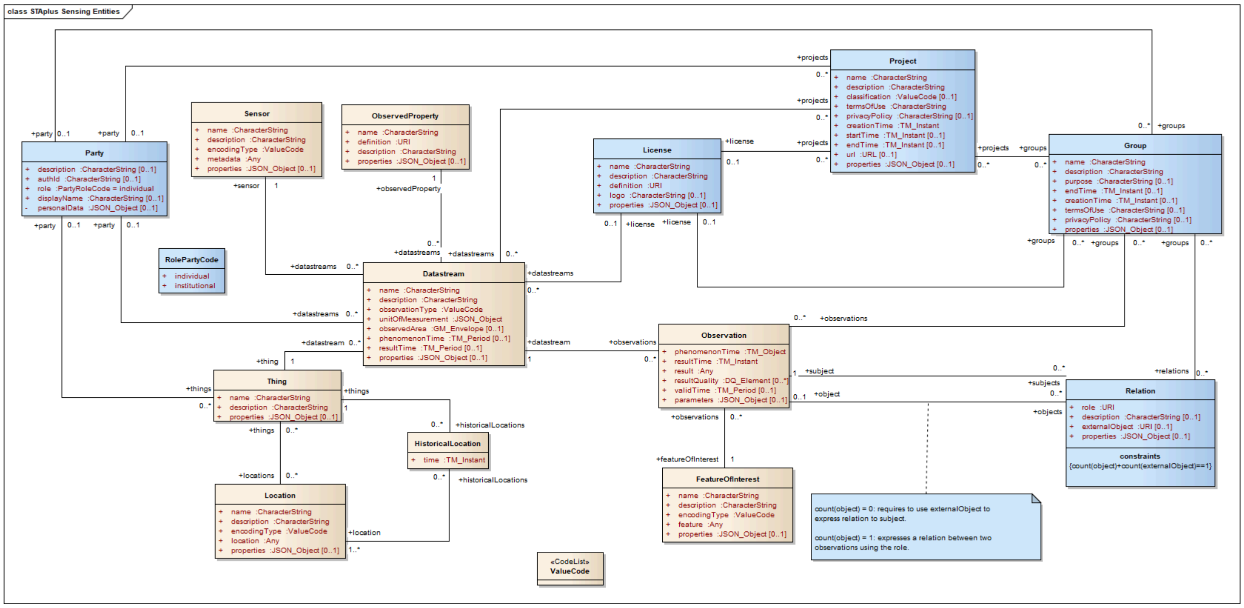
class STAplus Sensing Entities
Party
+ description :CharacterString [0..1]
+ authId :CharacterString [0..1]
+ role :PartyRoleCode = individual
+ displayName :CharacterString [0..1]
- personalData :JSON_Object [0..1]
Sensor
+ name :CharacterString
+ description :CharacterString
+ encodingType :ValueCode
+ metadata :Any
+ properties :JSON_Object [0..1]
ObservedProperty
+ name :CharacterString
+ definition :URI
+ description :CharacterString
+ properties :JSON_Object [0..1]
License
+ name :CharacterString
+ description :CharacterString
+ definition :URI
+ logo :CharacterString [0..1]
+ properties :JSON_Object [0..1]
Project
+ name :CharacterString
+ description :CharacterString
+ classification :ValueCode [0..1]
+ termsOfUse :CharacterString
+ privacyPolicy :CharacterString [0..1]
+ creationTime :TM_Instant
+ startTime :TM_Instant [0..1]
+ endTime :TM_Instant [0..1]
+ url :URL [0..1]
+ properties :JSON_Object [0..1]
Group
+ name :CharacterString
+ description :CharacterString
+ purpose :CharacterString [0..1]
+ endTime :TM_Instant [0..1]
+ creationTime :TM_Instant [0..1]
+ termsOfUse :CharacterString [0..1]
+ privacyPolicy :CharacterString [0..1]
+ properties :JSON_Object [0..1]
RolePartyCode
+ individual
+ institutional
Datastream
+ name :CharacterString
+ description :CharacterString
+ observationType :ValueCode
+ unitOfMeasurement :JSON_Object
+ observedArea :GM_Envelope [0..1]
+ phenomenonTime :TM_Period [0..1]
+ resultTime :TM_Period [0..1]
+ properties :JSON_Object [0..1]
Observation
+ phenomenonTime :TM_Object
+ resultTime :TM_Instant
+ result :Any
+ resultQuality :DQ_Element [0..*]
+ validTime :TM_Period [0..1]
+ parameters :JSON_Object [0..1]
Relation
+ role :URI
+ description :CharacterString [0..1]
+ externalObject :URI [0..1]
+ properties :JSON_Object [0..1]
constraints
{count(object)+count(externalObject)==1}
Thing
+ name :CharacterString
+ description :CharacterString
+ properties :JSON_Object [0..1]
HistoricalLocation
+ time :TM_Instant
Location
+ name :CharacterString
+ description :CharacterString
+ encodingType :ValueCode
+ location :Any
+ properties :JSON_Object [0..1]
FeatureOfInterest
+ name :CharacterString
+ description :CharacterString
+ encodingType :ValueCode
+ feature :Any
+ properties :JSON_Object [0..1]
«CodeList»
ValueCode
count(object) = 0: requires to use externalObject to express relation to subject.
count(object) = 1: expresses a relation between two observations using the role.
+party 0..1
+projects 0..*
+license 0..1
+groups 0..*
+sensor 1
+observedProperty 1
+datastreams 0..*
+datastream 1
+observations 0..*
+thing 1
+things 0..*
+locations 0..*
+historicalLocations 0..*
+location 1..*
+featureOfInterest 1
+subject 1
+subjects 0..*
+object 0..1
+objects 0..*
+relations 0..*

Figure 1 — STAplus Entities (Datastream)

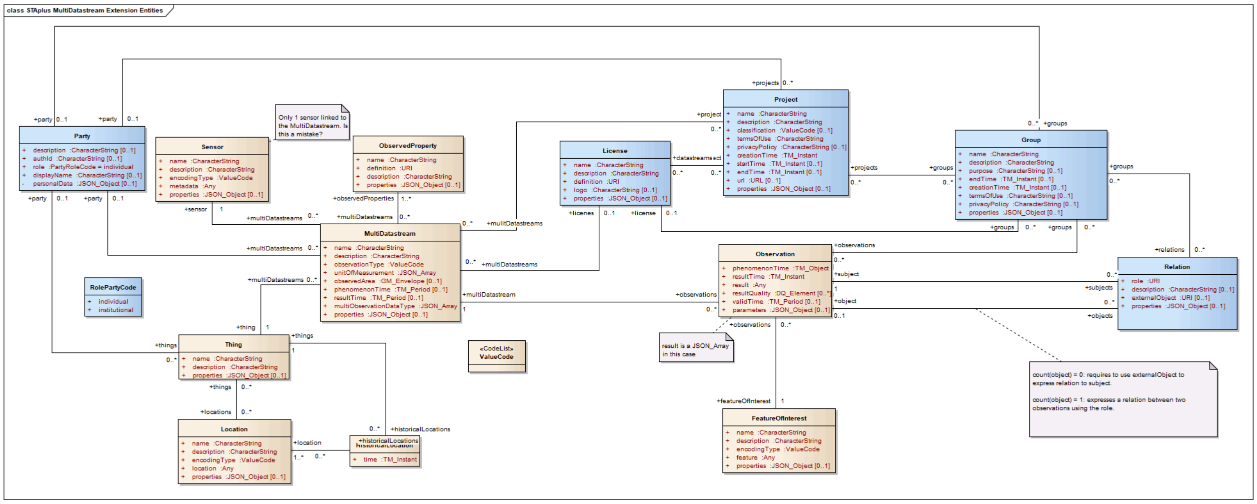
class STAplus MultiDatastream Extension Entities
Party
+ description :CharacterString [0..1]
+ authId :CharacterString [0..1]
+ role :PartyRoleCode = individual
+ displayName :CharacterString [0..1]
- personalData :JSON_Object [0..1]
Sensor
+ name :CharacterString
+ description :CharacterString
+ encodingType :ValueCode
+ metadata :Any
+ properties :JSON_Object [0..1]
Only 1 sensor linked to the MultiDatastream. Is this a mistake?
ObservedProperty
+ name :CharacterString
+ definition :URI
+ description :CharacterString
+ properties :JSON_Object [0..1]
License
+ name :CharacterString
+ description :CharacterString
+ definition :URI
+ logo :CharacterString [0..1]
+ properties :JSON_Object [0..1]
Project
+ name :CharacterString
+ description :CharacterString
+ classification :ValueCode [0..1]
+ termsOfUse :CharacterString
+ privacyPolicy :CharacterString [0..1]
+ creationTime :TM_Instant
+ startTime :TM_Instant [0..1]
+ endTime :TM_Instant [0..1]
+ url :URL [0..1]
+ properties :JSON_Object [0..1]
Group
+ name :CharacterString
+ description :CharacterString
+ purpose :CharacterString [0..1]
+ endTime :TM_Instant [0..1]
+ creationTime :TM_Instant [0..1]
+ termsOfUse :CharacterString [0..1]
+ privacyPolicy :CharacterString [0..1]
+ properties :JSON_Object [0..1]
RolePartyCode
+ individual
+ institutional
MultiDatastream
+ name :CharacterString
+ description :CharacterString
+ observationType :ValueCode
+ unitOfMeasurement :JSON_Array
+ observedArea :GM_Envelope [0..1]
+ phenomenonTime :TM_Period [0..1]
+ resultTime :TM_Period [0..1]
+ multiObservationDataType :JSON_Array
+ properties :JSON_Object [0..1]
Observation
+ phenomenonTime :TM_Object
+ resultTime :TM_Instant
+ result :Any
+ resultQuality :DQ_Element [0..*]
+ validTime :TM_Period [0..1]
+ parameters :JSON_Object [0..1]
result is a JSON_Array in this case
Relation
+ role :URI
+ description :CharacterString [0..1]
+ externalObject :URI [0..1]
+ properties :JSON_Object [0..1]
Thing
+ name :CharacterString
+ description :CharacterString
+ properties :JSON_Object [0..1]
«CodeList»
ValueCode
Location
+ name :CharacterString
+ description :CharacterString
+ encodingType :ValueCode
+ location :Any
+ properties :JSON_Object [0..1]
HistoricalLocation
+ time :TM_Instant
FeatureOfInterest
+ name :CharacterString
+ description :CharacterString
+ encodingType :ValueCode
+ feature :Any
+ properties :JSON_Object [0..1]
count(object) = 0: requires to use externalObject to express relation to subject.
count(object) = 1: expresses a relation between two observations using the role.
+party 0..1
+projects 0..*
+project 0..*
+datastreamsect
+licenes 0..1
+license 0..1
+groups 0..*
+sensor 1
+observedProperties 1..*
+multiDatastreams 0..*
+mulitDatastreams 0..*
+multiDatastream 1
+observations 0..*
+thing 1
+things 0..*
+locations 0..*
+historicalLocations 0..*
+location 1..*
+featureOfInterest 1
+subject 1
+subjects 0..*
+object 0..1
+objects 0..*
+relations 0..*

Figure 2 — STAplus Entities (MultiDatastream)

In this section, the properties for each entity type and the direct relation to the other entity types are explained.

7.1. Requirements Class Entity Control Information

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

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

When annotating a name/value pair for which the value is represented as a JSON array or primitive value, each annotation that applies to this name/value pair is placed next to the annotated name/value pair and represented as a single name/value pair. The name is the same as the name of the name/value pair being annotated, followed by the “at” sign (@), followed by the namespace iot, followed by a dot (.), followed by the name of the term. (e.g., `"Locations@iot.navigationLink":"http://example.org/v1.1/Things(1)/Locations"`)

REQUIREMENTS CLASS 1: ENTITY CONTROL INFORMATION	
IDENTIFIER	`/req-class/entity-control-information`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
CONFORMANCE CLASS	Conformance class 1: `/conf/core`
PREREQUISITE	OGC SensorThings API Part 1: Sensing Version 1.1
NORMATIVE STATEMENT	Requirement 1: `/req/common-control-information`

REQUIREMENT 1	
IDENTIFIER	`/req/common-control-information`
INCLUDED IN	Requirements class 1: `/req-class/entity-control-information`
STATEMENT	Each entity SHALL have the following common control information listed in Table 1.

Table 1 — Common control information

NAME	DEFINITION	DATA TYPE	MULTIPLICITY AND USE
id	`id` is the system-generated identifier of an entity that is unique among the entities of the same entity type in a SensorThings API service instance.	Any	One (mandatory)
selfLink	`selfLink` is the absolute URL of an entity that is unique among all other entities.	URL	One (mandatory)
navigationLink	`navigationLink` is the relative or absolute URL that retrieves content of related entities.	URL	One-to-many (mandatory)

7.2. Requirements Class Party

The `Party` entity can be used to represent acting users and to model ownership. One example for ownership is that a satellite `Thing` is owned by a space agency. This ownership may entitle the space agency to be the only party that can update the thing's location. Other parties can then mount their sensor on the satellite and provide `Datastream` or `MultiDatastream` instances to upload observations. Via the association to the `Datastream` resp. `MultiDatastream` their ownership of the observations is guaranteed. By associating a license to the (multi)datastream, they could also define re-use conditions.

REQUIREMENTS CLASS 2: PARTY	
IDENTIFIER	`/req-class/party`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
PREREQUISITE	Requirements class 1: `/req-class/entity-control-information`
NORMATIVE STATEMENTS	Requirement 2: `/req/party/properties` Requirement 3: `/req/party/relations`

REQUIREMENT 2	
IDENTIFIER	`/req/party/properties`

REQUIREMENT 2	
INCLUDED IN	Requirements class 2: /req-class/party
STATEMENT	Each Party entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 2.

Table 2 — Properties of a Party entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY AND USE
role	This is the role of the party	CharacterString ['individual', 'institutional']	One (mandatory)
description	This is a short description of the party	CharacterString	One (optional)
displayName	A property commonly used for saluting the party	CharacterString	One (optional)
authId	A system wide unique property (e.g. REMOTE_USER or SUB from authentication) to identify the party	CharacterString	One (optional)
personalData	A property to store personal data of the party	JSON Object	One (optional)

REQUIREMENT 3	
IDENTIFIER	/req/party/relations
INCLUDED IN	Requirements class 2: /req-class/party
STATEMENT	Each Party entity MAY have direct relations to other entity types listed in Table 3.

NOTEThe personalData property has private visibility. An implementation must ensure GDPR compliance when allowing CRUD access.

Table 3 — Direct relation between a Party entity and other entity types

ENTITY TYPE	RELATION	DESCRIPTION
Datastream	One optional to many optional	A Party MAY have zero-to-many Datastreams.
MultiDatastream	One optional to many optional	A Party MAY have zero-to-many MultiDatastreams.

ENTITY TYPE	RELATION	DESCRIPTION
Thing	One optional to many optional	A `Party` MAY have zero-to-many `Things`.
Group	One optional to many optional	A `Party` MAY have zero-to-many `Groups`.
Project	One optional to many optional	A `Party` MAY have zero-to-many `Projects`.

7.3. Requirements Class License

The `License` entity can be used to associate a re-use condition to observations via a `Datastream` or `MultiDatastream`. It can also be used to express re-use conditions for a group (a set of observations).

REQUIREMENTS CLASS 3: LICENSE	
IDENTIFIER	`/req-class/license`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
PREREQUISITE	Requirements class 1: `/req-class/entity-control-information`
NORMATIVE STATEMENTS	Requirement 4: `/req/license/properties` Requirement 5: `/req/license/relations`

REQUIREMENT 4	
IDENTIFIER	`/req/license/properties`
INCLUDED IN	Requirements class 3: `/req-class/license`
STATEMENT	Each `License` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 4.

Table 4 — Properties of a License entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY AND USE
name	A property provides a label for License entity, commonly a descriptive name.	CharacterString	One (mandatory)
description	This is a short description of the corresponding License entity.	CharacterString	One (mandatory)
definition	This is a URI referencing the License entity.	URI	One (mandatory)
logo	This is the data URI encoding of the logo for the License entity.	CharacterString	One (optional)
properties	The SensorThings API definition applies	JSON Object	One (optional)

REQUIREMENT 5	
IDENTIFIER	`/req/license/relations`
INCLUDED IN	Requirements class 3: `/req-class/license`
STATEMENT	Each `License` entity MAY have direct relations to other entity types listed in Table 5.

Table 5 — Direct relation between a License entity and other entity types

ENTITY TYPE	RELATION	DESCRIPTION
Datastream	One optional to many optional	A `License` MAY have zero-to-many `Datastreams`.
MultiDatastream	One optional to many optional	A `License` MAY have zero-to-many `MultiDatastreams`.
Project	One optional to many optional	A `License` MAY have zero-to-many `Projects`.
Group	One optional to many optional	A `License` MAY have zero-to-many `Groups`.

7.4. Requirements Class Group

The Group entity can be used to create a bag of observations and/or relations that can be shared and re-used.

REQUIREMENTS CLASS 4: GROUP	
IDENTIFIER	/req-class/group
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
PREREQUISITE	Requirements class 1: /req-class/entity-control-information
NORMATIVE STATEMENTS	Requirement 6: /req/group/properties Requirement 7: /req/group/relations

REQUIREMENT 6	
IDENTIFIER	/req/group/properties
INCLUDED IN	Requirements class 4: /req-class/group
STATEMENT	Each Group entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 6.

Table 6 — Properties of a Group entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY AND USE
name	A property provides a label for Group entity, commonly a descriptive name.	CharacterString	One (mandatory)
description	This is a short description of the corresponding Group entity.	CharacterString	One (mandatory)
purpose	This is a short description of the purpose for the Group entity.	CharacterString	One (optional)
creationTime	This is the starting time of the Group entity. Depending on the business logic, after this time it	TM Instant	One (optional)

NAME	DEFINITION	DATA TYPE	MULTIPLICITY AND USE
	could be possible to add observations or relations to the Group.		
endTime	This is the end time of the `Group` entity. Depending on the business logic, after this time it is no longer possible to add observations or relations to the Group.	TM Instant	One (optional)
termsOfUse	Express the term of use for the `Group` entity.	CharacterString	One (optional)
privacyPolicy	Express the term of use for personal data that are contained in the `Group` entity.	CharacterString	One (optional)
properties	The SensorThings API definition applies	JSON Object	One (optional)

REQUIREMENT 7	
IDENTIFIER	`/req/group/relations`
INCLUDED IN	Requirements class 4: `/req-class/group`
STATEMENT	Each `Group` entity MAY have direct relations to other entity types listed in Table 7.

Table 7 — Direct relation between a `Group` entity and other entity types

ENTITY TYPE	RELATION	DESCRIPTION
License	One optional to one optional	A `Group` MAY have zero-to-one `License`.
Observation	Many optional to many optional	A `Group` MAY have zero-to-many `Observations`.
Relation	Many optional to many optional	A `Group` MAY have zero-to-many `Relations`.
Party	Many optional to one optional	A `Group` MAY have zero-to-one `Party`.
Project	Many optional to many optional	A `Group` MAY have zero-to-more `Project`.

7.5. Requirements Class Relation

The `Relation` entity can be used to describe relationships between (1) two observations, or (2) one observation and a resolvable external object identified by a URI.

REQUIREMENTS CLASS 5: GROUP	
IDENTIFIER	`/req-class/relation`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
PREREQUISITE	Requirements class 1: `/req-class/entity-control-information`
NORMATIVE STATEMENTS	Requirement 8: `/req/relation/properties` Requirement 9: `/req/relation/relations`

REQUIREMENT 8	
IDENTIFIER	`/req/relation/properties`
INCLUDED IN	Requirements class 5: `/req-class/relation`
STATEMENT	Each `Relation` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 8.

Table 8 — Properties of a `Relation` entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY AND USE
externalObject	This URI references the external object for the `Relation` entity.	CharacterString	One (optional)
description	This is a short description of the corresponding `Relation` entity.	CharacterString	One (optional)
role	This URI references the definition of `Relation` entity.	URI	One (mandatory)

NAME	DEFINITION	DATA TYPE	MULTIPLICITY AND USE
properties	The SensorThings API definition applies	JSON Object	One (optional)

NOTE The subject of a relation entity is always an observation. For expressing the object of a relation, the `object` relation XOR `externalObject` property must be used.

REQUIREMENT 9	
IDENTIFIER	`/req/relation/relations`
INCLUDED IN	Requirements class 5: `/req-class/relation`
STATEMENT	Each `Relation` entity MAY have direct relations to other entity types listed in Table 9.

Table 9 — Direct relation between a `Relation` entity and other entity types

ENTITY TYPE	RELATION	DESCRIPTION
Observation	One mandatory to one optional	A `Relation` SHALL have one `Subject`.
Observation	One optional to one optional	A `Relation` MAY have zero-to-one `Object` XOR `externalObject`.
Group	Many optional to many optional	A `Relation` MAY have zero-to-many `Groups`.

7.6. Requirements Class Project

The `Project` entity can be used to create a container of `Datastream` or `MultiDatastream` entities. A Project can have a particular purpose and a managing party.

REQUIREMENTS CLASS 6: PROJECT	
IDENTIFIER	`/req-class/project`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service

REQUIREMENTS CLASS 6: PROJECT	
PREREQUISITE	Requirements class 1: `/req-class/entity-control-information`
NORMATIVE STATEMENTS	Requirement 10: `/req/project/properties` Requirement 11: `/req/project/relations`

REQUIREMENT 10	
IDENTIFIER	`/req/project/properties`
INCLUDED IN	Requirements class 6: `/req-class/project`
STATEMENT	Each `Project` entity SHALL have the mandatory properties and MAY have the optional properties listed in Table 10.

Table 10 — Properties of a `Project` entity

NAME	DEFINITION	DATA TYPE	MULTIPLICITY AND USE
name	A property provides a label for `Project` entity, commonly a descriptive name.	CharacterString	One (mandatory)
description	This is a short description of the corresponding `Project` entity.	CharacterString	One (mandatory)
classification	Determines if the data stream(s), multi data stream(s) or group(s) of the `Project` entity contain sensitive information	ValueCode	One (optional)
description	This is a short description of the corresponding `Project` entity.	CharacterString	One (mandatory)
creationTime	This is the starting time of the `Project` entity. Depending on the business logic, after this time it could be possible to add observations or relations to the Group.	TM Instant	One (optional)
endTime	This is the end time of the `Project` entity. Depending on the business logic, after this time it is no longer possible to add observations or relations to the Group.	TM Instant	One (optional)
termsOfUse	Express the term of use for the `Project` entity.	CharacterString	One (optional)
privacyPolicy	Express the term of use for personal data that are contained in the `Project` entity.	CharacterString	One (optional)

NAME	DEFINITION	DATA TYPE	MULTIPLICITY AND USE
url	This is the URL for the `Project` entity that provides additional information that cannot be captured in this entity alone.	URL	One (optional)

REQUIREMENT 11	
IDENTIFIER	`/req/project/relations`
INCLUDED IN	Requirements class 6: `/req-class/project`
STATEMENT	Each `Project` entity MAY have direct relations to other entity types listed in Table 11.

Table 11 — Direct relation between a `Project` entity and other entity types

ENTITY TYPE	RELATION	DESCRIPTION
Datastream	Many optional to many optional	A `Project` MAY have zero-to-many `Datastreams`.
MultiDatastream	Many optional to many optional	A `Project` MAY have zero-to-many `MultiDatastreams`.
Party	Many optional to one optional	A `Project` MAY have zero-to-one `Party`.
Group	Many optional to many optional	A `Project` MAY have zero-to-many `Group`.
License	Many optional to one optional	A `Project` MAY have zero-to-one `License`.

8

STAPLUS READ, CREATE, UPDATE AND DELETE REQUIREMENTS

8 STAPLUS READ, CREATE, UPDATE AND DELETE REQUIREMENTS

8.1. Overview

As many IoT devices are resource-constrained, the SensorThings API adopts the efficient REST web service style. That means the Read, Create, Update, Delete actions can be performed on the STAplus entity types.

8.2. Requirements Class Read

REQUIREMENTS CLASS 7: READ	
IDENTIFIER	`/req-class/read`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
PREREQUISITE	Requirements class 1: `/req-class/entity-control-information`
NORMATIVE STATEMENT	Requirement 12: `/req/read/entity`

REQUIREMENT 12	
IDENTIFIER	`/req/read/entity`
INCLUDED IN	Requirements class 7: `/req-class/read`
STATEMENT	To read an entity or a collection of entities, the client SHALL send a HTTP GET request to that entity or a collection's URL. If the target URL for the collection is a navigationLink, the entity is automatically linked to the entity or entities represented by the navigationLink. Upon successful completion, the response SHALL contain the representation of the entity or entities.

8.3. Requirements Class Create

REQUIREMENTS CLASS 8: CREATE	
IDENTIFIER	`/req-class/create`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
CONFORMANCE CLASS	Conformance class 2: `/conf/create`
PREREQUISITE	Requirements class 1: `/req-class/entity-control-information`
NORMATIVE STATEMENTS	Requirement 13: `/req/create/entity` Requirement 14: `/req/create/link-to-existing-entities` Requirement 15: `/req/create/deep-insert` Requirement 16: `/req/create/deep-insert-status-code`

REQUIREMENT 13	
IDENTIFIER	`/req/create/entity`
INCLUDED IN	Requirements class 8: `/req-class/create`
STATEMENT	To create an entity in a collection, the client SHALL send a HTTP POST request to that collection's URL. The POST body SHALL contain a single valid entity representation. If the target URL for the collection is a navigationLink, the new entity is automatically linked to the entity containing the navigationLink. Upon successful completion, the response SHALL contain a HTTP location header that contains the selfLink of the created entity. Upon successful completion the service SHALL respond with either 201 Created, or 204 No Content. Adapted from OData Version 4.01. Part 1: Protocol, §11.4.2 Create an Entity In addition, the link between entities SHALL be established upon creating an entity. Two use cases SHALL be considered: (1) link to existing entities when creating an entity, and (2) create related entities when creating an entity. The requests for these two use cases are described in the following subsection.

REQUIREMENT 14	
IDENTIFIER	`/req/create/link-to-existing-entities`
INCLUDED IN	Requirements class 8: `/req-class/create`
STATEMENT	A STAplus implementation that supports entity creation SHALL support linking new entities to existing entities upon creation. To create a new entity with links to existing entities in a single request, the client SHALL include the unique identifiers of the related entities associated with the corresponding navigation properties in the request body.

REQUIREMENT 15	
IDENTIFIER	`/req/create/deep-insert`
INCLUDED IN	Requirements class 8: `/req-class/create`
STATEMENT	A request to create an entity that includes related entities, represented using the appropriate inline representation, is referred to as a “deep insert”. A STAplus implementation that supports entity creation SHALL support deep insert. If the inline representation contains a value for a computed property (*i.e.*, id), the service SHALL ignore that value when creating the related entity. On success, the service SHALL create all entities and relate them. On failure, the service SHALL NOT create any of the entities. Adapted from OData Version 4.01. Part 1: Protocol, §11.4.2.2 Create Related Entities When Creating an Entity

REQUIREMENT 16	
IDENTIFIER	`/req/create/deep-insert-status-code`
INCLUDED IN	Requirements class 8: `/req-class/create`
STATEMENT	Upon successfully creating an entity, the service response SHALL contain a Location header that contains the URL of the created entity. Upon successful completion the service SHALL respond with 201 Created. Regarding all the HTTP status code, please refer to the HTTP Status Code section.

8.4. Requirements Class Update

REQUIREMENTS CLASS 9: UPDATE	
IDENTIFIER	`/req-class/update`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
CONFORMANCE CLASS	Conformance class 3: `/conf/update`
PREREQUISITES	Requirements class 1: `/req-class/entity-control-information` https://docs.ogc.org/is/18-088/18-088.html#req-create-update-delete-update-entity https://docs.ogc.org/is/18-088/18-088.html#req-create-update-delete-update-entity-put https://docs.ogc.org/is/18-088/18-088.html#req-create-update-delete-update-entity-jsonpatch
NORMATIVE STATEMENTS	Requirement 17: `/req/update/entity` Requirement 18: `/req/update/entity-put` Requirement 19: `/req/update/entity-jsonpatch`

REQUIREMENT 17	
IDENTIFIER	`/req/update/entity`
INCLUDED IN	Requirements class 9: `/req-class/update`
STATEMENT	To update an entity in a collection a STAplus implementation SHALL follow the requirements as defined in https://docs.ogc.org/is/18-088/18-088.html#req-create-update-delete-update-entity.

REQUIREMENT 18	
IDENTIFIER	`/req/update/entity-put`
INCLUDED IN	Requirements class 9: `/req-class/update`
STATEMENT	A STAplus implementation that supports updates with PUT SHALL follow the requirements as defined in https://docs.ogc.org/is/18-088/18-088.html#req-create-update-delete-update-entity.

REQUIREMENT 19	
IDENTIFIER	`/req/update/entity-jsonpatch`

REQUIREMENT 19	
INCLUDED IN	Requirements class 9: `/req-class/update`
STATEMENT	A STAplus implementation that supports updates with the JSON PATCH format SHALL follow the requirements as defined in https://docs.ogc.org/is/18-088/18-088.html#req-create-update-delete-update-entity-jsonpatch

8.5. Requirements Class Delete

REQUIREMENTS CLASS 10: DELETE	
IDENTIFIER	`/req-class/delete`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
CONFORMANCE CLASS	Conformance class 4: `/conf/delete`
PREREQUISITES	Requirements class 1: `/req-class/entity-control-information` https://docs.ogc.org/is/18-088/18-088.html#req-create-update-delete-delete-entity
NORMATIVE STATEMENT	Requirement 20: `/req/delete/entity`

REQUIREMENT 20	
IDENTIFIER	`/req/delete/entity`
INCLUDED IN	Requirements class 10: `/req-class/delete`
STATEMENT	To delete an entity in a collection a STAplus implementation SHALL follow the requirements as defined in https://docs.ogc.org/is/18-088/18-088.html#req-create-update-delete-delete-entity.

9

STAPLUS MQTT EXTENSION REQUIREMENTS

9 STAPLUS MQTT EXTENSION REQUIREMENTS

The MQTT capabilities allows that a client to receive changes to STAplus entities via MQTT.

NOTEThe publishing of observations as defined in OGC SensorThings API Part 1: Sensing Version 1.1 via MQTT does not apply to STAplus entities.

9.1. Overview

NOTEIn the context of MQTT, all STAplus entities as defined in this Standard are equivalent to the STA entities. Therefore, the implementation of MQTT capabilities must be compliant with the SensorThings API Part 1: Sensing v1.1 Standard.

9.2. Requirements Class MQTT Subscribe

REQUIREMENTS CLASS 11: MQTT SUBSCRIBE	
IDENTIFIER	`/req-class/mqtt-subscribe`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
CONFORMANCE CLASS	Conformance class 8: `/conf/mqtt-subscribe`
PREREQUISITE	http://www.opengis.net/spec/iot_sensing/1.1/req/receive-updates-via-mqtt
NORMATIVE STATEMENT	Requirement 21: `/req/mqtt-subscribe`

REQUIREMENT 21	
IDENTIFIER	`/req/mqtt-subscribe`
INCLUDED IN	Requirements class 11: `/req-class/mqtt-subscribe`

REQUIREMENT 21	
STATEMENT	A STAplus implementation SHALL support the receiving of STAplus entity updates with MQTT Subscribe as defined in http://www.opengis.net/spec/iot_sensing/1.1/req/receive-updates-via-mqtt.

10

STAPLUS BUSINESS LOGIC REQUIREMENTS

10 STAPLUS BUSINESS LOGIC REQUIREMENTS

10.1. Overview

The STAplus extension is defined based on requirements from the Citizen Science community. One major requirement from the Citizen Science community is that an implementation of STAplus can be used by many users simultaneously to create, update and delete observations. To ensure integrity of the entities and to prevent inconsistencies in the data, it is important that an implementation is not only compliant with this Standard but also has functionality that ensures entity consistency.

Even though the business logic can be very complex and most likely depend on many factors, providing a hint to developers and guiding end users of the implementation in case a request results in an unexpected response should be possible.

As a machine readable and understandable description of the implemented business logic is preferred, the idea of the business logic requirements class is to provide a URL where the business logic in defined in English text.

10.2. Requirements Class Business Logic

REQUIREMENTS CLASS 12: BUSINESS LOGIC	
IDENTIFIER	`/req-class/business-logic`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
CONFORMANCE CLASS	Conformance class 9: `/conf/business-logic`
NORMATIVE STATEMENTS	Requirement 22: `/req/business-logic/definition` Requirement 23: `/req/business-logic/location`

REQUIREMENT 22	
IDENTIFIER	/req/business-logic/definition
INCLUDED IN	Requirements class 12: /req-class/business-logic
STATEMENT	The implementation's business logic SHALL be described in English text.

REQUIREMENT 23	
IDENTIFIER	/req/business-logic/location
INCLUDED IN	Requirements class 12: /req-class/business-logic
STATEMENT	The implementation SHALL provide a JSON object in the serverSettings object on the landing page with the name http://www.opengis.net/doc/is/sensorthings/1.1/staplus/1.0/conf/business-logic that contains a property href which value is the URL of the HTML page describing the business logic.

11

STAPLUS AUTHENTICATION REQUIREMENTS

11 STAPLUS AUTHENTICATION REQUIREMENTS

11.1. Overview

To regulate access to entities using CRUD operations, implementations may wish to use information from user / client application authentication to make access control decisions.

In addition, some implementations may wish to link the authentication identifier (e.g. `REMOTE_USER`) with the `Party.authId`. Therefore, the STAplus authentication must provide deployment-wide unique user identifiers.

11.2. Requirements Class Authentication

REQUIREMENTS CLASS 13: AUTHENTICATION	
IDENTIFIER	`/req-class/authentication`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
CONFORMANCE CLASS	Conformance class 5: `/conf/authentication`
PREREQUISITE	Requirements class 2: `/req-class/party`
NORMATIVE STATEMENTS	Requirement 24: `/req/authentication/id` Requirement 25: `/req/authentication/anon-personal-data-crud` Requirement 26: `/req/authentication/own-personal-data-r` Requirement 27: `/req/authentication/own-personal-data-cud` Requirement 28: `/req/authentication/other-personal-data-r` Requirement 29: `/req/authentication/other-personal-data-cud`

REQUIREMENT 24	
IDENTIFIER	`/req/authentication/id`

REQUIREMENT 24	
INCLUDED IN	Requirements class 13: `/req-class/authentication`
STATEMENT	The STAplus authentication SHALL provide a user identifier permanent for same user but different for each user.

REQUIREMENT 25	
IDENTIFIER	`/req/authentication/anon-personal-data-crud`
INCLUDED IN	Requirements class 13: `/req-class/authentication`
STATEMENT	The STAplus implementation SHALL prevent any anonymous user access (read, create, update, and delete) to personal data stored in `Party.personalData`.

REQUIREMENT 26	
IDENTIFIER	`/req/authentication/own-personal-data-r`
INCLUDED IN	Requirements class 13: `/req-class/authentication`
STATEMENT	The STAplus implementation SHALL entitle a user to read their own personal data via a HTTP GET request to their own `Party` entity.

REQUIREMENT 27	
IDENTIFIER	`/req/authentication/own-personal-data-cud`
INCLUDED IN	Requirements class 13: `/req-class/authentication`
STATEMENT	The STAplus implementation SHALL entitle a user to create, update or delete their own personal data via a HTTP POST, PATCH, and DELETE request to their own `Party` entity.

REQUIREMENT 28	
IDENTIFIER	`/req/authentication/other-personal-data-r`
INCLUDED IN	Requirements class 13: `/req-class/authentication`

REQUIREMENT 28	
STATEMENT	The STAplus implementation SHALL prevent that a user can read personal data of other users via a HTTP GET request to `Party` entites of other users.

REQUIREMENT 29	
IDENTIFIER	`/req/authentication/other-personal-data-cud`
INCLUDED IN	Requirements class 13: `/req-class/authentication`
STATEMENT	The STAplus implementation SHALL prevent that a user can create, update or delete personal data of other users via a HTTP POST, PATCH, and DELETE request to `Party` entities of other users.

12

STAPLUS FEATURE AND LOCATION GEOMETRY ENCODING REQUIREMENTS

12 STAPLUS FEATURE AND LOCATION GEOMETRY ENCODING REQUIREMENTS

12.1. Overview

The `<Location>` and `<FeatureOfInterest>` entities enable storing data objects with unspecified structure. Both entities potentially store geometry information. The STAplus extension supports multi-user interactions including the creation, updating, reading and deletion of information. For example in Citizen Science, one service endpoint could accept that users can upload data using different applications. These applications might be developed by different parties and serve different purposes, and thus use different geometry structures and coordinate reference systems (CRS) to encode the coordinates of the geometries. This can result in the situation where the interoperability of uploaded geometry in the `<Location>` and `<FeatureOfInterest>` entities might become an issue.

For an implementation of STAplus, this situation becomes complicated when a `$filter` is leveraged that includes spatial conditions such as `ST_Within`. The filter expression does not carry any CRS information. Therefore, how does the implementation "know" how to apply a spatial filter to geometries stored in `feature` and `location` properties uploaded by different applications? Because the SensorThings data model does not define how to store CRS information inside the `feature` and `location`, it only recommends using GeoJSON. However, applying the `$filter` with spatial operators introduces two problems. First, the CRS data would need to be stored in a standardized location inside `feature` and `location` properties. Second, the implementation would need to apply coordinate transformation 'on the fly' when processing a spatial filter condition and stored geometries are encoded in different CRS. The first problem causes interoperability issues and the later inevitably causes performance issues.

To get out of this situation, the STAplus Standard defines a default storage-CRS based on WGS84 with axis-order longitude/latitude as defined in GeoJSON. Any uploaded geometry data encoded in the storage-CRS will be stored as-is. Any uploaded geometry data encoded differently will be transformed into the storage-CRS and then stored.

To indicate that the default storage-CRS and GeoJSON geometry encoding is used, the `encodingType` property of the `FeatureOfInterest` and `Location` entity is to be used with the value `application/geo+json`.

In addition, the STAplus extension supports other (commonly used) encodings via the `Geometry Encoding` Requirements Class.

12.2. Storage-CRS Requirements Class

This Requirements Class defines the default CRS with axis-order and its media-type.

REQUIREMENTS CLASS 14: STORAGE-CRS	
IDENTIFIER	/req-class/storage-crs
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
PREREQUISITE	GeoJSON
NORMATIVE STATEMENTS	Requirement 30: /req/storage-crs/crs-definition Requirement 31: /req/storage-crs/axis-order Requirement 32: /req/storage-crs/media-type Requirement 33: /req/storage-crs/processing

REQUIREMENT 30	
IDENTIFIER	/req/storage-crs/crs-definition
INCLUDED IN	Requirements class 14: /req-class/storage-crs
STATEMENT	The implementation SHALL use the GeoJSON CRS urn:ogc:def:crs:OGC::CRS84 as the storage-CRS.

REQUIREMENT 31	
IDENTIFIER	/req/storage-crs/axis-order
INCLUDED IN	Requirements class 14: /req-class/storage-crs
STATEMENT	The implementation SHALL use the GeoJSON axis-order with the storage-CRS.

REQUIREMENT 32	
IDENTIFIER	`/req/storage-crs/media-type`
INCLUDED IN	Requirements class 14: `/req-class/storage-crs`
STATEMENT	The implementation SHALL accept the media-type `encodingType=application/geo+json` to indicate that the structuring of the `FeatureOfInterest` and `Location` entities geometry encoding is compliant with the RFC GeoJSON.

REQUIREMENT 33	
IDENTIFIER	`/req/storage-crs/processing`
INCLUDED IN	Requirements class 14: `/req-class/storage-crs`
STATEMENT	The implementation SHALL enforce the `storage-CRS` to any geometry data contained in the `FeatureOfInterest` and `Location` entity. The implementation SHALL store geometry encoded in the `default-CRS` without further processing.

12.3. Geometry-FG Requirements Class

This Requirements Class defines the use of geometry encoding compliant with the OGC Draft Standard *OGC Features and Geometries JSON — Part 1: Core* (Geometry-FG)[3]

REQUIREMENTS CLASS 15: GEOMETRY-FG	
IDENTIFIER	`/req-class/geometry-fg`
OBLIGATION	requirement
TARGET TYPE	Target Type: Web Service
CONFORMANCE CLASS	Conformance class 6: `/conf/geometry-fg`
PREREQUISITE	OGC Features and Geometries JSON - Part 1: Core

[3] draft OGC Standard at the time of writing: https://docs.ogc.org/DRAFTS/21-045.html

REQUIREMENTS CLASS 15: GEOMETRY-FG	
NORMATIVE STATEMENTS	Requirement 34: /req/geometry-fg/media-type Requirement 35: /req/geometry-fg/default-crs Requirement 36: /req/geometry-fg/supported-crs Requirement 37: /req/geometry-fg/crs-error Requirement 38: /req/geometry-fg/processing Requirement 39: /req/geometry-fg/out

REQUIREMENT 34	
IDENTIFIER	/req/geometry-fg/media-type
INCLUDED IN	Requirements class 15: /req-class/geometry-fg
STATEMENT	The implementation SHALL accept the media-type application/vnd.ogc.fg+json as value to the encodingType property of the FeatureOfInterest and Location entities to indicate that the structuring of the geometry is be compliant with the OGC Draft Standard *OGC Features and Geometries JSON — Part 1: Core*[3].

[3] draft OGC Standard at the time of writing: https://docs.ogc.org/DRAFTS/21-045.html

REQUIREMENT 35	
IDENTIFIER	/req/geometry-fg/default-crs
INCLUDED IN	Requirements class 15: /req-class/geometry-fg
STATEMENT	The implementation SHALL advertise the default CRS on the conformance page.

REQUIREMENT 36	
IDENTIFIER	/req/geometry-fg/supported-crs
INCLUDED IN	Requirements class 15: /req-class/geometry-fg
STATEMENT	The implementation SHALL advertise the list of the supported CRS on the conformance page.

REQUIREMENT 37	
IDENTIFIER	`/req/geometry-fg/crs-error`
INCLUDED IN	Requirements class 15: `/req-class/geometry-fg`
STATEMENT	The implementation SHALL return an error if the geometry data inside `feature` or `location` properties is encoded in an unsupported CRS.

REQUIREMENT 38	
IDENTIFIER	`/req/geometry-fg/processing`
INCLUDED IN	Requirements class 15: `/req-class/geometry-fg`
STATEMENT	If necessary, the implementation SHALL apply a CRS transformation to the `default-CRS` if necessary before further processing or storing the geometry data.

REQUIREMENT 39	
IDENTIFIER	`/req/geometry-fg/out`
INCLUDED IN	Requirements class 15: `/req-class/geometry-fg`
STATEMENT	The implementation SHALL use the storage-CRS to encode the `feature` and `location` geometries in a response.

12.4. Geometry WKT Requirements Class

This Requirements Class defines the use of geometry encoding compliant with Well Known Text (WKT).

REQUIREMENTS CLASS 16: GEOMETRY WKT	
IDENTIFIER	`/req-class/geometry-wkt`
OBLIGATION	requirement

REQUIREMENTS CLASS 16: GEOMETRY WKT	
TARGET TYPE	Target Type: Web Service
CONFORMANCE CLASS	Conformance class 7: `/conf/geometry-wkt`
PREREQUISITE	Geographic information - Simple features access - Part 1: Common architecture
NORMATIVE STATEMENTS	Requirement 40: `/req/geometry-wkt/media-type` Requirement 16-2: `/req/geometry-wkt/crs-defintion` Requirement 42: `/req/geometry-wkt/default-crs` Requirement 43: `/req/geometry-wkt/supported-crs` Requirement 44: `/req/geometry-wkt/crs-error` Requirement 45: `/req/geometry-wkt/value` Requirement 46: `/req/geometry-wkt/processing` Requirement 47: `/req/geometry-wkt/out`

REQUIREMENT 40	
IDENTIFIER	`/req/geometry-wkt/media-type`
INCLUDED IN	Requirements class 16: `/req-class/geometry-wkt`
STATEMENT	The implementation SHALL accept the media-type `wkt` as value for the `encodingType` property of the `FeatureOfInterest` and `Location` entities.

REQUIREMENT 41	
IDENTIFIER	`/req/geometry-wkt/crs-definition`
STATEMENT	If a non-default CRS is used then either the CRS identifier SHALL be put into a property `crs`, or the CRS identifier (number) SHALL be put into a property `srid` of the `properties` property of the `FeatureOfInterest` or `Location` entity.

REQUIREMENT 42	
IDENTIFIER	`/req/geometry-wkt/default-crs`
INCLUDED IN	Requirements class 16: `/req-class/geometry-wkt`
STATEMENT	The implementation SHALL provide a JSON object in the `serverSettings` object on the landing page with the name `http://www.opengis.net/doc/is/sensorthings/1.1/staplus/1.0/`

REQUIREMENT 42

	`conf/geometry-wkt` that contains a property `default-crs` whose value represents the default CRS identifier.

REQUIREMENT 43

IDENTIFIER	`/req/geometry-wkt/supported-crs`
INCLUDED IN	Requirements class 16: `/req-class/geometry-wkt`
STATEMENT	The implementation SHALL provide a JSON object in the `serverSettings` object on the landing page with the name `http://www.opengis.net/doc/is/sensorthings/1.1/staplus/1.0/conf/geometry-wkt` that contains a property `supported-crs` of type Array which values represent the supported CRS identifiers.

REQUIREMENT 44

IDENTIFIER	`/req/geometry-wkt/crs-error`
INCLUDED IN	Requirements class 16: `/req-class/geometry-wkt`
STATEMENT	The implementation SHALL return an error if the geometry data inside the `FeatureOfInterest` or `Location` is encoded in an unsupported CRS.

REQUIREMENT 45

IDENTIFIER	`/req/geometry-wkt/value`
INCLUDED IN	Requirements class 16: `/req-class/geometry-wkt`
STATEMENT	The WKT encoded geometry SHALL be the value of the `feature` or `location` property (the type Any is a String).

REQUIREMENT 46

IDENTIFIER	`/req/geometry-wkt/processing`
INCLUDED IN	Requirements class 16: `/req-class/geometry-wkt`

REQUIREMENT 46	
STATEMENT	The implementation SHALL apply CRS transformation to the `storage-CRS` if necessary before further processing or storing the geometry data.

REQUIREMENT 47	
IDENTIFIER	`/req/geometry-wkt/out`
INCLUDED IN	Requirements class 16: `/req-class/geometry-wkt`
STATEMENT	The implementation SHALL use the storage-CRS to encode the `feature` and `location` geometries in a response.

13

MEDIA TYPES FOR FEATUREOFINTEREST AND LOCATION ENCODING

13 MEDIA TYPES FOR FEATUREOFINTEREST AND LOCATION ENCODING

This Standard offers support for different structuring of the `FeatureOfInterest` and `Location` entities. To indicate the support for a particular structure, an implementation can use one of the following media-types:

- `application/geo+json`
- `application/vnd.ogc.fg+json`
- `wkt` (there is no standard media type definition for WKT)

A

ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE

ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE

This normative section defines the STAplus 1.0 conformance classes tests.

NOTEA STAplus compliant implementation must also be compliant with the SensorThings API conformance as defined in OGC #18-088.

A.1. STAplus Core Conformance Class Tests

CONFORMANCE TEST A.1	
IDENTIFIER	`/conf/core/common-control-information`
REQUIREMENTS	Conformance class 1: `/conf/core` Requirement 1: `/req/common-control-information`
INCLUDED IN	Conformance class 1: `/conf/core`
TEST PURPOSE	To verify that the common control information as defined in the requirement.
TEST-METHOD-TYPE	Manually Inspect
TEST METHOD	Inspect the full JSON object of the entity sets (*i.e.*, without $select) to identify, if each entity has the common control information defined in the above requirement and the service sends appropriate responses as defined in this Standard.

CONFORMANCE TEST A.2	
IDENTIFIER	`/conf/core/entities`
REQUIREMENTS	Conformance class 1: `/conf/core` Requirement 2: `/req/party/properties`

CONFORMANCE TEST A.2	
	Requirement 3: `/req/party/relations` Requirement 4: `/req/license/properties` Requirement 5: `/req/license/relations` Requirement 6: `/req/group/properties` Requirement 7: `/req/group/relations` Requirement 8: `/req/relation/properties` Requirement 9: `/req/relation/relations` Requirement 10: `/req/project/properties` Requirement 11: `/req/project/relations`
INCLUDED IN	Conformance class 1: `/conf/core`
PREREQUISITE	Conformance test A.1: `/conf/core/common-control-information`
TEST PURPOSE	Verify that each STAplus entity has the mandatory properties and mandatory relations as defined in this specification.
TEST-METHOD-TYPE	Manually Inspect
TEST METHOD	Evaluate for each STAplus entity:
A	Inspect the full JSON object of the entity sets (*i.e.*, without $select) to identify, if each entity has the mandatory properties defined in the corresponding requirement.
B	Inspect the full JSON object of each entity set (*i.e.*, without using the $select query option) to identify, if each entity has the mandatory relations (*i.e.*, @iot.navigationLink) defined in the corresponding requirement.

CONFORMANCE TEST A.3	
IDENTIFIER	`/conf/core/read`
REQUIREMENTS	Conformance class 1: `/conf/core` Requirement 12: `/req/read/entity`
INCLUDED IN	Conformance class 1: `/conf/core`
PREREQUISITE	Conformance test A.2: `/conf/core/entities`
TEST PURPOSE	Verify that the implementation supports reading STAplus entities via HTTP GET.
TEST-METHOD-TYPE	Manually Inspect
TEST METHOD	Evaluate that the implementation accepts a Sensor Things API compliant HTTP Get request to the STAplus entities:

CONFORMANCE TEST A.3	
A	Construct a URL to the `Party` entity and verify the response.
B	Construct a URL to the `License` entity and verify the response.
C	Construct a URL to the `Group` entity and verify the response.
D	Construct a URL to the `Relation` entity and verify the response.
E	Construct a URL to the `Project` entity and verify the response.

CONFORMANCE TEST A.4	
IDENTIFIER	`/conf/core/storage-crs/crs-definition`
REQUIREMENTS	Conformance class 1: `/conf/core` Requirement 30: `/req/storage-crs/crs-definition`
INCLUDED IN	Conformance class 1: `/conf/core`
TEST PURPOSE	Verify that the implementation supports and uses the default CRS.
TEST METHOD	Evaluate that the implementation uses the default CRS.
A	Construct a `Location` entity that contains a `Location` property whose geometry is encoded using the default CRS and check that the implementation is processing the geometry accordingly and that the geometry data is stored using the default CRS.
B	Construct a `FeatureOfInterst` entity that contains a `Feature` property whose geometry is encoded using the default CRS and check that the implementation is processing the geometry accordingly and that the geometry data is stored using the default CRS.

CONFORMANCE TEST A.5	
IDENTIFIER	`/conf/core/storage-crs/axis-order`
REQUIREMENTS	Conformance class 1: `/conf/core` Requirement 31: `/req/storage-crs/axis-order`
INCLUDED IN	Conformance class 1: `/conf/core`
TEST PURPOSE	Verify that the implementation supports and uses the default axis-order.

CONFORMANCE TEST A.5	
TEST METHOD	Evaluate that the implementation uses the default axis-order.
A	Construct a `Location` entity that contains a `location` property whose geometry is encoded using the default axis-order and check that the implementation is processing the geometry accordingly and that the geometry data is stored using the default CRS.
B	Construct a `FeatureOfInterst` entity that contains a `feature` property whose geometry is encoded using the default axis-order and check that the implementation is processing the geometry accordingly and that the geometry data is stored using the default CRS.

CONFORMANCE TEST A.6	
IDENTIFIER	`/conf/core/storage-crs/media-type`
REQUIREMENTS	Conformance class 1: `/conf/core` Requirement 32: `/req/storage-crs/media-type`
INCLUDED IN	Conformance class 1: `/conf/core`
TEST PURPOSE	Verify that the implementation supports and uses the default media-type.
TEST METHOD	Evaluate that the implementation uses the default media-type.
A	Construct a `Location` entity that contains a `location` property whose geometry is encoded using the default CRS and axis-order where the `encodingType` property’s value is `application/geo+json` and check that the implementation is processing the geometry accordingly and that the geometry data is stored using the default CRS and axis-order.
B	Construct a `FeatureOfInterst` entity that contains a `feature` property whose geometry is encoded using the default CRS and axis-order where the `encodingType` property’s value is `application/geo+json` and check that the implementation is processing the geometry accordingly and that the geometry data is stored using the default CRS and axis-order.

CONFORMANCE TEST A.7	
IDENTIFIER	`/conf/core/storage-crs/processing`
REQUIREMENTS	Conformance class 1: `/conf/core` Requirement 33: `/req/storage-crs/processing`
INCLUDED IN	Conformance class 1: `/conf/core`
TEST PURPOSE	Verify that the implementation stores geometry that is encoded in the default CRS and axis-order without processing.

CONFORMANCE TEST A.7	
TEST METHOD	Evaluate that the implementation stores geometry that is encoded in the default CRS and axis-order without processing.
A	Construct a `Location` entity that contains a `location` property whose geometry is encoded using the default CRS and axis-order where the `encodingType` property's value is `application/geo+json` and check that the implementation is stores the geometry data without processing.
B	Construct a `FeatureOfInterst` entity that contains a `feature` property whose geometry is encoded using the default CRS and axis-order where the `encodingType` property's value is `application/geo+json` and check that the implementation is storing the geometry data without a CRS transformation.

A.2. STAplus Create Conformance Class Tests

CONFORMANCE TEST A.8	
IDENTIFIER	`/conf/create/http`
REQUIREMENTS	Conformance class 2: `/conf/create` Requirement 13: `/req/create/entity` Requirement 14: `/req/create/link-to-existing-entities` Requirement 15: `/req/create/deep-insert` Requirement 16: `/req/create/deep-insert-status-code`
INCLUDED IN	Conformance class 2: `/conf/create`
TEST PURPOSE	To verify that the service implementation supports the creation of entities as defined in this Standard.
TEST METHOD	For each STAplus entity:
A	Create an entity instance by following the integrity constraints and creating the related entities with a single request (*i.e.*, deep insert), check if the entity instance is successfully created and the implementation responds as defined in this Standard.
B	Create an entity instance and its related entities with a deep insert request that does not conform to the Standard (e.g., missing a mandatory property), check if the service fails the request without creating any entity within the deep insert request and responds the appropriate HTTP status code.
C	Issue an entity creation request that does not follow the integrity constraints with deep insert, check if the service fails the request without creating any entity within the deep insert request and responds the appropriate HTTP status code.

CONFORMANCE TEST A.8	
D	Create an entity instance by linking to existing entities with a single request, check if the server responds as defined in this Standard.
E	Create an entity instance that does not follow the integrity constraints by linking to existing entities with a single request, check if the server responds as defined in this specification.

A.3. STAplus Update Conformance Class Tests

CONFORMANCE TEST A.9	
IDENTIFIER	`/conf/update/put`
REQUIREMENTS	Conformance class 3: `/conf/update` Requirement 17: `/req/update/entity` Requirement 18: `/req/update/entity-put`
INCLUDED IN	Conformance class 3: `/conf/update`
TEST PURPOSE	To verify that the service implementation supports the update of entities as defined in this specification.
TEST METHOD	For each STAplus entity:
A	Send an update request with HTTP PUT and check if the service responds as defined.

CONFORMANCE TEST A.10	
IDENTIFIER	`/conf/update/patch`
REQUIREMENTS	Conformance class 3: `/conf/update` Requirement 17: `/req/update/entity` Requirement 19: `/req/update/entity-jsonpatch`
INCLUDED IN	Conformance class 3: `/conf/update`
TEST PURPOSE	To verify that the service implementation supports the update of entities as defined in this Standard.
TEST METHOD	For each STAplus entity:
A	Send an update request with PATCH, check (1) if the properties provided in the payload corresponding to updatable properties replace the value of the corresponding property in the

CONFORMANCE TEST A.10	
	entity and (2) if the missing properties of the containing entity or complex property are not directly altered.
B	Send an update request with PATCH that contains related entities as inline content, check if the service fails the request and returns appropriate HTTP status code.
C	Send an update request with PATCH that contains binding information for navigation properties, check if the service updates the navigationLink accordingly.

A.4. STAplus Delete Conformance Class Tests

CONFORMANCE TEST A.11	
IDENTIFIER	`/conf/delete/entity`
REQUIREMENTS	Conformance class 4: `/conf/delete` Requirement 20: `/req/delete/entity`
INCLUDED IN	Conformance class 4: `/conf/delete`
TEST PURPOSE	To verify that the service implementation supports the deletion of entities as defined
TEST METHOD	For each STAplus entity:
A	Delete an entity instance, and check if the service responds as defined

A.5. STAplus Authentication Conformance Class Tests

CONFORMANCE TEST A.12	
IDENTIFIER	`/conf/authentication/id`
REQUIREMENTS	Conformance class 5: `/conf/authentication` Requirement 24: `/req/authentication/id`
INCLUDED IN	Conformance class 5: `/conf/authentication`

CONFORMANCE TEST A.12

TEST PURPOSE	To verify that the user’s identifier is permanent and unique.
TEST METHOD	Verify the following:
A	Compare the user identifier after repeated login of the same user and verify that the identifier is identical.
B	Compare the user identifier for different users and verify that the identifiers are different.

CONFORMANCE TEST A.13

IDENTIFIER	`/conf/authentication/anon-personal-data-crud`
REQUIREMENTS	Conformance class 1: `/conf/core` Requirement 25: `/req/authentication/anon-personal-data-crud`
INCLUDED IN	Conformance class 5: `/conf/authentication`
TEST PURPOSE	To verify that an anonymous user cannot read, create, update or delete personal data stored in any `Party.personalData`.
TEST METHOD	For an existing `Party` entity:
A	Submit a HTTP GET request to any (all) `Party` entity(ies) and check that the response does not contain the `personalData` property.
B	Submit a HTTP POST, PATCH, and DELETE request to any (all) `Party` entity(ies) and check that the response is compliant with the business logic.

CONFORMANCE TEST A.14

IDENTIFIER	`/conf/authentication/own-personal-data-crud`
REQUIREMENTS	Conformance class 5: `/conf/authentication` Requirement 26: `/req/authentication/own-personal-data-r` Requirement 27: `/req/authentication/own-personal-data-cud`
INCLUDED IN	Conformance class 5: `/conf/authentication`
TEST PURPOSE	To verify that a user can read, create, update and delete the own personal data stored in `Party.personalData`.
TEST METHOD	Verify that access to the own personal data is possible for an authenticated user by sending HTTP requests with different methods to the `Party` entity that represents the user:

CONFORMANCE TEST A.14	
A	Have the user authenticate and identify the corresponding `Party` entity.
B	Construct a HTTP POST request to create a `Party` entity including personal data and verify that the entity is stored.
C	Construct a HTTP GET request to the corresponding `Party` entity and verify that the personal data is contained in the response.
D	Construct a HTTP PATCH request to update the personal data of the corresponding `Party`entity. Verify that the update was successful.
E	Construct a HTTP PATCH request to delete the personal data (set values to `null`) of the corresponding `Party` entity. Verify that the erasure of the personal data was successful.

CONFORMANCE TEST A.15	
IDENTIFIER	`/conf/authentication/other-personal-data-crud`
REQUIREMENTS	Conformance class 5: `/conf/authentication` Requirement 28: `/req/authentication/other-personal-data-r` Requirement 29: `/req/authentication/other-personal-data-cud`
INCLUDED IN	Conformance class 5: `/conf/authentication`
TEST PURPOSE	To verify that a user can **not** read, create, update and delete **other user's** personal data stored in `Party.personalData`.
TEST METHOD	Verify that access to other personal data is **not** possible for an authenticated user by sending HTTP requests with different methods to the `Party` entity that represents **another** user:
A	Have the user authenticate and identify a `Party` entity of another user.
B	Construct a HTTP POST request to create a `Party` entity including personal data using a `partyId` value for another user. Verify that the response is compliant with the business logic.
C	Construct a HTTP GET request to `Party` entity of another user and verify that the response is compliant with the business logic.
D	Construct a HTTP PATCH request to update the personal data of another `Party`entity. Verify that the response is compliant with the business logic.
E	Construct a HTTP PATCH request to delete the personal data (set values to `null`) of **another** `Party` entity. Verify that the response is compliant with the business logic.

A.6. STAplus Business Logic Conformance Class Tests

CONFORMANCE TEST A.16	
IDENTIFIER	/conf/business-logic/definition
REQUIREMENTS	Conformance class 9: /conf/business-logic Requirement 22: /req/business-logic/definition
INCLUDED IN	Conformance class 9: /conf/business-logic
TEST PURPOSE	To verify that the description of the business logic is human readable and in English.
TEST METHOD	Verify that the HTML page for the business logic is in English language.

CONFORMANCE TEST A.17	
IDENTIFIER	/conf/business-logic/location
REQUIREMENTS	Conformance class 9: /conf/business-logic Requirement 23: /req/business-logic/location
INCLUDED IN	Conformance class 9: /conf/business-logic
TEST PURPOSE	To verify that the business logic is available from the provided URL.
TEST METHOD	On the landing page, find the JSON object with name http://www.opengis.net/doc/is/sensorthings/1.1/staplus/1.0/conf/business-logic and follow the link provided in the href property. Verify that the loaded HTML page contains the description of the business logic.

A.7. STAplus Geometry FG Conformance Class Tests

CONFORMANCE TEST A.18	
IDENTIFIER	/conf/geometry-fg//media-type
REQUIREMENTS	Conformance class 6: /conf/geometry-fg

CONFORMANCE TEST A.18	
	Requirement 34: `/req/geometry-fg/media-type`
INCLUDED IN	Conformance class 6: `/conf/geometry-fg`
TEST PURPOSE	To verify that the implementation accepts media-type for Geometry-FG.
TEST METHOD	Verify that the implementation supports the use of the media-type for Geometry-FG.

CONFORMANCE TEST A.19	
IDENTIFIER	`/conf/geometry-fg/default-crs`
REQUIREMENTS	Conformance class 6: `/conf/geometry-fg` Requirement 35: `/req/geometry-fg/default-crs`
INCLUDED IN	Conformance class 6: `/conf/geometry-fg`
TEST PURPOSE	To verify that the default-CRS is used for processing geometry data from `Feature` and `Location`.
TEST METHOD	Verify that the implementation applies the default CRS advertised in the conformance page to the geometry data from `Feature` and `Location`.

CONFORMANCE TEST A.20	
IDENTIFIER	`/conf/geometry-fg/supported-crs`
REQUIREMENTS	Conformance class 6: `/conf/geometry-fg` Requirement 36: `/req/geometry-fg/supported-crs`
INCLUDED IN	Conformance class 6: `/conf/geometry-fg`
TEST PURPOSE	To verify that all CRS, advertised as supported in the conformance page are accepted.
TEST METHOD	Verify that the implementation accepts geometry encodings for `Feature` and `Location`. For each supported CRS:
A	Construct a geometry and create a `Location` and `FeatureOfInterest` entity. Verify that the geometry data is accepted by the implementation.

CONFORMANCE TEST A.21	
IDENTIFIER	`/conf/geometry-fg/crs-error`

CONFORMANCE TEST A.21	
REQUIREMENTS	Conformance class 6: `/conf/geometry-fg` Requirement 37: `/req/geometry-fg/crs-error`
INCLUDED IN	Conformance class 6: `/conf/geometry-fg`
TEST PURPOSE	To verify that no additional CRS, as advertised as supported in the conformance page are accepted.
TEST METHOD	Verify that the implementation **does not** accept geometry encodings for `Feature` and `Location` that are not listed as supported. For a CRS **not** listed as supported:
A	Construct a geometry and create a `Location` and `FeatureOfInterest` entity. Verify that the geometry data is **rejected** by the implementation.

CONFORMANCE TEST A.22	
IDENTIFIER	`/conf/geometry-fg/processing`
REQUIREMENTS	Conformance class 6: `/conf/geometry-fg` Requirement 38: `/req/geometry-fg/processing`
INCLUDED IN	Conformance class 6: `/conf/geometry-fg`
TEST PURPOSE	To verify that a geometry not encoded in the storage-CRS is transformed before storage.
TEST METHOD	Verify that the implementation accepts geometry encodings for `Feature` and `Location` that use a supported CRS:
A	Construct a geometry and create a `Location` and `FeatureOfInterest` entity. Verify that the geometry data is accepted and transformed to the storage-CRS before processed and stored by the implementation.

CONFORMANCE TEST A.23	
IDENTIFIER	`/conf/geometry-fg/out`
REQUIREMENTS	Conformance class 6: `/conf/geometry-fg` Requirement 39: `/req/geometry-fg/out`
INCLUDED IN	Conformance class 6: `/conf/geometry-fg`
TEST PURPOSE	To verify that a geometry included in a response is encoded in the storage-CRS.
TEST METHOD	Verify that the geometry data for a `Feature` and `Location` is using storage-CRS, independent from the geometry CRS used with the creation or updating of the entity.

A.8. STAplus Geometry WKT Conformance Class Tests

CONFORMANCE TEST A.24	
IDENTIFIER	`/conf/geometry-wkt/media-type`
REQUIREMENTS	Conformance class 7: `/conf/geometry-wkt` Requirement 40: `/req/geometry-wkt/media-type`
INCLUDED IN	Conformance class 7: `/conf/geometry-wkt`
TEST PURPOSE	To verify that the implementation accepts media-type for WKT.
TEST METHOD	Verify that the implementation supports the use of the media-type for WKT.

CONFORMANCE TEST A.25	
IDENTIFIER	`/conf/geometry-wkt/crs-definition`
REQUIREMENTS	Conformance class 7: `/conf/geometry-wkt` Requirement 41: `/req/geometry-wkt/crs-definition`
INCLUDED IN	Conformance class 7: `/conf/geometry-wkt`
TEST PURPOSE	To verify that the implementation accepts CRS definition provided in the associated property.
TEST METHOD	Verify that the implementation supports the use of the CRS property.
A	Construct a WKT geometry in a CRS different from the `default-crs`.
B	Set the `crs` property to the CRS identifier.
C	Verify that the implementation processes the geometry honoring the CRS identified by the `crs` value.
D	Set the `srid` property to the CRS identifier number.
E	Verify that the implementation processes the geometry honoring the CRS identified by the `srid` value.

CONFORMANCE TEST A.26	
IDENTIFIER	`/conf/geometry-wkt/default-crs`
REQUIREMENTS	Conformance class 7: `/conf/geometry-wkt` Requirement 42: `/req/geometry-wkt/default-crs`
INCLUDED IN	Conformance class 7: `/conf/geometry-wkt`
TEST PURPOSE	To verify that the default-CRS is used for processing geometry data from `feature` and `location` property.
TEST METHOD	Verify that the implementation defines and applies the default CRS to the geometry data from `feature` and `location` property.
A	Find the JSON object in the `serverSettings` object on the landing page with the name `http://www.opengis.net/doc/is/sensorthings/1.1/staplus/1.0/conf/geometry-wkt` and check the value of the property `default-crs`.
B	Verify that the `default-crs` is applied to a WKT geometry if no `crs` or `srid` property is used.

CONFORMANCE TEST A.27	
IDENTIFIER	`/conf/geometry-wkt/supported-crs`
REQUIREMENTS	Conformance class 7: `/conf/geometry-wkt` Requirement 43: `/req/geometry-wkt/supported-crs`
INCLUDED IN	Conformance class 7: `/conf/geometry-wkt`
TEST PURPOSE	To verify that all supported CRS are accepted.
TEST METHOD	Verify that the implementation accepts geometry encodings for `feature` and `location` properties. For each supported CRS:
A	Execute test /conf/geometry-wkt/crs-definition and verify that the implementation processes the geometry correctly.

CONFORMANCE TEST A.28	
IDENTIFIER	`/conf/geometry-wkt/crs-error`
REQUIREMENTS	Conformance class 7: `/conf/geometry-wkt` Requirement 44: `/req/geometry-wkt/crs-error`
INCLUDED IN	Conformance class 7: `/conf/geometry-wkt`

CONFORMANCE TEST A.28	
TEST PURPOSE	To verify that only supported CRSs are accepted.
TEST METHOD	Verify that the implementation **does not** accept geometry encodings for `feature` and `location` properties that are not listed as supported. For a CRS **not** listed as supported:
A	Execute test /conf/geometry-wkt/crs-definition and verify that the geometry data is **rejected** by the implementation.

CONFORMANCE TEST A.29	
IDENTIFIER	`/conf/geometry-wkt/value`
REQUIREMENTS	Conformance class 7: `/conf/geometry-wkt` Requirement 45: `/req/geometry-wkt/value`
INCLUDED IN	Conformance class 7: `/conf/geometry-wkt`
TEST PURPOSE	To verify that the geometry value, compliant to WKT is accepted as value for the `feature` and `location` property.
TEST METHOD	Verify that the implementation accepts WKT geometry values for `feature` and `location` properties.

CONFORMANCE TEST A.30	
IDENTIFIER	`/conf/geometry-wkt/processing`
REQUIREMENTS	Conformance class 7: `/conf/geometry-wkt` Requirement 46: `/req/geometry-wkt/processing`
INCLUDED IN	Conformance class 7: `/conf/geometry-wkt`
TEST PURPOSE	To verify that a geometry not encoded in the storage-CRS is transformed before storage.
TEST METHOD	Verify that the implementation accepts geometry encodings for `Feature` and `Location` that use a supported CRS:
A	Construct a geometry and create a `Location` and `FeatureOfInterest` entity. Verify that the geometry data in the `location` and `feature` properties is accepted and transformed to the storage-CRS before processed and stored by the implementation.

CONFORMANCE TEST A.31	
IDENTIFIER	`/conf/geometry-wkt/out`
REQUIREMENTS	Conformance class 7: `/conf/geometry-wkt` Requirement 47: `/req/geometry-wkt/out`
INCLUDED IN	Conformance class 7: `/conf/geometry-wkt`
TEST PURPOSE	To verify that a geometry included in a response is encoded in the storage-CRS.
TEST METHOD	Verify that the geometry data for a `feature` and `location` properties is using storage-CRS, independent from the geometry CRS used with the creation or updating of the entity.

A.9. STAplus MQTT Subscribe Conformance Class Tests

CONFORMANCE TEST A.32	
IDENTIFIER	`/conf/mqtt-subscribe/definition`
REQUIREMENTS	Conformance class 8: `/conf/mqtt-subscribe` Requirement 21: `/req/mqtt-subscribe`
INCLUDED IN	Conformance class 8: `/conf/mqtt-subscribe`
TEST PURPOSE	To verify that a client can receive notifications for the updates of a STAplus entity set or an individual entity with MQTT.
TEST METHOD	For each STAplus entity:
A	Subscribe to an entity set with MQTT Subscribe. Then create a new entity of the subscribed entity set. Check if a complete JSON representation of the newly created entity through MQTT is received.
B	Subscribe to an entity set with MQTT Subscribe. Then update an existing entity of the subscribed entity set. Check if a complete JSON representation of the updated entity through MQTT is received.
C	part:: Subscribe to an entity's property with MQTT Subscribe. Then update the property with PATCH. Check if the JSON object of the updated property is received.
DESCRIPTION	Subscribe to multiple properties of an entity set with MQTT Subscribe. Then create a new entity of the entity set. Check if a JSON object of the subscribed properties is received. part Subscribe to multiple properties of an entity set with MQTT Subscribe. Then update an existing entity of the entity set with

CONFORMANCE TEST A.32	
	PATCH. Check if a JSON object of the subscribed properties is received.

B

ANNEX B (INFORMATIVE) REVISION HISTORY

B

ANNEX B (INFORMATIVE) REVISION HISTORY

Table B.1

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2022-09-21	0.1	Andreas Matheus	all	Initial version
2022-10-21	0.2	Andreas Matheus	mainly Annex A, B	Updating conformance class structure and abstract test site
2022-11-03	0.3	Andreas Matheus	section Business Logic, Annex A, B	Section added for Business Logic, Control Information and Authentication, Updating conformance class structure and abstract test site to include Business Logic, Adding Requirements and Tests for accessing personal data
2022-12-19	0.4	Andreas Matheus	STAplus Feature and Location Encoding, Annex A, Annex B, Conformance	incorporating requirements for encoding Feature and Location entities reflecting results from SensorThings SWG session on 7 December 2022
2023-01-17	0.5	Andreas Matheus	Mainly normative sections and Annex A, B	Applied OGC NA-Policy using Metanorma annotations, Annex-B merged into Annex-A, Conformance Class definition now in section Conformance
2023-01-20	0.6	Andreas Matheus	All sections	Title adopted to include versions, updated requirements for Business Logic and CRS conformance classes, incorporated proof read from Hylke van der Schaaf

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Andreas Matheus: OGC 21-068, *OGC Best Practice for using SensorThings API with Citizen Science*. Open Geospatial Consortium (2022). https://docs.ogc.org/bp/21-068.pdf.

[2] *OGC Features and Geometries JSON — Part 1: Core*, (2021)[3]

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