OGC® DOCUMENT: 23-002

External identifier of this OGC® document: http://www.opengis.net/doc/IS/ogcapi-connectedsystems-2/1.0

Open Geospatial Consortium

OGC API - CONNECTED SYSTEMS - PART 2: DYNAMIC DATA

STANDARD
Implementation

APPROVED

Version: 1.0
Submission Date: 2025-03-19
Approval Date: 2025-06-02
Publication Date: 2025-07-16
Editor: Alexandre Robin

License Agreement

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

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

Copyright notice

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

OGC API Standards define modular API building blocks to spatially enable Web APIs in a consistent way. The OpenAPI specification is used to define the API building blocks.

The OGC API family of Standards is organized by resource type. This Standard specifies the fundamental API building blocks for interacting with Connected Systems and associated resources. A Connected System represents any kind of system that can either directly transmit data via communication networks (being connected to them in a permanent or temporary fashion), or whose data is made available in one form or another via such networks. This definition encompasses systems of all kinds, including in-situ and remote sensors, actuators, fixed and mobile platforms, airborne and space-borne systems, robots and drones, and even humans who collect data or execute specific tasks.

Since many of the resource types defined in this document, including the systems themselves, are also features, the OGC API — Connected Systems Standard is logically written as an extension of OGC API — Features.

But beyond features, this Standard is also intended to act as a bridge between static data (geographic and other application domain features) and dynamic data (observations of these features properties, and commands/actuations that change these features properties). To this end, this Standard also describes protocols and formats to transmit dynamic data to/from connected systems through the API. Some of these protocols allow efficient real-time delivery of data while some others are more suited for transmitting data in batch.

In addition to providing its own mechanism for interacting with static and dynamic data, the API allows linking to other APIs from the OGC ecosystem, such as 3D GeoVolumes, 3D Tiles, Coverages, EDR, SensorThings, Processes, and other Features API instances. Among other things, this linking capability allows one to retrieve more advanced representations of features of interest (3D buildings, etc.) and gridded data (coverages) than the one that would typically be provided through this API.

The API is comprised of multiple parts, each of them being a separate standard. “Part 1 — Feature Resources” defines the feature types and corresponding schemas for some concepts of the Semantic Sensor Network Ontology (SOSA/SSN). This part (“Part 2 — Dynamic Data”) defines the resources, encodings and protocols that allow efficient exchange of dynamic (time-varying) data related to these features, in a way that is also aligned with SOSA/SSN.

II KEYWORDS

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

ogcdoc, OGC document, OpenAPI, REST, feature, API, system, smart system, connected system, IoT, sensorweb, ssn, sensor, actuator, transducer, sampling, platform, robot, drone, unmanned, autonomous, observation, measurement, datastream, command, control, trajectory, dynamic

PREFACE

The OGC API — Connected Systems Standard is part of the suite of OGC API Standards.

To increase the brevity and readability of this Standard, many OGC document titles are shortened and/or abbreviated. Therefore, in the context of this document, the following phrases are defined.

- "this Standard" shall be interpreted as equivalent to "OGC API — Connected Systems — Part 2: Dynamic Resources Standard."
- "CS API" or "CS API Standard" shall be interpreted as equivalent to "OGC API — Connected Systems Standard" (including all its parts).
- "OGC API — Features" shall be interpreted as equivalent to "OGC API — Features — Part 1: Core corrigendum."
- "OGC API — Common" shall be interpreted as equivalent to "OGC API — Common — Part 1: Core."

SECURITY CONSIDERATIONS

All security considerations detailed in OGC API — Connected Systems — Part 1 also apply to this Standard.

V SUBMITTING ORGANIZATIONS

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

- GeoRobotix, Inc.
- Botts Innovative Research, Inc.
- Cesium GS, Inc.
- 52°North Spatial Information Research GmbH
- Riverside Research
- Pelagis Data Solutions
- National Geospatial-Intelligence Agency (NGA)

VI SUBMITTERS

All questions regarding this submission should be directed to the editor or the submitters listed in the following table:

NAME	AFFILIATION
Alex Robin (Editor)	GeoRobotix, Inc.
Christian Autermann	52° North Spatial Information Research GmbH
Chuck Heazel	Heazeltech
Glenn Laughlin	Pelagis Data Solutions
Mike Botts	Botts Innovative Research, Inc.
Patrick Cozzi	Cesium GS, Inc.
Sam Bolling	Riverside Research

Additional contributors to this Standard include the following:

NAME	AFFILIATION
Chris Tucker	GeoRobotix, Inc.
Ian Patterson	Botts Innovative Research, Inc.
Qihua Li	GovTech Singapore
Rob Atkinson	Open Geospatial Consortium, Inc.
Simon Cox	Open Geospatial Consortium, Inc.
Jan Speckamp	52°North Spatial Information Research GmbH

1

SCOPE

1 SCOPE

This Standard defines extensions to OGC API — Features for exposing metadata and dynamic data regarding all kinds of observing systems and associated resources. It provides an actionable implementation of concepts defined in the Semantic Sensor Network Ontology (SSN) and also complies with OGC API — Common.

More specifically, Part 2 of the API, specified in this document, implements the SSN concepts allowing exchange of dynamic (possibly real-time) data flowing to and from various types of connected systems (e.g., sensors, actuators, platforms). Several encoding formats are defined in this Standard, including JSON, CSV and binary formats based on the OGC — SWE Common Standard. Additional encodings can be added by extensions.

The following types of resources are defined by Part 2 of the API.

- **DataStream** resources represent data feeds coming out of `Systems` and are containers for `Observations`. They are used for receiving and ingesting real-time `Observations` as well as accessing historical `Observations`. A `DataStream` is a particular case of `sosa:ObservationCollection` where all `Observations` are coming from the same sensor.
- **Observation** resources record all information regarding an act of observation, whether it is made by an automated device or a human. In particular, they carry the observation result that is structured according to a well defined schema. `Observations` are grouped into `DataStreams`.
- **ControlStream** resources represent data feeds going into `Systems` and are containers for `Commands`. They are used for receiving and ingesting real-time `Commands` as well as accessing historical `Commands`.
- **Command** resources represent messages sent to a `System` to control the parameters of feature of interest. In particular, a command includes parameters that are structured according to a well define schema.
- **CommandStatus** resources provide status reports during the execution of a command.
- **SystemEvent** resources provide information about a system event, such as sensor activation, recalibration, maintenance, etc.

CS API Part 1 defines the feature resources that these dynamic data feeds are associated to, including both the `Systems` that produce or receive these data feeds, and the features of interest that these data feeds provide information about.

CS API Part 3 defines pub/sub protocol bindings for exchanging dynamic data that can be used jointly with the HTTP API.

2

CONFORMANCE

2 CONFORMANCE

This Standard was written to be compliant with the OGC Specification Model – A Standard for Modular Specification (OGC 08-131r3). Extensions of this Standard shall themselves be conformant to the OGC Specification Model.

This Standard defines the following requirements classes.

- Clause 8, Requirements Class "Common" defines requirements that are shared by several other requirements classes.
- Clause 9, Requirements Class "Datastreams Observations" defines requirements for `DataStream` and `Observation` resources.
- Clause 10, Requirements Class "Control Streams Commands" defines requirements for `ControlStream` and `Command` resources.
- Clause 11, Requirements Class "Command Feasibility" defines requirements for `Feasibility` resources.
- Clause 12, Requirements Class "System Events" defines requirements for `SystemEvent` resources.
- Clause 13, Requirements Class "Advanced Filtering" defines requirements for additional filters that can be used to query *CS resources**.
- Clause 14, Requirements Class "Create/Replace/Delete" defines requirements for creating, replacing, and deleting *CS resources**.
- Clause 15, Requirements Class "Update" defines requirements for updating *CS resources**.
- Clause 16.1, Requirements Class "JSON Encoding" defines requirements for encoding *CS resources** as JSON.
- Clause 16.2, Requirements Class "SWE Common JSON Encoding" defines requirements for encoding `Observation` and `Command` resources using SWE Common JSON Encoding rules.
- Clause 16.3, Requirements Class "SWE Common Text Encoding" defines requirements for encoding `Observation` and `Command` resources using SWE Common Text Encoding rules.

- Clause 16.4, Requirements Class “SWE Common Binary Encoding” defines requirements for encoding `Observation` and `Command` resources using SWE Common Binary Encoding rules.

The standardization target for these requirements classes is an implementation of the Web API.

There is no *Core* requirements class but an implementation target is expected to implement at least one of the *CS resource** types and one encoding.

The conformance classes corresponding to these requirements classes are presented in Annex A (normative). Conformance with this Standard shall be checked using all the relevant tests specified in Annex A. 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.

[*] *“CS resources” means “Connected Systems resources” and refers to all resource types defined in this Standard.*

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.

Policy SWG: OGC 08-131r3, *The Specification Model — Standard for Modular specifications*. Open Geospatial Consortium (2009). https://portal.ogc.org/files/?artifact_id=34762&version=2.

Alexandre Robin: OGC 23-001, *OGC API — Connected Systems — Part 1: Feature Resources, version 1.0*. Open Geospatial Consortium (2025). https://docs.ogc.org/is/23-001/23-001.html

Clemens Portele, Panagiotis (Peter) A. Vretanos, Charles Heazel: OGC 17-069r4, *OGC API — Features — Part 1: Core corrigendum*. Open Geospatial Consortium (2022). http://www.opengis.net/doc/IS/ogcapi-features-1/1.0.1.

OGC API — Features — Part 4: Create, Replace, Update and Delete, version 1.0.0-DRAFT. http://www.opengis.net/doc/IS/ogcapi-features-4/1.0

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

Semantic Sensor Network Ontology, (October 19 2017), https://www.w3.org/TR/vocab-ssn

Alexandre Robin: OGC 23-000, *OGC SensorML Encoding Standard, version 3.0*. Open Geospatial Consortium (2025). https://docs.ogc.org/is/23-000/23-000.html

Alexandre Robin: OGC 24-014, *OGC SWE Common Data Model Encoding Standard, version 3.0*. Open Geospatial Consortium (2025). https://docs.ogc.org/is/24-014/24-014.html

ISO: ISO 8601:2019, *Date and time — Representations for information interchange — Part 1: Basic rules. International Organization for Standardization, Geneva (2019). https://www.iso.org/standard/70907.html.*. ISO (2019).

ISO: ISO 8601:2019, *Date and time — Representations for information interchange — Part 2: Extensions. International Organization for Standardization, Geneva (2019). https://www.iso.org/standard/70908.html.*. ISO (2019).

T. Bray (ed.): IETF RFC 8259, *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC Publisher (2017). https://www.rfc-editor.org/info/rfc8259.

M. Nottingham: IETF RFC 8288, *Web Linking*. RFC Publisher (2017). https://www.rfc-editor.org/info/rfc8288.

JSON Schema Validation: A Vocabulary for Structural Validation of JSON, Version 2020-12, https://json-schema.org/draft/2020-12/json-schema-validation.html

The WebSocket Protocol, December 2011, Proposed Standard. https://www.rfc-editor.org/rfc/rfc6455

MQTT Version 5.0, 07 March 2019, OASIS Standard. http://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

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.

All terms defined in OGC API — Common — Part 1: Core, OGC API — Features — Part 1: Core and OGC API — Features — Part 4: Create, Replace, Update and Delete also apply.

4.1. Application Programming Interface (API)

A formally defined set of types and methods which establish a contract between client code which uses the API and implementation code which provides the API.

4.2. Actuation

An Actuation carries out an (Actuation) Procedure to change the state of the world using an Actuator.

[**SOURCE:** SOSA-SSN,]

4.3. Actuator

A device that is used by, or implements, an (Actuation) Procedure that changes the state of the world.

[**SOURCE:** SOSA-SSN,]

4.4. Command

A Command is a message that is sent to a System to trigger an Actuation. The Command message contains the parameters of the Actuation.

4.5. Connected Systems

Collections of interrelated systems consisting of information technology (IT) devices, sensors, actuators, platforms, and processes that can seamlessly interact.

4.6. Control Stream

A Control Stream is a collection of Commands targeted at the same System, and sharing the same controlled properties.

4.7. Data Stream

A Data Stream (or Datastream) is a collection of Observations acquired by the same System, and sharing the same observed properties.

4.8. Deployment

Describes the Deployment of one or more Systems for a particular purpose. Deployment may be done on a Platform.

[**SOURCE:** SOSA/SSN,]

4.9. Feature

Abstraction of real-world phenomena.

[**SOURCE:** ISO-19101, definition 4.11]

4.10. Feature Collection

A set of features from a dataset.

[**SOURCE:** OGC API — Features, definition 4.1.4]

4.11. Feature of Interest

The thing whose property is being estimated or calculated in the course of an Observation to arrive at a Result, or whose property is being manipulated by an Actuator, or which is being sampled or transformed in an act of Sampling.

[**SOURCE:** SOSA/SSN,]

4.12. Observation

Act of carrying out an (Observation) Procedure to estimate or calculate a value of a property of a Feature of Interest.

[**SOURCE:** SOSA/SSN,]

4.13. Platform

A Platform is an entity that hosts other entities, particularly Sensors, Actuators, Samplers, and other Platforms.

[**SOURCE:** SOSA/SSN,]

4.14. Procedure

A workflow, protocol, plan, algorithm, or computational method specifying how to make an Observation, create a Sample, or make a change to the state of the world (via an Actuator). A Procedure is re-usable, and might be involved in many Observations, Samplings, or Actuations. It explains the steps to be carried out to arrive at reproducible Results.

[**SOURCE:** SOSA/SSN,]

4.15. Property

Facet or attribute of an object referenced by a name.

Example : Abby's car has the color red, where "color red" is a property of the car instance

[**SOURCE:** ISO-19143]

4.16. Sample

Feature which is intended to be representative of a FeatureOfInterest on which Observations may be made.

[**SOURCE:** SOSA/SSN,]

4.17. Sampler

A device that is used by, or implements, a (Sampling) Procedure to create or transform one or more samples.

[**SOURCE:** SOSA/SSN,]

4.18. **Sampling Feature**

Feature representing a subset of a FeatureOfInterest on which properties are observed or controlled. For Observations, Sampling Feature is a synonym of Sample.

4.19. **Sensor**

Device, agent (including humans), or software (simulation) involved in, or implementing, a Procedure. Sensors respond to a Stimulus, e.g., a change in the environment, or Input data composed from the Results of prior Observations, and generate a Result. Sensors can be hosted by Platforms.

[**SOURCE:** SOSA/SSN,]

4.20. **System**

System is a unit of abstraction for pieces of infrastructure that implement Procedures. A System may have components, its subsystems, which are other Systems.

[**SOURCE:** SOSA/SSN,]

6 CONVENTIONS

6 CONVENTIONS

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

6.1. Identifiers

The normative provisions in this standard are denoted by the URI

`http://www.opengis.net/spec/ogcapi-connectedsystems-2/1.0`

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

6.2. Abbreviated terms

In this document the following abbreviations and acronyms are used or introduced:

- API: Application Programming Interface
- CRS: Coordinate Reference System
- CSML: Climate Science Modeling Language
- CSV: Comma-Separated Values
- DSV: Delimiter-Separated Values (a generalization of CSV)
- GPS: Global Positioning System
- ISO: International Organization for Standardization
- MISB: Motion Imagery Standards Board
- OGC: Open Geospatial Consortium
- SAS: Sensor Alert Service
- SensorML: Sensor Model Language
- SI: Système International (International System of Units)
- SOS: Sensor Observation Service

- SPS: Sensor Planning Service
- SWE: Sensor Web Enablement
- TAI: Temps Atomique International (International Atomic Time)
- UML: Unified Modeling Language
- UTC: Coordinated Universal Time
- XML: eXtensible Markup Language
- 1D: One Dimensional
- 2D: Two Dimensional
- 3D: Three Dimensional

7 OVERVIEW

7 OVERVIEW

7.1. General

All resources defined in Part 1 of this Standard are feature resources, among which the `System` resource. Part 2 (this document) defines additional resource types to describe and interact with the dynamic data that flows in and out of these systems.

Part 2 of this Standard defines resource types that allow the provision of dynamic data about all kinds of devices, hardware components or processes that can transmit and/or receive data via communication networks (a.k.a. connected systems), including sensors, platforms, robots, human observers, forecast models, computer simulations, etc.

Flows carrying observation and status data coming out of a system are called `Datastreams` while flows carrying commands sent to a system are called `Control Streams` (note that the direction of the flow mentioned here is relative to the real system, which is different from the direction of the data flows going in and out of the API server).

7.2. Design Considerations

In this Standard, `Observations` and `Commands` are purposefully not modelled as `Features`. This choice was made to keep a clear separation between the `Features of Interest` that represent concrete or virtual objects (or things) of interest (and in the vast majority of use cases, real-world objects) and the other concepts that are used to encapsulate dynamic data related to these features:

- `Observations` carry the result of the estimation of one or more feature properties, at a given time (and location); and
- `Commands` carry the desired value of one or more feature properties, at a given time.

Likewise, `DataStreams` and `ControlStreams` are not modelled as features, as they are simply containers for `Observations` and `Commands`, respectively. More specifically, they are particular cases of homogeneous collections that are associated to a single `System` (see Clauses 9 and 10).

7.3. Resource Types

As indicated above, while part 1 of this Standard focused on defining “static” feature types, part 2 defines additional resources to deal with dynamic data associated to these features.

Figure 1 shows a UML class diagram of all Connected Systems API resources. Resources defined in part 2 are shown with a solid border while resources that were already defined in part 1 are shown with a dashed light gray outline.

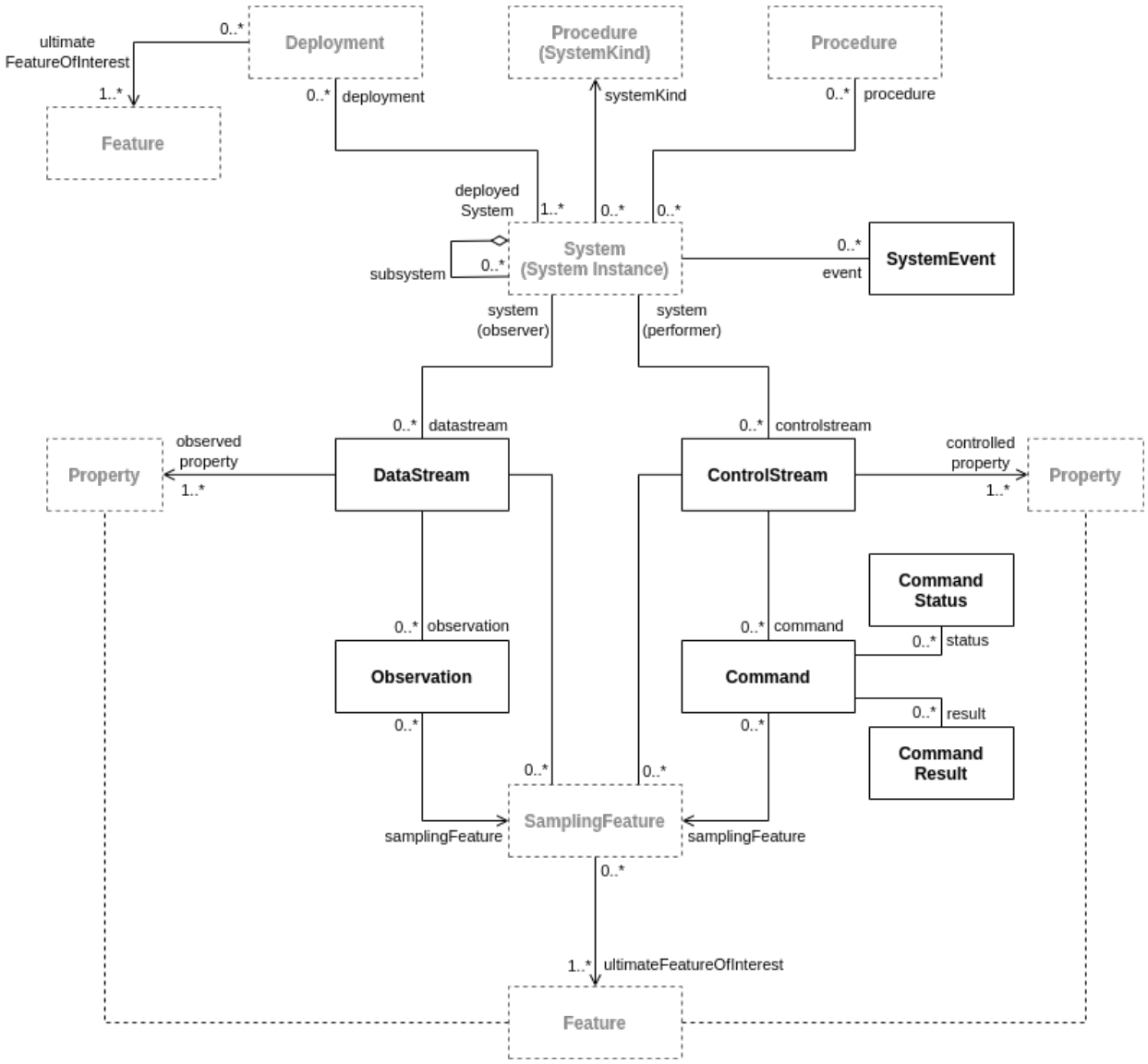

Figure 1 — Class diagram of API resources

The table below provides a short summary description of these resource types:

Table 1 — Overview of resource types defined by this Standard

RESOURCE TYPE	REQUIREMENTS CLASS	DESCRIPTION	POSSIBLE ENCODINGS
DataStream	Clause 9	Description of datastreams, including observed properties and features of interest.	JSON
Observation	Clause 9	Actual observations, including the result data.	JSON, SWE-JSON, SWE-Text, SWE-Binary
ControlStream	Clause 10	Description of control channels, including controllable properties and features of interest.	JSON
Command	Clause 10	Actual command messages, including the command parameters data.	JSON, SWE-JSON, SWE-Text, SWE-Binary
Command Status	Clause 10	Status info about a given command.	JSON
Command Result	Clause 10	Result of a given command.	JSON
Feasibility	Clause 11	Feasibility requests, including the command parameters data.	JSON, SWE-JSON, SWE-Text, SWE-Binary
Feasibility Status	Clause 11	Status info about a given feasibility request.	JSON
Feasibility Result	Clause 11	Result of a given feasibility request.	JSON
System Event	Clause 12	System events (e.g., deployment, maintenance or replacement events).	JSON

NOTE: The encodings listed in the table are the ones defined in this Standard document but extensions can define additional encodings.

7.4. API Endpoints

OGC API — Connected Systems — Part 1 defines the concept of canonical **resources endpoint** and **canonical resource endpoint**. This section provides the canonical endpoints used in CS API Part 2.

7.4.1. Canonical Resources Endpoints

The canonical resources endpoints for resource types defined in Part 2 of the CS API Standard are:

- `{api_root}/datastreams`

- {api_root}/observations
- {api_root}/controlstreams
- {api_root}/commands
- {api_root}/feasibility
- {api_root}/systemEvents

7.4.2. Canonical Resource Endpoints

The canonical URL templates to access a single resource defined in Part 2 of the CS API Standard are:

- {api_root}/datastreams/{id}
- {api_root}/observations/{id}
- {api_root}/controlstreams/{id}
- {api_root}/commands/{id}
- {api_root}/feasibility/{id}
- {api_root}/systemEvents/{id}

8

REQUIREMENTS CLASS "COMMON"

8 REQUIREMENTS CLASS "COMMON"

REQUIREMENTS CLASS 1	
IDENTIFIER	/req/api-common
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.1: /conf/api-common
PREREQUISITE	http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/req/api-common
NORMATIVE STATEMENTS	Requirement 1: /req/api-common/resources Requirement 2: /req/api-common/resource-collection

8.1. Overview

All resource types defined in this Standard must comply with a common set of rules, many of which are inherited from OGC API — Common.

8.2. Non-feature Resources

Resources defined in this Standard are not considered features, but many of the requirements specified in OGC API — Features — Part 1: Core and OGC API — Features — Part 4: Create, Replace, Update and Delete still apply.

REQUIREMENT 1	
IDENTIFIER	/req/api-common/resources
INCLUDED IN	Requirements class 1: /req/api-common
A	All references to the term "features" or "feature" in the OGC API — Features — Part 1: Core and OGC API — Features — Part 4: Create, Replace, Update and Delete Standards SHALL be replaced by the

REQUIREMENT 1	
	terms “resources” or “resource” when interpreting requirements for OGC API — Connected Systems — Part 2.

8.3. Resource Collections

Resource collections are exposed in the same way feature collections are in OGC API — Features — Part 1: Core, except that the `itemType` is set to a different value.

REQUIREMENT 2	
IDENTIFIER	`/req/api-common/resource-collection`
INCLUDED IN	Requirements class 1: `/req/api-common`
A	A resource collection SHALL fulfill all requirements from Clauses 7.14, 7.15 and 7.16 of OGC API — Features — Part 1: Core, except for clauses 7.15.3 (parameter `bbox`) and 7.15.4 (parameter `datetime`).
B	All references to the term “features” or “feature” in these requirements SHALL be replaced by the terms “resources” or “resource”, respectively.

9

REQUIREMENTS CLASS "DATASTREAMS & OBSERVATIONS"

9 REQUIREMENTS CLASS "DATASTREAMS & OBSERVATIONS"

9.1. Overview

REQUIREMENTS CLASS 2	
IDENTIFIER	/req/datastream
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.2: /conf/datastream
PREREQUISITE	Requirements class 1: /req/api-common
NORMATIVE STATEMENTS	Requirement 3: /req/datastream/sf-ref-from-datastream Requirement 4: /req/datastream/foi-ref-from-datastream Requirement 5: /req/datastream/canonical-url Requirement 6: /req/datastream/resources-endpoint Requirement 7: /req/datastream/canonical-endpoint Requirement 8: /req/datastream/ref-from-system Requirement 9: /req/datastream/ref-from-deployment Requirement 10: /req/datastream/collections Requirement 11: /req/datastream/schema-op Requirement 12: /req/datastream/obs-canonical-url Requirement 13: /req/datastream/obs-resources-endpoint Requirement 14: /req/datastream/obs-canonical-endpoint Requirement 15: /req/datastream/obs-ref-from-datastream

REQUIREMENTS CLASS 2

Requirement 16: `/req/datastream/obs-collections`

The “Datastreams & Observations” requirements class specifies how `DataStream` and `Observation` resources are provided using the CS API.

A `DataStream` resource represents data produced by a single output of an (observation) system (itself represented by a `System` feature in the CS API). The `DataStream` can be used to provide access to real-time data only, archived data only, or both. The metadata in the `DataStream` description can be used to disambiguate between these cases.

`DataStream` and `Observation` resources implement the *ObservationCollection* and *Observation* concepts defined in the Semantic Sensor Network Ontology (SOSA/SSN), respectively.

9.2. DataStream Resource

9.2.1. Introduction

In the CS API Standard, `DataStream` resources are a special kind of resource that implements the *sosa:ObservationCollection* concept, with the following restrictions:

- All observations in a `DataStream` are produced by the same `System`; and
- All observations in a `DataStream` share the same observed properties and the same result schema.

This section defines the attributes and associations composing a `DataStream` resource, but the exact way they are encoded in the payload is defined by each encoding. For encodings defined in this document, please see:

- DataStream resource encoded in JSON

Below is the contextual class diagram of the `DataStream` resource:

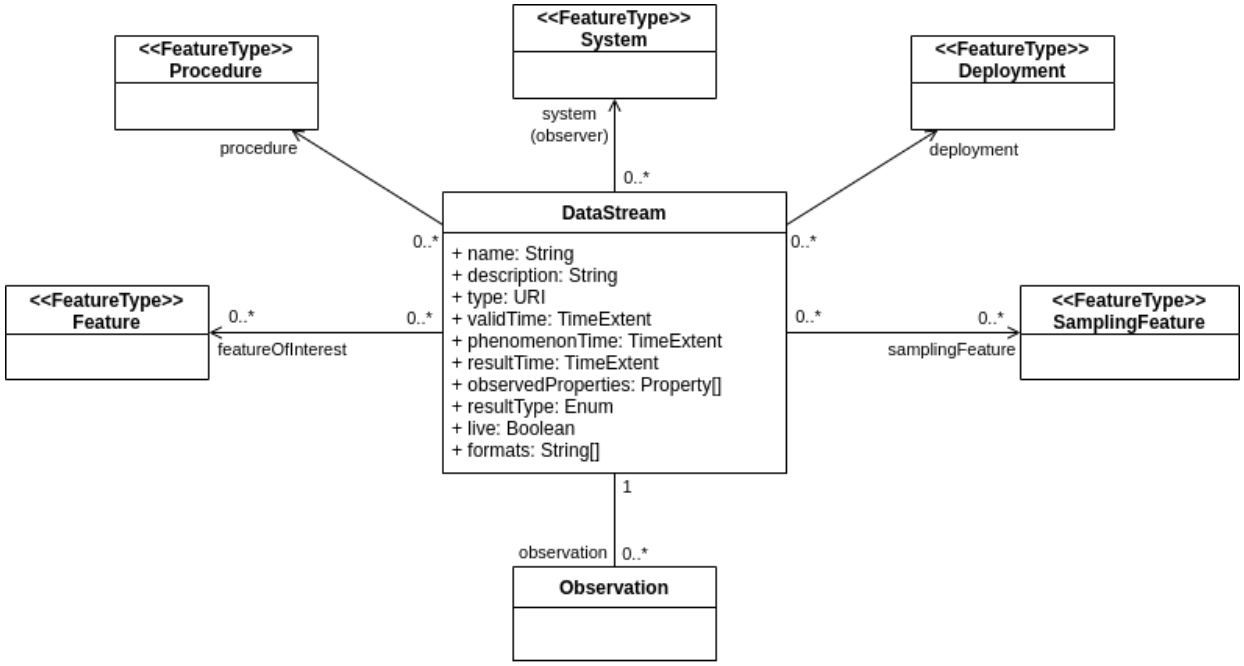

Figure 2 — DataStream Resource Diagram

9.2.2. Properties

The following tables describe the attributes and associations of a `DataStream` resource and their mapping to SOSA/SSN.

Table 2 — DataStream Attributes

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
`name`	`rdfs:label`	The human readable name of the datastream.	String	Mandatory
`description`	`rdfs:comment`	A human readable description for the datastream.	String	Optional
`type`	-	The type of datastream (see Table 3).	Enum	Optional
`validTime`	`sosa:validTime`	The validity period of the datastream's description.	TimeExtent	Optional
`phenomenonTime`	`sosa:phenomenonTime`	The time period spanned by the phenomenon times of all observations in the datastream.	TimeExtent	Required
`resultTime`	`sosa:resultTime`	The time period spanned by the result times of all observations in the datastream.	TimeExtent	Required

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
observedProperties	sosa: observedProperty	Properties for which the observations in the datastream provide measurements.	List<URI>	Required
resultType	-	The type of result for observations in the datastream (see Table 4).	Enum	Required
live	-	Indicates whether live data is available from the datastream.	Boolean	Required
formats	-	The list of formats that the observations in the datastream can be encoded to.	List <String>	Required

The values for the properties observedProperties, phenomenonTime, resultTime, resultType SHALL be automatically generated by the server based the Observations that are linked to the Datastream. If there are no linked Observations the properties SHALL be set to null. The property live MAY be generated by the server. In this case the server MAY ignore updates to the property.

Table 3 — DataStream Types

DATASTREAM TYPE	USAGE
status	For datastreams providing status observations of the parent system itself or one of its subsystems.
observation	For datastreams providing observations of other features of interest (not the system itself).

Table 4 — Result Types

RESULT TYPE	USAGE
measure	When the result is a single scalar value with a unit of measure
vector	When the result is a vector quantity (e.g velocity vector, stress tensor)
record	When the result is a record containing only scalar values and/or vectors
coverage	When the result is a coverage (any number of dimensions)
complex	When the result is a record with nested records and/or arrays

Table 5 — DataStream Associations

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
`system`	`sosa:isObservedBy`	The `System` that is the producer of the datastream.	A single `System` resource (by reference).	Required
`observations`	`sosa:hasMember`	The `Observations` that are part of the datastream.	A list of `Observation` resources.	Required
`procedure`	`sosa:usedProcedure`	The procedure used to generate observations in the datastream.	A single `Procedure` resource.	Optional
`deployment`	-	The deployment during which the datastream was generated.	A single `Deployment` resource.	Optional
`samplingFeatures`	`sosa:hasFeatureOfInterest`	The `Sampling Features` that are the subject of observations in the datastream.	A list of `SamplingFeature` resources.	Optional
`featuresOfInterest`	`sosa: hasUltimateFeatureOfInterest`	The ultimate features of interest that are the subject of observations in the datastream.	A list of `Feature` resources.	Optional

When sampling features and ultimate features of interest are hosted on the same server, they are made accessible through sub-endpoints of the `DataStream` resource.

REQUIREMENT 3	
IDENTIFIER	/req/datastream/sf-ref-from-datastream
INCLUDED IN	Requirements class 2: /req/datastream
CONDITIONS	The server implements http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/req/sampling
A	The server SHALL implement a Sampling Feature resources endpoint at path {api_root}/datastreams/{dsId}/samplingFeatures for each available DataStream resource.
B	The endpoint SHALL only expose the Sampling Feature resources that are associated to observations in the parent DataStream with ID dsId.

REQUIREMENT 4	
IDENTIFIER	/req/datastream/foi-ref-from-datastream
INCLUDED IN	Requirements class 2: /req/datastream
CONDITIONS	• The server provides the featuresOfInterest association as part of DataStream resource representations. • The server hosts the features of interest descriptions locally.
A	The server SHALL implement a Feature resources endpoint at path {api_root}/datastreams/{dsId}/featuresOfInterest for each available DataStream resource.
B	The endpoint SHALL only expose the Feature resources that are the ultimate features of interest of observations in the parent DataStream with ID dsId.

9.3. DataStream Canonical URL

The CS API Standard requires that every DataStream resource has a canonical URL.

REQUIREMENT 5	
IDENTIFIER	/req/datastream/canonical-url
INCLUDED IN	Requirements class 2: /req/datastream

REQUIREMENT 5	
A	All `DataStream` resources exposed by the server SHALL be accessible through their canonical URL of the form `{api_root}/datastreams/{id}` where `id` is the local identifier of the `DataStream` resource.
B	If a `DataStream` resource is retrieved through any other URL than its canonical URL, the server response SHALL include a link to its canonical URL with relation type `canonical`.

9.4. DataStream Resources Endpoints

9.4.1. Definition

A `DataStream` resources endpoint is an endpoint exposing a set of `DataStream` resources that can be further filtered using query parameters.

REQUIREMENT 6	
IDENTIFIER	`/req/datastream/resources-endpoint`
INCLUDED IN	Requirements class 2: `/req/datastream`
A	The server SHALL support the HTTP GET operation at the path associated to the `DataStream` resources endpoint.
B	The operation SHALL support the parameter `limit` defined in Clause 7.15.2 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in OGC API — Features — Part 1: Core requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	The operation SHALL support the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in OGC API — Features — Part 1: Core requirements SHALL be replaced by the terms “resources” or “resource”, respectively. Only `DataStream` resources that have a `validTime` property that intersects the temporal information in the `datetime` parameter SHALL be part of the result set.
D	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The response SHALL only include the DataStream resources selected by the request.
E	Error cases SHALL be reported using HTTP status codes listed in Clause 7.5.1 of OGC API — Features — Part 1: Core.

Clause 13.2 defines additional query parameters applicable to `DataStream` resources endpoint.

9.4.2. Canonical DataStream Resources Endpoint

The CS API Standard requires that a canonical `DataStream` resources endpoint, exposing all `DataStream` resources, be made available by the server.

REQUIREMENT 7	
IDENTIFIER	`/req/datastream/canonical-endpoint`
INCLUDED IN	Requirements class 2: `/req/datastream`
A	The server SHALL expose a DataStream resources endpoint at the path `{api_root}/datastreams`.
B	The endpoint SHALL expose all `DataStream` resources available on the server.

9.4.3. Nested DataStream Resources Endpoints

The set of datastreams produced by a specific system is available at a nested endpoint under the corresponding `System` resource:

REQUIREMENT 8	
IDENTIFIER	`/req/datastream/ref-from-system`
INCLUDED IN	Requirements class 2: `/req/datastream`
CONDITIONS	The server implements `http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/req/system`
A	The server SHALL implement a DataStream resources endpoint at path `{api_root}/systems/{sysId}/datastreams` for each available `System` resource.
B	The endpoint SHALL only expose the `DataStream` resources associated to the `System` with ID `sysId`.

The set of datastreams associated to a specific deployment is available at a nested endpoint under the corresponding `Deployment` resource:

REQUIREMENT 9	
IDENTIFIER	`/req/datastream/ref-from-deployment`

REQUIREMENT 9	
INCLUDED IN	Requirements class 2: `/req/datastream`
CONDITIONS	The server implements `http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/req/deployment`
A	The server SHALL implement a DataStream resources endpoint at path `{api_root}/deployments/{depId}/datastreams` for each available `Deployment` resource.
B	The endpoint SHALL only expose the `DataStream` resources associated to a system that was deployed during the `Deployment` with ID `depId`, and whose valid time intersects the deployment time period.

9.5. DataStream Collections

Any number of resources collections containing `DataStream` resources can be available at a CS API endpoint. `DataStream` collections are identified with the item type `DataStream`.

`DataStream` resources can be grouped into collections according to any arbitrary criteria, as exemplified below:

Example: Examples of DataStream Collections

- All data collected by organization X at URL `{api_root}/collections/orgx_datastreams`
- All data collected during a field survey involving multiple sensors at URL `{api_root}/collections/campaignX_datastreams`

Note that a given datastream can be part of multiple collections at the same time.

REQUIREMENT 10	
IDENTIFIER	`/req/datastream/collections`
INCLUDED IN	Requirements class 2: `/req/datastream`
A	If the server exposes collections of `DataStream` resources, it SHALL be done as specified in Clause 8.3.
B	The server SHALL identify all resource collections containing `DataStream` resources by setting the `itemType` attribute to `DataStream` in the Collection metadata.
C	For any resource collection with `itemType` set to `DataStream`, the HTTP GET operation at the path `/collections/{collectionId}/items` SHALL support the query parameters and response of a DataStream resources endpoint.

9.6. Observation Schemas

A different observation schema is needed for each individual datastream because the exact content of the observations result changes according to result type and the properties being observed. Moreover, multiple observation formats can be offered for any given datastream, and the schema is typically expressed differently for each format.

Thus, for each `DataStream` resource, the CS API provides a way for the server to communicate an observation schema (*not necessarily a JSON schema*) for each supported observation format. The exact content of a schema resource is defined by each encoding. See section Clause 16.1.4 for example schemas used for observations encoded using the default JSON format.

Extensions to this Standard can define additional representation formats for observations. Such format extensions must clearly define the mapping between elements of the representation format and the Observation resource defined in the next clause.

REQUIREMENT 11	
IDENTIFIER	`/req/datastream/schema-op`
INCLUDED IN	Requirements class 2: `/req/datastream`
A	For every `DataStream` resource exposed at the CS API endpoint, the server SHALL support the HTTP GET operation at the path `{api_root}/datastreams/{id}/schema`.

B	The operation SHALL support the parameter `obsFormat` with the following characteristics (using an OpenAPI 3.0 fragment):
	`name: obsFormat`
	`in: query`
	`description: Media type of the desired observation format`
	`required: true`
	`schema:`
	`  type: string`

C	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The server SHALL return a single schema corresponding to the format identified by the `obsFormat` parameter.
D	Error cases SHALL be reported using HTTP status codes listed in Clause 7.5.1 of OGC API — Features — Part 1: Core.

Example: Example
If a datastream with ID `{id}` reports the following supported observation formats:

- `application/json`
- `application/swe+csv`
- `application/swe+binary`

The schema for each of these formats is obtained with the following requests, respectively:

- `https://{api_root}/datastreams/{id}/schema?obsFormat=application/json`
- `https://{api_root}/datastreams/{id}/schema?obsFormat=application/swe%2Bcsv`
- `https://{api_root}/datastreams/{id}/schema?obsFormat=application/swe%2Bbinary`

Note that the media type in the request has to be properly URL encoded, leading to the `%2B` in place of the `+` character.

9.7. Observation Resource

9.7.1. Introduction

In the CS API Standard, `Observation` resources are a special kind of resource that implements the *sosa:Observation* concept.

In the CS API, `Observation` resources are always associated to a `DataStream` (e.g., a type of *ObservationCollection*). Some properties of the observation (e.g., link to the observing system, observed properties) can thus be omitted as they are provided at the datastream level.

In addition, the CS API does not restrict `Observation` resources to have a single observed property. It is thus possible to package the observation result of several properties in a single resource.

This section defines the attributes and associations composing a `Observation` resource, but the exact way they are encoded in the payload is defined by each encoding. For encodings defined in this document, please see:

- Observation resource encoded in JSON

Below is the contextual class diagram of the `Observation` resource:

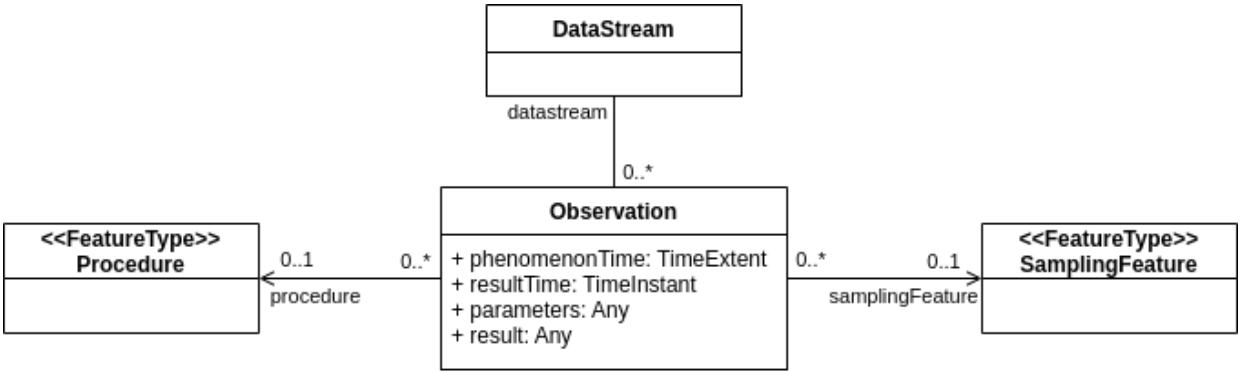

Figure 3 — Observation Resource Diagram

9.7.2. Properties

The following tables describe the attributes and associations of an `Observation` resource and their mapping to SOSA/SSN.

Table 6 — Observation Attributes

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
`phenomenonTime`	`sosa:phenomenonTime`	The time the observed property value applies to the feature of interest.	DateTime	Required
`resultTime`	`sosa:resultTime`	The time the result value was obtained.	DateTime	Required
`parameters`	-	Observation parameters, providing information about how the procedure was used to produce this specific observation.	Any	Optional
`result`	`sosa:hasResult`	Observation result, carrying the estimated values of the observed properties.	Any	Required

NOTE: The `phenomenonTime` can be in the far past (e.g., geological or deep space observations) or in the future (e.g., weather forecast). The `resultTime`, however, can never be in the future.

Table 7 — Observation Associations

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
`datastream`	-	The `DataStream` that the observation is part of.	A single `DataStream` resource.	Required

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
`samplingFeature`	`sosa:hasFeatureOfInterest`	The sampling feature that is the subject of the observation.	A single `SamplingFeature` resource.	Optional
`procedure`	`sosa:usedProcedure`	The procedure that was used to make the observation	A single `Procedure` resource.	Optional

9.8. Observation Canonical URL

The CS API Standard requires that every `Observation` resource has a canonical URL.

REQUIREMENT 12	
IDENTIFIER	`/req/datastream/obs-canonical-url`
INCLUDED IN	Requirements class 2: `/req/datastream`
A	All `Observation` resources exposed by the server SHALL be accessible through their canonical URL of the form `{api_root}/observations/{id}` where `id` is the local identifier of the `Observation` resource.
B	If a `Observation` resource is retrieved through any other URL than its canonical URL, the server response SHALL include a link to its canonical URL with relation type `canonical`.

9.9. Observation Resources Endpoint

9.9.1. Definition

An `Observation` resources endpoint is an endpoint exposing a set of `Observation` resources that can be further filtered using query parameters.

REQUIREMENT 13	
IDENTIFIER	`/req/datastream/obs-resources-endpoint`

REQUIREMENT 13	
INCLUDED IN	Requirements class 2: `/req/datastream`
A	The server SHALL support the HTTP GET operation at the path associated to the `Observation` resources endpoint.
B	The operation SHALL support the parameter `limit` defined in Clause 7.15.2 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in OGC API — Features — Part 1: Core requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The response SHALL only include the Observation resources selected by the request.
D	Error cases SHALL be reported using HTTP status codes listed in Clause 7.5.1 of OGC API — Features — Part 1: Core.

9.9.2. Canonical Observation Resources Endpoint

The CS API Standard requires that a canonical `Observation` resources endpoint, exposing all `Observation` resources, be made available by the server.

REQUIREMENT 14	
IDENTIFIER	`/req/datastream/obs-canonical-endpoint`
INCLUDED IN	Requirements class 2: `/req/datastream`
A	The server SHALL expose a Observation resources endpoint at the path `{api_root}/observations`.
B	The endpoint SHALL expose all `Observation` resources available on the server.

9.9.3. Nested Observation Resources Endpoint

The set of observations produced as part of a specific datastream is available at a nested endpoint under the corresponding `DataStream` resource:

REQUIREMENT 15	
IDENTIFIER	`/req/datastream/obs-ref-from-datastream`
INCLUDED IN	Requirements class 2: `/req/datastream`

REQUIREMENT 15	
A	The server SHALL implement a Observation resources endpoint at path `{api_root}/datastreams/{dsId}/observations` for each available `DataStream` resource.
B	The endpoint SHALL only expose the `Observation` resources that are part of the parent `DataStream` with ID `dsId`.

9.10. Observation Collections

Any number of resources collections containing `Observation` resources can be available at a CS API endpoint. `Observation` collections are identified with the item type `Observation`.

`Observation` resources can be grouped into collections according to any arbitrary criteria, as exemplified below:

Example: Examples of Observation Collections

- All observations collected by organization X at URL `{api_root}/collections/orgx_obs`
- All observations collected during a field survey involving multiple sensors at URL `{api_root}/collections/campaignX_obs`

Note that a given observation can be part of multiple collections at the same time.

REQUIREMENT 16	
IDENTIFIER	`/req/datastream/obs-collections`
INCLUDED IN	Requirements class 2: `/req/datastream`
A	If the server exposes collections of `Observation` resources, it SHALL be done as specified in Clause 8.3.
B	The server SHALL identify all resource collections containing `Observation` resources by setting the `itemType` attribute to `Observation` in the Collection metadata.
C	For any resource collection with `itemType` set to `Observation`, the HTTP GET operation at the path `/collections/{collectionId}/items` SHALL support the query parameters and response of a Observation resources endpoint.

10

REQUIREMENTS CLASS "CONTROL STREAMS & COMMANDS"

10 REQUIREMENTS CLASS "CONTROL STREAMS & COMMANDS"

10.1. Overview

REQUIREMENTS CLASS 3	
IDENTIFIER	/req/controlstream
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.3: /conf/controlstream
PREREQUISITE	Requirements class 1: /req/api-common
NORMATIVE STATEMENTS	Requirement 17: /req/controlstream/sf-ref-from-controlstream Requirement 18: /req/controlstream/foi-ref-from-controlstream Requirement 19: /req/controlstream/canonical-url Requirement 20: /req/controlstream/resources-endpoint Requirement 21: /req/controlstream/canonical-endpoint Requirement 22: /req/controlstream/ref-from-system Requirement 23: /req/controlstream/ref-from-deployment Requirement 24: /req/controlstream/collections Requirement 25: /req/controlstream/schema-op Requirement 26: /req/controlstream/cmd-canonical-url Requirement 27: /req/controlstream/cmd-resources-endpoint Requirement 28: /req/controlstream/cmd-canonical-endpoint Requirement 29: /req/controlstream/cmd-ref-from-controlstream

REQUIREMENTS CLASS 3	
	Requirement 30: `/req/controlstream/cmd-collections` Requirement 31: `/req/controlstream/status-resources-endpoint` Requirement 32: `/req/controlstream/command-status-endpoint` Requirement 33: `/req/controlstream/result-resources-endpoint` Requirement 34: `/req/controlstream/command-result-endpoint`

This requirements class specifies how `ControlStream`, `Command`, and `CommandStatus` resources are provided using the CS API.

A `ControlStream` resource represents a control channel that is used to change the state of (or affect) a feature of interest (which can be a `System` itself). The state is changed by sending the desired values of certain controllable properties of the feature of interest, but note that the resulting state will not necessarily reflect the exact values requested (If the exact result state must be known, it can be monitored separately using an associated `DataStream` resource).

A `ControlStream` resource represents the real-time stream of command messages sent to the system, as well as all historical commands received through the channel. It can be used to provide access to real-time commands only, archived commands only, or both. The metadata in the `ControlStream` description can used to disambiguate between these cases.

`Command` resources are available through their parent `ControlStream` resource, and each command can lead to the creation of one or more status reports (i.e., `CommandStatus` resources).

`ControlStream` and `Command` resources implement the *ActuationCollection* and *Actuation* concepts defined in the Semantic Sensor Network Ontology (SOSA/SSN), respectively.

10.2. ControlStream Resource

10.2.1. Introduction

In the CS API Standard, `ControlStream` resources are a special kind of resource that implements the *sosa:ActuationCollection* concept, with the following restrictions:

- All commands in a `ControlStream` are received by the same `System`; and
- All commands in a `ControlStream` share the same controlled properties and the same parameter schema.

This section defines the attributes and associations composing a `ControlStream` resource, but the exact way they are encoded in the payload is defined by each encoding. For encodings defined in this document, please see:

- ControlStream resource encoded in JSON

Below is the contextual class diagram of the `ControlStream` resource:

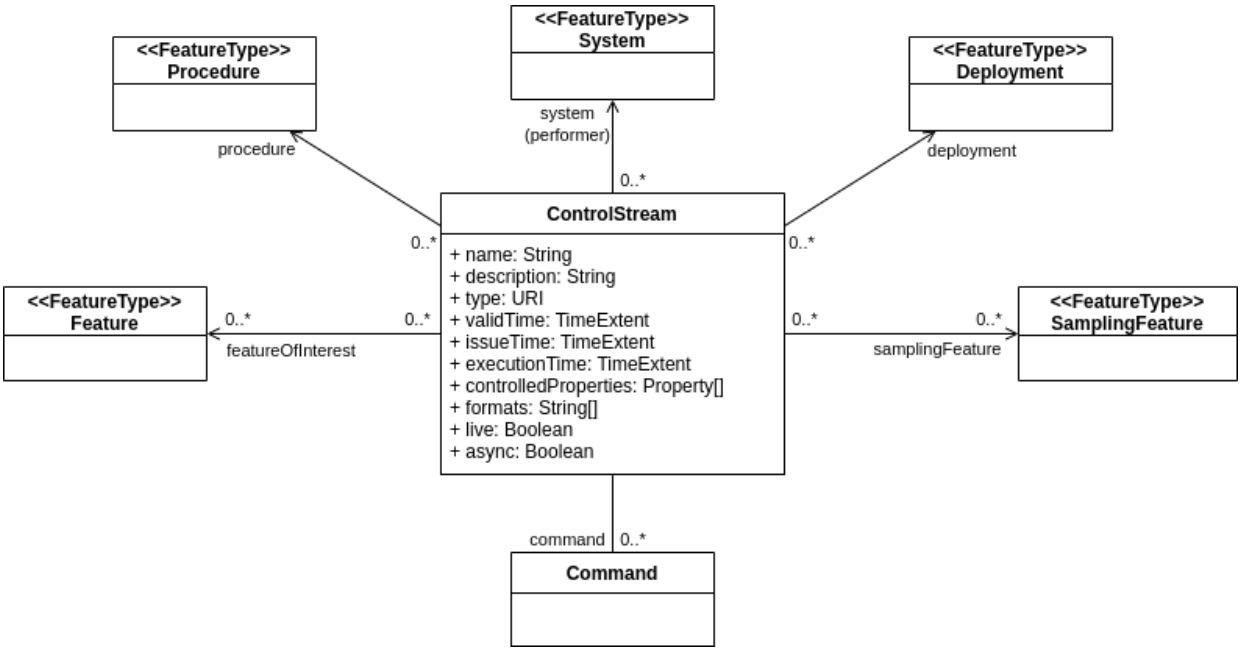

Figure 4 — ControlStream Resource Diagram

10.2.2. Properties

The following tables describe the attributes and associations of the `ControlStream` resource and their mapping to SOSA/SSN.

Table 8 — ControlStream Attributes

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
`name`	`rdfs:label`	The human readable name of the control stream.	String	Required
`description`	`rdfs:comment`	A human readable description for the control stream.	String	Optional
`type`	-	The type of control stream (see Table 9).	Enum	Optional
`validTime`	`sosa:validTime`	The validity period of the control stream's description.	TimeExtent	Optional

NAME	SOSA/SSN PROPERTY	DEFINITION	DATA TYPE	USAGE
`issueTime`	-	The time period spanned by the issue times of all commands in the control stream.	TimeExtent	Required
`executionTime`	`sosa:phenomenonTime`	The time period spanned by the execution times of all commands in the control stream.	TimeExtent	Required
`controlledProperties`	`sosa:actsOnProperty`	Properties whose value can be changed by commands in the control stream.	List<URI>	Required
`live`	-	Indicates whether the control stream currently accepts commands.	Boolean	Required
`async`	-	Indicates whether commands are processed asynchronously in the control stream.	Boolean	Required
`formats`	-	The list of formats that the commands in the control stream can be encoded to.	List <String>	Required

Table 9 — ControlStream Types

TYPE	USAGE
self	For control streams that affect the parent system itself or one of its subsystems.
external	For control streams that affect external features of interest.

Table 10 — ControlStream Associations

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
`system`	`sosa:madeByActuator`	The `System` that received commands from the `ControlStrea`	A single `System` resource.	Required
`commands`	`sosa:hasMember`	The `Commands` that were sent to the control channel.	A list of `Command` resources.	Required
`procedure`	`sosa:usedProcedure`	The procedure used to	A single `Procedure` resource.	Optional

NAME	SOSA/SSN PROPERTY	DEFINITION	TARGET CONTENT	USAGE
		process commands received in the control stream.		
`deployment`	-	The deployment during which the control stream was used.	A single `Deployment` resource.	Optional
`samplingFeatures`	`sosa:hasFeatureOfInterest`	The `Sampling Features` that are the target of commands in this control stream.	A list of `SamplingFeature` resources.	Optional
`featuresOfInterest`	`sosa: hasUltimateFeatureOfInterest`	The ultimate features of interest that are affected by the commands in this control stream.	A list of `Feature` resources.	Optional

NOTE: In the case of commands/actuations, the `sampling feature` is used to describe where the effector interacts with the feature of interest (e.g., the vent of an A/C system, the part of a larger system, etc.).

When sampling features and ultimate features of interest are hosted on the same server, they are made accessible through sub-endpoints of the `DataStream` resource.

REQUIREMENT 17	
IDENTIFIER	`/req/controlstream/sf-ref-from-controlstream`
INCLUDED IN	Requirements class 3: `/req/controlstream`
CONDITIONS	The server implements `http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/req/sampling`
A	The server SHALL implement a Sampling Feature resources endpoint at path `{api_root}/controlstreams/{dsId}/samplingFeatures` for each available `DataStream` resource.

REQUIREMENT 17	
B	The endpoint SHALL only expose the `Sampling Feature` resources that are associated to observations in the parent `DataStream` with ID `dsId`.

REQUIREMENT 18	
IDENTIFIER	`/req/controlstream/foi-ref-from-controlstream`
INCLUDED IN	Requirements class 3: `/req/controlstream`
CONDITIONS	• The server provides the `featuresOfInterest` association as part of `ControlStream` resource representations. • The server hosts the features of interest descriptions locally.
A	The server SHALL implement a Feature resources endpoint at path `{api_root}/controlstreams/{dsId}/featuresOfInterest` for each available `ControlStream` resource.
B	The endpoint SHALL only expose the `Feature` resources that are the ultimate features of interest of commands in the parent `ControlStream` with ID `dsId`.

10.3. ControlStream Canonical URL

The CS API Standard requires that every `ControlStream` resource has a canonical URL.

REQUIREMENT 19	
IDENTIFIER	`/req/controlstream/canonical-url`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	All `ControlStream` resources exposed by the server SHALL be accessible through their canonical URL of the form `{api_root}/controls/{id}` where `id` is the local identifier of the `ControlStream` resource.
B	If a `ControlStream` resource is retrieved through any other URL than its canonical URL, the server response SHALL include a link to its canonical URL with relation type `canonical`.

10.4. ControlStream Resources Endpoints

10.4.1. Definition

A `DataStream` resources endpoint is an endpoint exposing a set of `ControlStream` resources that can be further filtered using query parameters.

REQUIREMENT 20	
IDENTIFIER	`/req/controlstream/resources-endpoint`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	The server SHALL support the HTTP GET operation at the path associated to the `ControlStream` resources endpoint.
B	The operation SHALL support the parameter `limit` defined in Clause 7.15.2 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in OGC API — Features — Part 1: Core requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	The operation SHALL support the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in OGC API — Features — Part 1: Core requirements SHALL be replaced by the terms “resources” or “resource”, respectively. Only `DataStream` resources that have a `validTime` property that intersects the temporal information in the `datetime` parameter SHALL be part of the result set.
D	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The response SHALL only include the ControlStream resources selected by the request.
E	Error cases SHALL be reported using HTTP status codes listed in Clause 7.5.1 of OGC API — Features — Part 1: Core.

Clause 13.2 defines additional query parameters applicable to `ControlStream` resources endpoint.

10.4.2. Canonical ControlStream Resources Endpoint

The CS API Standard requires that a canonical `ControlStream` resources endpoint, exposing all `ControlStream` resources, be made available by the server.

REQUIREMENT 21	
IDENTIFIER	`/req/controlstream/canonical-endpoint`

REQUIREMENT 21	
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	The server SHALL expose a ControlStream resources endpoint at the path `{api_root}/controlstreams`.
B	The endpoint SHALL expose all `ControlStream` resources available on the server.

10.4.3. Nested ControlStream Resources Endpoints

The set of control streams available on a specific system is available at a nested endpoint under the corresponding `System` resource:

REQUIREMENT 22	
IDENTIFIER	`/req/controlstream/ref-from-system`
INCLUDED IN	Requirements class 3: `/req/controlstream`
CONDITIONS	The server implements `http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/req/system`
A	The server SHALL implement a ControlStream resources endpoint at path `{api_root}/systems/{sysId}/controlstreams` for each available `System` resource.
B	The endpoint SHALL only expose the `ControlStream` resources associated to the `System` with ID `sysId`.

The set of control streams associated to a specific deployment can also be made available at a nested endpoint under the corresponding `Deployment` resource:

REQUIREMENT 23	
IDENTIFIER	`/req/controlstream/ref-from-deployment`
INCLUDED IN	Requirements class 3: `/req/controlstream`
CONDITIONS	• The server implements `http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/req/deployment` • The server provides the `controlstreams` association as part of `Deployment` resource representations.
A	The server SHALL implement a ControlStream resources endpoint at path `{api_root}/deployments/{depId}/controlstreams` for each available `Deployment` resource.

REQUIREMENT 23	
B	The endpoint SHALL only expose the `ControlStream` resources associated to a system that was deployed during the `Deployment` with ID `depId`, and whose valid time intersects the deployment time period.

10.5. ControlStream Collections

Any number of resources collections containing `ControlStream` resources can be available at a CS API endpoint. `ControlStream` collections are identified with the item type `ControlStream`.

`ControlStream` resources can be grouped into collections according to any arbitrary criteria, as exemplified below:

Example: Examples of ControlStream Collections

- All control streams used to control a fleet of unmanned systems at URL `{api_root}/collections/uxs_controlstreams`
- All control streams used to task agents in a squad at URL `{api_root}/collections/squad1_controlstreams`

Note that a given control stream can be part of multiple collections at the same time.

REQUIREMENT 24	
IDENTIFIER	`/req/controlstream/collections`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	If the server exposes collections of `ControlStream` resources, it SHALL be done as specified in Clause 8.3.
B	The server SHALL identify all resource collections containing `ControlStream` resources by setting the `itemType` attribute to `ControlStream` in the Collection metadata.
C	For any resource collection with `itemType` set to `ControlStream`, the HTTP GET operation at the path `/collections/{collectionId}/items` SHALL support the query parameters and response of a ControlStream resources endpoint.

10.6. Command Schemas

A different command schema is needed for each individual control stream because the exact content of the command parameters changes according to the the properties being controlled. Moreover, multiple command formats can be offered for any given control stream, and each format may express the schema in a different manner.

Thus, for each `ControlStream` resource, the CS API provides a way for the server to communicate a command schema (*not necessarily a JSON schema*) for each supported command format. The exact content of a schema resource is defined by each encoding. See section Clause 16.1.7 for example schemas used for commands encoded using the default JSON format.

Extensions to this Standard can define additional representation formats for `Command` resources. Such format extensions must clearly define the mapping between elements of the representation format and the Command resource defined in the next clause.

REQUIREMENT 25	
IDENTIFIER	`/req/controlstream/schema-op`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	For every `ControlStream` resource exposed at the CS API endpoint, the server SHALL support the HTTP GET operation at the path `{api_root}/controlstreams/{id}/schema`.

B	The operation SHALL support the parameter `cmdFormat` with the following characteristics (using an OpenAPI 3.0 fragment):
	`name: cmdFormat`
	`in: query`
	`description: Media type of the desired command format`
	`required: false`
	`schema:`
	`  type: string`

C	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The server SHALL return a single schema corresponding to the format identified by the `cmdFormat` parameter.
D	Error cases SHALL be reported using HTTP status codes listed in Clause 7.5.1 of OGC API — Features — Part 1: Core.

Example: Example
If a control stream reports the following supported command formats:

- `application/json`
- `application/swe+csv`
- `application/swe+binary`

The schema for each of these formats is obtained with the following requests, respectively:

- `https://{api_root}/controlstreams/{id}/schema?cmdFormat=application/json`
- `https://{api_root}/controlstreams/{id}/schema?cmdFormat=application/swe%2Bcsv`
- `https://{api_root}/controlstreams/{id}/schema?cmdFormat=application/swe%2Bbinary`

Note that the media type in the request has to be properly URL encoded.

10.7. Command Resource

In the CS API Standard, `Command` resources are a special kind of resource that implements a generalization of the *sosa:Actuation* concept.

In the CS API, `Command` resources are always associated to a `ControlStream`. Some properties of the command (e.g., link to the parent system, controlled properties) can thus be omitted as they are provided at the control stream level.

In addition, the CS API does not restrict `Command` resources to have a single controlled property. It is thus possible to package the desired value of several controlled parameters in a single command, and processing a command can result in actions on several properties at once (e.g., both orientation and FOV of a camera can be modified with a single 'ptz' command).

This section defines the attributes and associations composing a `Command` resource, but the exact way they are encoded in the payload is defined by each encoding. For encodings defined in this document, please see:

- Command resource encoded in JSON

Below is the contextual class diagram of the `Command` resource:

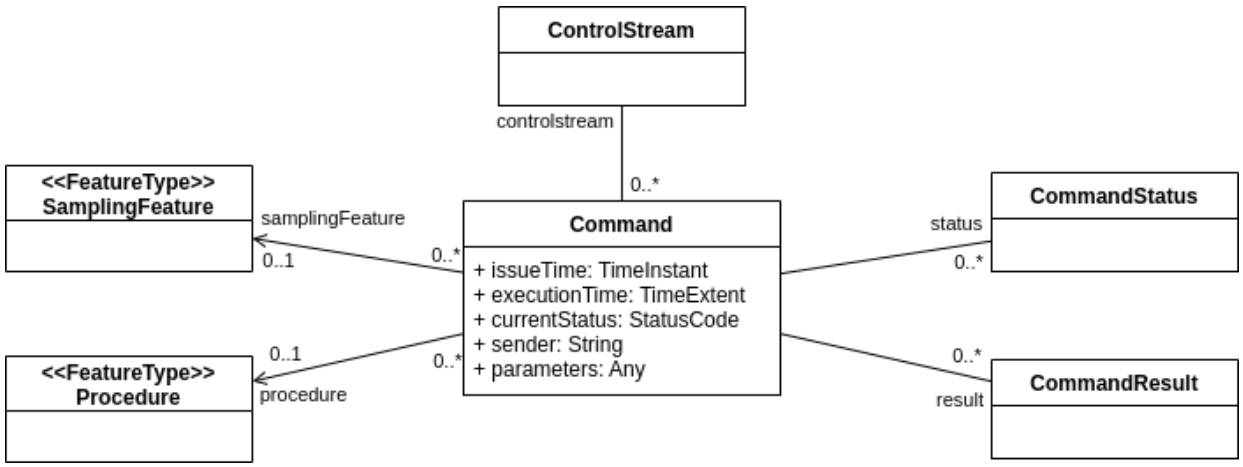

Figure 5 — Command Resource Diagram

10.7.1. Properties

The following tables describe the attributes and associations of the `Command` resource.

Table 11 — Command Attributes

NAME	DEFINITION	DATA TYPE	USAGE
`issueTime`	The time the command was received by the system.	DateTime	Required*
`executionTime`	The time period during which the command was executed. The time period ends when the effect of the command has modified all controlled properties of the feature of interest.	TimeExtent	Optional
`sender`	Identifier of the user or entity who issued the command	String	Optional
`currentStatus`	Current status code of the command (see Table 14).	Enum	Required*
`parameters`	The value of the command parameters.	Any	Required

(*) These properties are required when a command is reported by the server but not when creating or updating a command. If provided on creation, they should be ignored by the server.

Table 12 — Command Associations

RELATION NAME	DEFINITION	TARGET CONTENT
`controlstream`	The `ControlStream` that the command is part of.	A single `ControlStream` resource.
`samplingFeature`	The feature of interest whose properties are changed by the command.	A single `SamplingFeature` resource.
`procedure`	The procedure used to process the command.	A single `Procedure` resource.
`status`	List of status reports related to the command.	A list of `CommandStatus` resources.
`result`	List of results generated during the execution of the command.	A list of `CommandResult` resources.

10.8. Command Canonical URL

The CS API Standard requires that every `Command` resource has a canonical URL.

REQUIREMENT 26	
IDENTIFIER	`/req/controlstream/cmd-canonical-url`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	All `Command` resources exposed by the server SHALL be accessible through their canonical URL of the form `{api_root}/commands/{id}` where `id` is the local identifier of the `Command` resource.
B	If a `Command` resource is retrieved through any other URL than its canonical URL, the server response SHALL include a link to its canonical URL with relation type `canonical`.

10.9. Command Resources Endpoint

10.9.1. Definition

A `Command` resources endpoint is an endpoint exposing a set of `Command` resources that can be further filtered using query parameters.

REQUIREMENT 27	
IDENTIFIER	`/req/controlstream/cmd-resources-endpoint`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	The server SHALL support the HTTP GET operation at the path associated to the `Command` resources endpoint.
B	The operation SHALL support the parameter `limit` defined in Clause 7.15.2 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in OGC API — Features — Part 1: Core requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The response SHALL only include the Observation resources selected by the request.
D	Error cases SHALL be reported using HTTP status codes listed in Clause 7.5.1 of OGC API — Features — Part 1: Core.

10.9.2. Canonical Command Resources Endpoint

The CS API Standard requires that a canonical `Command` resources endpoint, exposing all `Command` resources, be made available by the server.

REQUIREMENT 28	
IDENTIFIER	`/req/controlstream/cmd-canonical-endpoint`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	The server SHALL expose a Command resources endpoint at the path `{api_root}/commands`.
B	The endpoint SHALL expose all `Command` resources available on the server.

10.9.3. Nested Command Resources Endpoint

The set of commands received within a specific control stream is available at a nested endpoint under the corresponding `ControlStream` resource:

REQUIREMENT 29	
IDENTIFIER	`/req/controlstream/cmd-ref-from-controlstream`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	The server SHALL implement a Command resources endpoint at path `{api_root}/controlstream/{csId}/commands` for each available `ControlStream` resource.
B	The endpoint SHALL only expose the `Command` resources that are part of the parent `ControlStream` with ID `csId`.

10.10. Command Collections

Any number of resources collections containing `Command` resources can be available at a CS API endpoint. `Command` collections are identified with the item type `Command`.

`Command` resources can be grouped into collections according to any arbitrary criteria.

Example: Examples of Command Collections

- All commands received from user A `{api_root}/collections/userA_commands` (would likely be visible only to this user)
- All commands that targeted a specific feature of interest B `{api_root}/collections/featureB_commands`

Note that a given commands can be part of multiple collections at the same time.

REQUIREMENT 30	
IDENTIFIER	`/req/controlstream/cmd-collections`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	If the server exposes collections of `Command` resources, it SHALL be done as specified in Clause 8.3.
B	The server SHALL identify all resource collections containing `Command` resources by setting the `itemType` attribute to `Command` in the Collection metadata.
C	For any resource collection with `itemType` set to `Command`, the HTTP GET operation at the path `/collections/{collectionId}/items` SHALL support the query parameters and response of a Command resources endpoint.

10.11. CommandStatus Resource

`CommandStatus` resources represent a status report describing the status/progress of a command at a given point in time.

When commands are processed synchronously, a single status report is provided in the HTTP response. The status can be either `COMPLETED`,`REJECTED` or `FAILED`.

When commands are processed asynchronously, several status reports can be issued for any given command. They are used to report early acceptance/rejection of the command, scheduling and execution steps as well as failure and cancellations. It is recommended for a server to generate appropriate status reports to report incremental progress of long running commands.

This section defines the attributes and associations composing a `CommandStatus` resource, but the exact way they are encoded in the payload is defined by each encoding. For encodings defined in this document, please see:

- Command Status resource encoded in JSON

Below is the contextual class diagram of the `CommandStatus` resource:

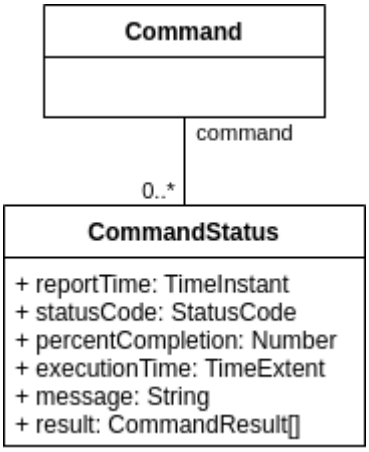

Figure 6 — Command Status Resource Diagram

10.11.1. Properties

The following tables describe the attributes and associations of the `CommandStatus` resource.

Table 13 — Command Status Attributes

NAME	DEFINITION	DATA TYPE	USAGE
`reportTime`	The time when the status report was generated.	DateTime	Required
`statusCode`	Code describing the state of the command (see Table 14).	Enum	Required
`percentCompletion`	Estimated progress as a percentage of total progress.	Number	Optional
`executionTime`	The time period during which the command was or will be executed. This can either represent the estimated or actual execution time period depending on the associated status code (see status code in Table 14). The time period ends when the effect of the command has modified all controlled properties of the feature of interest.	TimeExtent	Optional
`message`	A human readable status message.	String	Optional
`result`	The result of the command associated to the progress report (this can be a partial result).	List of Command Result	Optional

Table 14 — Command Status Codes

CODE	USAGE
PENDING	The command is pending, meaning it has been received by the system but no decision to accept or reject it has been made.
ACCEPTED	The command was accepted by the receiving system. This usually means that the command has passed the first validation steps, but it can still be rejected or fail later during execution.
REJECTED	The command was rejected by the receiving system. It won't be executed at all and the `message` property provides the reason for the rejection. This is a final state. No further status updates will be sent.
SCHEDULED	The command was validated and effectively scheduled by the receiving system. When this status code is used, the scheduled execution time must be provided.
UPDATED	An update to the command was received and accepted. This code must be used if the system supports task updates.
CANCELED	The command was canceled by an authorized user. This code must be used if the system supports user driven task cancellations. The `REJECTED` state should be used instead if the command was canceled by the receiving system. This is a final state. No further status updates will be sent.
EXECUTING	The command is currently being executed by the receiving system. The status message can provide more information about the current progress. A system can send several status updates with this code but different time stamps to report progress incrementally. In particular, the progress percentage and the end of the (estimated) execution time period can be refined in each update.
COMPLETED	The command has completed after a successful execution. The actual execution time must be provided. This is a final state. No further status updates will be sent.
FAILED	The command has failed during execution. The error and/or status message provides the reason for failure. This is a final state. No further status updates will be sent.

Table 15 — Command Status Associations

NAME	DEFINITION	TARGET CONTENT
`command`	The `Command` that this status report relates to.	A single `Command` resource.

10.12. CommandStatus Resources Endpoint

10.12.1. Definition

A Command Status resources endpoint is an endpoint exposing a set of `CommandStatus` resources that can be further filtered using query parameters.

REQUIREMENT 31	
IDENTIFIER	`/req/controlstream/status-resources-endpoint`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	The server SHALL support the HTTP GET operation at the path associated to the `CommandStatus` resources endpoint.
B	The operation SHALL support the parameters `limit` and `datetime` defined in Clause 7.15.2 of OGC API — Features — Part 1: Core. All references to the term "features" or "feature" in OGC API — Features — Part 1: Core requirements SHALL be replaced by the terms "resources" or "resource", respectively.
C	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The response SHALL only include the CommandStatus resources selected by the request.
D	Error cases SHALL be reported using HTTP status codes listed in Clause 7.5.1 of OGC API — Features — Part 1: Core.

10.12.2. Status Endpoint

The set of status reports associated to a given command is available at a nested endpoint under the corresponding `Command` resource:

REQUIREMENT 32	
IDENTIFIER	`/req/controlstream/command-status-endpoint`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	The server SHALL implement a Command Status resources endpoint at path `{api_root}/command/{cmdId}/status` for every `Command` resource.
B	The endpoint SHALL only expose the `CommandStatus` resources that are related to the `Command` resource with local ID `cmdId`.

10.13. CommandResult Resource

Certain types of commands lead to the production of data (e.g., tasking a system for data collection, triggering a process such as a simulation run, etc.).

Different types of result can be attached to a command status report:

- Reference to one or more datastreams containing the result data, each with an optional time range;
- Reference to one or more individual observations containing the result data;
- Reference to a collection of observations containing the result data; and
- Inline result data encoded as described by the control stream result schema (can be one or more records).

NOTE 1:In the case of a command result provided inline, the result data may or may not be recorded separately in a datastream.
NOTE 2:For commands executed synchronously, the result can be provided as part of the status report returned in the HTTP response.

The following examples describe how command results are used in various use cases:

Example: Example 1: Chemical plume simulation
A command is used to trigger a new run of a chemical plume dispersion model with certain parameters. The output of the model is a time series of observations where each observation provides the location of all particles for a given time (phenomenonTime). Each processed command leads to the creation of a new datastream that will contain all observations resulting for the model run. When the run is completed, a last progress report is provided with a reference to the datastream.
Example 2: UAV video footage task
A command is used to task a UAV to collect video data while orbiting around a building. The output is a set of may observations (i.e., video frames) that are appended to the existing video datastream of the UAV. When the task is completed, a last status report is provided with a reference to the video datastream with a time range selecting the portion of the video stream that is relevant to the task.

Example 3: UAV picture task
A command is used to task a UAV to collect an image at a specific location. The output is a single observation that is appended to the existing image datastream of the UAV. When the task is completed, a last progress report is provided with a reference to the image observation.

Example 4: Satellite imagery acquisition
A command is used to task an earth observation satellite to collect imagery to cover a given geographic area (i.e., coverage request). The output is a set of one or more image observations that are appended to the existing image datastream of the EO sensor. When the task is completed, a last status report is provided with references to all the collected image observations.

Example 5: System state retrieval
A command is used to query the state of a system. The result of the query is provided inline in the status report (potentially synchronously if the state data can be retrieved quickly).

Example 6: On-demand processing

A command is used to trigger a simple on-demand process that computes temporal averages of parameters in a datastream over a certain time period. The output of the process is provided inline in the status report (potentially synchronously if the computation can be done quickly).

This section defines the attributes and associations composing a `CommandResult` resource, but the exact way they are encoded in the payload is defined by each encoding. For encodings defined in this document, please see:

- Command Result resource encoded in JSON

Below is the contextual class diagram of the `CommandResult` resource:

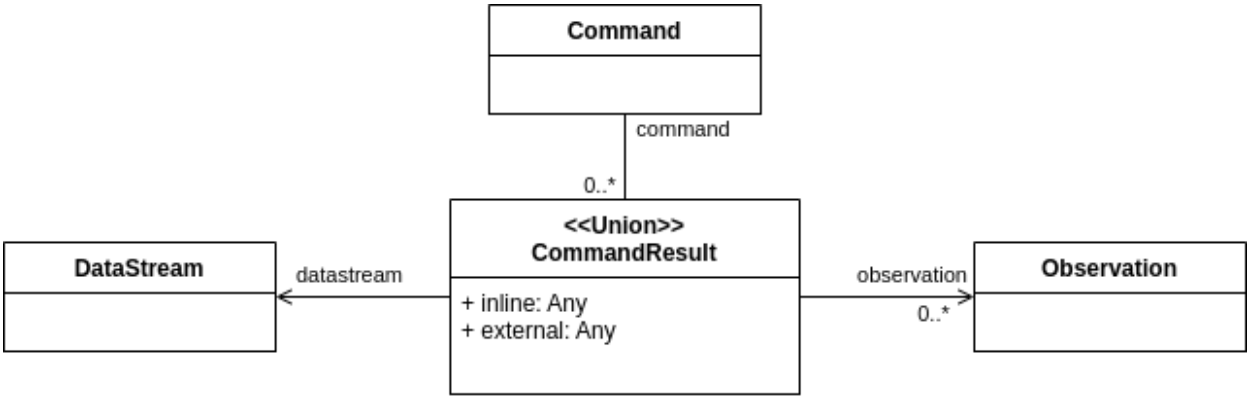

Figure 7 — Command Result Resource Diagram

10.13.1. Properties

The following tables describe the attributes and associations of the `CommandResult` resource. At least one of the properties must be provided.

Table 16 — Command Result Attributes

NAME	DEFINITION	DATA TYPE	USAGE
`inline`	The result data provided inline (encoded according to the `ControlStream` result schema).	Any	Optional

Table 17 — Command Result Associations

NAME	DEFINITION	TARGET CONTENT
`observation`	An observation resulting from the execution of the command	A list of `Observation` resources (by reference).
`datastream`	A datastream containing observations resulting from the execution of the command	A single `DataStream` resource (by reference).

NAME	DEFINITION	TARGET CONTENT
`external`	An external dataset containing the results of the command.	Any resource (by reference).

10.14. CommandResult Resources Endpoint

10.14.1. Definition

A Command Result resources endpoint is an endpoint exposing a set of `CommandResult` resources that can be further filtered using query parameters.

REQUIREMENT 33	
IDENTIFIER	`/req/controlstream/result-resources-endpoint`
INCLUDED IN	Requirements class 3: `/req/controlstream`
A	The server SHALL support the HTTP GET operation at the path associated to the `CommandResult` resources endpoint.
B	The operation SHALL support the parameter `limit` defined in Clause 7.15.2 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in OGC API — Features — Part 1: Core requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The response SHALL only include the CommandResult resources selected by the request.
D	Error cases SHALL be reported using HTTP status codes listed in Clause 7.5.1 of OGC API — Features — Part 1: Core.

10.14.2. Result Endpoint

The set of result items associated to a given command is available at a nested endpoint under the corresponding `Command` resource:

REQUIREMENT 34	
IDENTIFIER	`/req/controlstream/command-result-endpoint`
INCLUDED IN	Requirements class 3: `/req/controlstream`

REQUIREMENT 34	
A	The server SHALL implement a Command Result resources endpoint at path `{api_root}/command/{cmdId}/result` for every `Command` resource that can be associated to a result.
B	The endpoint SHALL only expose the `CommandResult` resources that are related to the `Command` resource with local ID `cmdId`.

11

REQUIREMENTS CLASS "COMMAND FEASIBILITY"

11 REQUIREMENTS CLASS "COMMAND FEASIBILITY"

11.1. Overview

REQUIREMENTS CLASS 4	
IDENTIFIER	`/req/feasibility`
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.4: `/conf/feasibility`
PREREQUISITE	Requirements class 3: `/req/controlstream`
NORMATIVE STATEMENTS	Requirement 35: `/req/feasibility/canonical-url` Requirement 36: `/req/feasibility/ref-from-controlstream` Requirement 37: `/req/feasibility/status-endpoint` Requirement 38: `/req/feasibility/result-endpoint` Requirement 39: `/req/feasibility/collections`

The execution of certain commands is sometimes impossible due to internal or external constraints (e.g., conflict with other scheduled commands, conflict with another user that has exclusive control on the system, etc.).

In order for a client to know if a command/task is feasible without sending the actual command, feasibility channels are supported by the CS API.

In addition to providing a binary (i.e., YES/NO) response to a feasibility request, the CS API also provides a mechanism for returning detailed feasibility analysis information to a client (e.g., provide chances of success, task execution steps, alternatives, etc.).

A feasibility request is initiated by creating a `Command` resource on the feasibility channel. The server can then respond synchronously or asynchronously just like for a regular command channel. The parameters used for the feasibility request are the same as the one for the

corresponding commands (i.e., both commands and feasibility share the same parameters schema).

11.2. Feasibility Resource

A `Feasibility` resource is a `Command` resource created on a control stream feasibility channel (see Clause 10.7, Command Resource for details).

All nested resources available under a regular command resource are also available under the feasibility resource.

11.3. Feasibility Canonical URL

The CS API Standard requires that every `Feasibility` resource has a canonical URL.

REQUIREMENT 35	
IDENTIFIER	`/req/feasibility/canonical-url`
INCLUDED IN	Requirements class 4: `/req/feasibility`
A	All `Feasibility` resources exposed by the server SHALL be accessible through their canonical URL of the form `{api_root}/feasibility/{id}` where `id` is the local identifier of the `Feasibility` resource.
B	If a `Feasibility` resource is retrieved through any other URL than its canonical URL, the server response SHALL include a link to its canonical URL with relation type `canonical`.

11.4. Feasibility Endpoint

The set of feasibility requests received for a specific control stream is available at a nested endpoint under the corresponding `ControlStream` resource:

REQUIREMENT 36	
IDENTIFIER	`/req/feasibility/ref-from-controlstream`

REQUIREMENT 36	
INCLUDED IN	Requirements class 4: `/req/feasibility`
A	The server SHALL implement a Command resources endpoint at path `{api_root}/controlstream/{csId}/feasibility` for each available `ControlStream` resource.
B	The endpoint SHALL only expose the `Feasibility` resources that are part of the parent `ControlStream` with ID `csId`.

11.5. Feasibility Status

Feasibility status is provided using a `CommandStatus` resource (see Clause 10.11, CommandStatus Resource for details).

The following table clarifies the meaning of status codes in the case of a feasibility request:

Table 18 — Feasibility Status Codes

CODE	USAGE
PENDING	The feasibility request is pending, meaning it has been received by the system but no decision to accept or reject it has been made.
ACCEPTED	The feasibility request was accepted by the receiving system. This usually means that the request parameters have passed the first validation steps, but it can still be rejected or fail later during the analysis.
REJECTED	The feasibility request was rejected by the receiving system. It won't be processed at all and the `message` property provides the reason for the rejection. This is a final state. No further status updates will be sent.
SCHEDULED	Unused for feasibility requests.
UPDATED	Unused for feasibility requests.
CANCELED	The feasibility request was canceled by an authorized user. This code must be used if the system supports user driven task cancellations. The `REJECTED` state should be used instead if the feasibility analysis was canceled by the receiving system. This is a final state. No further status updates will be sent.
EXECUTING	The feasibility request is currently being processed by the receiving system. The status message can provide more information about the current progress. A system can send several status updates with this code but different time stamps to report progress incrementally. In particular, the progress percentage and the end of the (estimated) execution time period can be refined in each update.
COMPLETED	The feasibility analysis has completed successfully. The actual execution time must be provided. This is a final state. No further status updates will be sent.

CODE	USAGE
FAILED	The feasibility analysis has failed during processing. The error and/or status message provides the reason for failure. This is a final state. No further status updates will be sent.

The set of status reports associated to a given feasibility request is available at a nested endpoint under the corresponding `Feasibility` resource:

REQUIREMENT 37	
IDENTIFIER	`/req/feasibility/status-endpoint`
INCLUDED IN	Requirements class 4: `/req/feasibility`
A	The server SHALL implement a Command Status resources endpoint at path `{api_root}/feasibility/{feasId}/status` for every `Feasibility` resource.
B	The endpoint SHALL only expose the `CommandStatus` resources that are related to the `Feasibility` resource with local ID `feasId`.

11.6. Feasibility Result

Feasibility results are provided using `CommandResult` resources (see Clause 10.13, CommandResult Resource for details).

The results of a feasibility analysis are usually provided inline. The result structure must match the "feasibility result schema" provided by the parent `ControlStream` resource. The "feasibility result schema" is typically different from the "command result schema".

Below are examples of feasibility results for various use cases:

Example: Example 1: Tasking a UAV to go to a lat/lon location
In addition to a binary (yes/no) feasibility response, the result of the feasibility analysis may include the earliest time at which the location could be reached, as well as the expected trajectory.
Example 2: Tasking a satellite to cover an area with visible imagery
The result of the feasibility analysis may include all the attempts needed to have a high enough chance of success to obtain a clear (i.e., cloud free) image of the area. If the area is too large to be covered with a single image, an estimated success rate and completion time could be provided separately for each subdivision of the geographic area.

The set of result items associated to a given feasibility request is available at a nested endpoint under the corresponding `Feasibility` resource:

REQUIREMENT 38	
IDENTIFIER	`/req/feasibility/result-endpoint`
INCLUDED IN	Requirements class 4: `/req/feasibility`
A	The server SHALL implement a Command Result resources endpoint at path `{api_root}/feasibility/{feasId}/result` for every `Feasibility` resource.
B	The endpoint SHALL only expose the `CommandResult` resources that are related to the `Feasibility` resource with local ID `feasId`.

11.7. Feasibility Collections

Collections of feasibility resources are also supported, but are optional.

REQUIREMENT 39	
IDENTIFIER	`/req/feasibility/collections`
INCLUDED IN	Requirements class 4: `/req/feasibility`
A	If the server exposes collections of `Feasibility` resources, it SHALL be done as specified in Clause 8.3.
B	The server SHALL identify all resource collections containing `Feasibility` resources by setting the `itemType` attribute to `Feasibility` in the Collection metadata.
C	For any resource collection with `itemType` set to `Feasibility`, the HTTP GET operation at the path `/collections/{collectionId}/items` SHALL support the query parameters and response of a Command resources endpoint.

12

REQUIREMENTS CLASS "SYSTEM EVENTS"

12 REQUIREMENTS CLASS "SYSTEM EVENTS"

REQUIREMENTS CLASS 5	
IDENTIFIER	`/req/system-event`
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.5: `/conf/system-event`
PREREQUISITES	Requirements class 1: `/req/api-common` http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/req/system
NORMATIVE STATEMENTS	Requirement 40: `/req/system-event/canonical-url` Requirement 41: `/req/system-event/resources-endpoint` Requirement 42: `/req/system-event/canonical-endpoint` Requirement 43: `/req/system-event/ref-from-system` Requirement 44: `/req/system-event/collections`

12.1. Overview

`SystemEvent` resources are used to capture information about various events and maintenance operations occurring on (observing) systems such as recalibrations, part replacements, software updates, relocations/deployments, operator handoffs, decommissioning, etc.

This section predefines a certain number of event types but the list can be extended further by extensions.

12.2. SystemEvent Resource

`SystemEvent` resources are modeled on the SensorML Event class.

This section defines the attributes and associations composing a `SystemEvent` resource, but the exact way they are encoded in the payload is defined by each encoding. For encodings defined in this document, please see:

- System Event resource encoded in JSON

Below is the contextual class diagram of the `SystemEvent` resource:

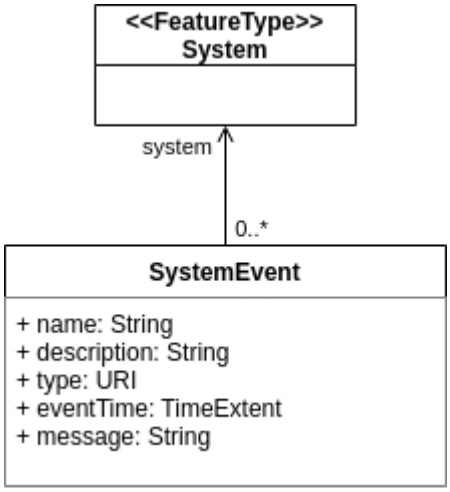

Figure 8 — System Event Diagram

12.2.1. Properties

The following tables describe the attributes and associations of a `SystemEvent` resource.

Table 19 — System Event Attributes

NAME	DEFINITION	DATA TYPE	USAGE
`name`	The name of the event.	String	Required
`description`	A human readable description for the event.	String	Optional
`type`	The type of event (see Table 20).	URI	Required
`eventTime`	The time the event occurred on the system.	TimeExtent	Required
`message`	A human readable message from the operator.	String	Optional

Table 20 — System Event Types

TYPE URI	USAGE
http://www.opengis.net/def/x-OGC/TBD/Calibration	System was calibrated or recalibrated.
http://www.opengis.net/def/	The configuration was changed.

TYPE URI	USAGE
x-OGC/TBD/ConfigurationChange	
http://www.opengis.net/def/x-OGC/TBD/SoftwareUpdate	The software was updated.
http://www.opengis.net/def/x-OGC/TBD/PartReplacement	One or more physical parts were replaced.
http://www.opengis.net/def/x-OGC/TBD/Relocation	The system was moved to a different location.
http://www.opengis.net/def/x-OGC/TBD/Deployment	The system was deployed.
http://www.opengis.net/def/x-OGC/TBD/Decommission	The system was decommissioned.

Table 21 — System Event Associations

RELATION NAME	DEFINITION	TARGET CONTENT
`system`	Link to the `System` this event relates to.	A single `System` resource.

12.3. SystemEvent Canonical URL

The CS API Standard requires that every `SystemEvent` resource has a canonical URL.

REQUIREMENT 40	
IDENTIFIER	`/req/system-event/canonical-url`
INCLUDED IN	Requirements class 5: `/req/system-event`

REQUIREMENT 40	
A	All `SystemEvent` resources exposed by the server SHALL be accessible through their canonical URL of the form `{api_root}/systemEvents/{id}` where `id` is the local identifier of the `SystemEvent` resource.
B	If a `SystemEvent` resource is retrieved through any other URL than its canonical URL, the server response SHALL include a link to its canonical URL with relation type `canonical`.

12.4. SystemEvent Resources Endpoints

12.4.1. Definition

A `SystemEvent` resources endpoint is an endpoint exposing a set of `SystemEvent` resources that can be further filtered using query parameters.

REQUIREMENT 41	
IDENTIFIER	`/req/system-event/resources-endpoint`
INCLUDED IN	Requirements class 5: `/req/system-event`
A	The server SHALL support the HTTP GET operation at the path associated to the `SystemEvent` resources endpoint.
B	The operation SHALL support the parameters `limit` and `datetime` defined in Clause 7.15.2 and Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in OGC API — Features — Part 1: Core requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	A successful execution of the operation SHALL be reported as a response with a HTTP status code 200. The response SHALL only include the SystemEvent resources selected by the request.
D	Error cases SHALL be reported using HTTP status codes listed in Clause 7.5.1 of OGC API — Features — Part 1: Core.

Clause 13.7 defines additional query parameters applicable to `SystemEvent` resources endpoint.

12.4.2. Canonical SystemEvent Resources Endpoint

The CS API Standard requires that a canonical `SystemEvent` resources endpoint, exposing all `SystemEvent` resources, be made available by the server.

REQUIREMENT 42	
IDENTIFIER	`/req/system-event/canonical-endpoint`
INCLUDED IN	Requirements class 5: `/req/system-event`
A	The server SHALL expose a System Event resources endpoint at the path `{api_root}/systemEvents`.
B	The endpoint SHALL expose all `SystemEvent` resources available on the server.

12.4.3. Nested SystemEvent Resources Endpoints

The set of events related to a specific system is available at a nested endpoint under the corresponding `System` resource:

REQUIREMENT 43	
IDENTIFIER	`/req/system-event/ref-from-system`
INCLUDED IN	Requirements class 5: `/req/system-event`
A	The server SHALL implement a System Event resources endpoint at path `{api_root}/systems/{sysId}/events` for each available `System` resource.
B	The endpoint SHALL only expose the `SystemEvent` resources associated to the `System` with ID `sysId`.

12.5. SystemEvent Collections

Any number of resources collections containing `SystemEvent` resources can be available at a CS API endpoint. `SystemEvent` collections are identified with the item type `SystemEvent`.

`SystemEvent` resources can be grouped into collections according to any arbitrary criteria, as exemplified below:

Example: Examples of SystemEvent Collections

- All events related to a fleet of UAVs at URL `{api_root}/collections/uas_fleet1_events`
- All events related to a particular mission at URL `{api_root}/collections/mission23_events`

Note that a given event can be part of multiple collections at the same time.

REQUIREMENT 44

IDENTIFIER	`/req/system-event/collections`
INCLUDED IN	Requirements class 5: `/req/system-event`
A	If the server exposes collections of `SystemEvent` resources, it SHALL be done as specified in Clause 8.3.
B	The server SHALL identify all resource collections containing `SystemEvent` resources by setting the `itemType` attribute to `SystemEvent` in the Collection metadata.
C	For any resource collection with `itemType` set to `SystemEvent`, the HTTP GET operation at the path `/collections/{collectionId}/items` SHALL support the query parameters and response of a System Event resources endpoint.

13

REQUIREMENTS CLASS "ADVANCED FILTERING"

13 REQUIREMENTS CLASS "ADVANCED FILTERING"

REQUIREMENTS CLASS 6	
IDENTIFIER	`/req/advanced-filtering`
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.6: `/conf/advanced-filtering`
PREREQUISITES	Requirements class 1: `/req/api-common` http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/req/advanced-filtering
NORMATIVE STATEMENTS	Requirement 45: `/req/advanced-filtering/datastream-by-phenomenontime` Requirement 46: `/req/advanced-filtering/datastream-by-resulttime` Requirement 47: `/req/advanced-filtering/datastream-by-obsprop` Requirement 48: `/req/advanced-filtering/datastream-by-foi` Requirement 49: `/req/advanced-filtering/obs-by-phenomenontime` Requirement 50: `/req/advanced-filtering/obs-by-resulttime` Requirement 51: `/req/advanced-filtering/obs-by-foi` Requirement 52: `/req/advanced-filtering/controlstream-by-issuetime` Requirement 53: `/req/advanced-filtering/controlstream-by-exectime` Requirement 54: `/req/advanced-filtering/controlstream-by-controlprop` Requirement 55: `/req/advanced-filtering/controlstream-by-foi` Requirement 56: `/req/advanced-filtering/cmd-by-issuetime` Requirement 57: `/req/advanced-filtering/cmd-by-exectime` Requirement 58: `/req/advanced-filtering/cmd-by-status` Requirement 59: `/req/advanced-filtering/cmd-by-sender` Requirement 60: `/req/advanced-filtering/cmd-by-foi` Requirement 61: `/req/advanced-filtering/status-by-statuscode` Requirement 62: `/req/advanced-filtering/event-by-type`

13.1. Overview

This requirements class specifies additional filtering options that may be used to select only a subset of the resources in a collection.

All filters defined in this section are implemented using URL query parameters and are used in addition to the ones defined in other requirements classes. In particular, all parameters

defined in Clause 16.3 of OGC API — Connected Systems — Part 1 shall also be supported on all resource types.

13.2. DataStream Query Parameters

The following query parameters are used to filter `DataStream` resources at a DataStream resources endpoint.

13.2.1. Phenomenon Time Filter

This filter is used to select datastreams based on their `phenomenonTime` extent.

REQUIREMENT 45	
IDENTIFIER	`/req/advanced-filtering/datastream-by-phenomenontime`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an DataStream resources endpoint SHALL support a parameter `phenomenonTime`.
B	The parameter SHALL fulfill the same requirements as the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in these requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	Only the `phenomenonTime` property of `DataStream` resources SHALL be used to determine the temporal extent evaluated against the parameter.

13.2.2. Result Time Filter

This filter is used to select datastreams based on their `resultTime` extent.

REQUIREMENT 46	
IDENTIFIER	`/req/advanced-filtering/datastream-by-resulttime`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an DataStream resources endpoint SHALL support a parameter `resultTime`.

REQUIREMENT 46	
B	The parameter SHALL fulfill the same requirements as the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in these requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	Only the `resultTime` property of `DataStream` resources SHALL be used to determine the temporal extent evaluated against the parameter.

13.2.3. Observed Property Filter

This filter is used to select datastreams that include specific observable properties.

REQUIREMENT 47	
IDENTIFIER	`/req/advanced-filtering/datastream-by-obsprop`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at a DataStream resources endpoint SHALL support a parameter `observedProperty` of type ID_List.
B	Only datastreams that include an observed property that has one of the requested identifiers SHALL be part of the result set.

13.2.4. Feature of Interest Filter

This filter is used to select datastreams that are associated to specific sampling features or (ultimate) features of interest.

REQUIREMENT 48	
IDENTIFIER	`/req/advanced-filtering/datastream-by-foi`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an DataStream resources endpoint SHALL support a parameter `foi` of type ID_List.
B	Only `DataStream` resources that are associated to a feature of interest that has one of the requested identifiers SHALL be part of the result set.
C	Both sampling features and domain features of interest SHALL be included in the search.

13.3. Observation Query Parameters

The following query parameters are used to filter `Observation` resources at an {obs-resources-endpoint}.

13.3.1. Phenomenon Time Filter

This filter is used to select observations based on their `phenomenonTime` property.

REQUIREMENT 49	
IDENTIFIER	`/req/advanced-filtering/obs-by-phenomenontime`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an Observation resources endpoint SHALL support a parameter `phenomenonTime`.
B	The parameter SHALL fulfill the same requirements as the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in these requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	Only the `phenomenonTime` property of `Observation` resources SHALL be used to determine the temporal extent evaluated against the parameter.

Example
Example queries to find `Observations` by `phenomenonTime`

closed interval	`{api_root}/datastreams/123/observations?phenomenonTime=2018-02-12T00:00:00Z/2018-03-18T12:31:12Z`
open interval	`{api_root}/datastreams/123/observations?phenomenonTime=2018-02-12T00:00:00Z/..`
special case `now`	`{api_root}/datastreams/123/observations?phenomenonTime=now`

13.3.2. Result Time Filter

This filter is used to select observations based on their `resultTime` property.

REQUIREMENT 50	
IDENTIFIER	`/req/advanced-filtering/obs-by-resulttime`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an Observation resources endpoint SHALL support a parameter `resultTime`.
B	The parameter SHALL fulfill the same requirements as the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in these requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	Only the `resultTime` property of `Observation` resources SHALL be used to determine the temporal extent evaluated against the parameter.
D	In addition to the possible parameter values defined in OGC API — Features — Part 1: Core, the parameter SHALL also support the special value `latest`. When this special value is used, only observations with the latest result time SHALL be included in the result set.

Example
Example queries to find `Observations` by `resultTime`

closed interval	`{api_root}/datastreams/123/observations?resultTime=2018-02-12T00:00:00Z/2018-03-18T12:31:12Z`
open interval	`{api_root}/datastreams/123/observations?resultTime=2018-02-12T00:00:00Z/..`
special case `latest`	`{api_root}/datastreams/123/observations?resultTime=latest`

13.3.3. Feature of Interest Filter

This filter is used to select observations that are associated to specific sampling features or ultimate features of interests.

REQUIREMENT 51	
IDENTIFIER	`/req/advanced-filtering/obs-by-foi`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an Observation resources endpoint SHALL support a parameter `foi` of type ID_List.
B	Only `Observation` resources that are associated to a feature of interest that has one of the requested identifiers SHALL be part of the result set.

REQUIREMENT 51	
C	Both sampling features and domain features of interest SHALL be included in the search.

13.4. ControlStream Query Parameters

The following query parameters are used to filter `ControlStream` resources at a ControlStream resources endpoint.

13.4.1. Issue Time Filter

This filter is used to select control streams based on their `issueTime` extent.

REQUIREMENT 52	
IDENTIFIER	`/req/advanced-filtering/controlstream-by-issuetime`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an ControlStream resources endpoint SHALL support a parameter `issueTime`.
B	The parameter SHALL fulfill the same requirements as the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in these requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	Only the `issueTime` property of `CommandStream` resources SHALL be used to determine the temporal extent evaluated against the parameter.

13.4.2. Execution Time Filter

This filter is used to select control streams based on their `executionTime` extent.

REQUIREMENT 53	
IDENTIFIER	`/req/advanced-filtering/controlstream-by-exectime`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`

REQUIREMENT 53	
A	The HTTP GET operation at an ControlStream resources endpoint SHALL support a parameter `executionTime`.
B	The parameter SHALL fulfill the same requirements as the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in these requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	Only the `executionTime` property of `CommandStream` resources SHALL be used to determine the temporal extent evaluated against the parameter.

13.4.3. Controlled Property Filter

This filter is used to select control streams that include specific controllable properties.

REQUIREMENT 54	
IDENTIFIER	`/req/advanced-filtering/controlstream-by-controlprop`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at a ControlStream resources endpoint SHALL support a parameter `controlledProperty` of type ID_List.
B	Only control streams that include a controlled property that has one of the requested identifiers SHALL be part of the result set.

13.4.4. Feature of Interest Filter

This filter is used to select control streams that are associated to specific sampling features or (ultimate) features of interest.

REQUIREMENT 55	
IDENTIFIER	`/req/advanced-filtering/controlstream-by-foi`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an ControlStream resources endpoint SHALL support a parameter `foi` of type ID_List.
B	Only `CommandStream` resources that are associated to a feature of interest that has one of the requested identifiers SHALL be part of the result set.

REQUIREMENT 55	
C	Both sampling features and domain features of interest SHALL be included in the search.

13.5. Command Query Parameters

The following query parameters are used to filter `Command` resources at a Command resources endpoint.

13.5.1. Issue Time Filter

This filter is used to select commands based on their `issueTime` property.

REQUIREMENT 56	
IDENTIFIER	`/req/advanced-filtering/cmd-by-issuetime`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an Command resources endpoint SHALL support a parameter `issueTime`.
B	The parameter SHALL fulfill the same requirements as the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in these requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	Only the `issueTime` property of `Command` resources SHALL be used to determine the temporal extent evaluated against the parameter.

13.5.2. Execution Time Filter

This filter is used to select commands based on their `executionTime` property.

REQUIREMENT 57	
IDENTIFIER	`/req/advanced-filtering/cmd-by-exectime`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`

REQUIREMENT 57	
A	The HTTP GET operation at an Command resources endpoint SHALL support a parameter `executionTime`.
B	The parameter SHALL fulfill the same requirements as the parameter `datetime` defined in Clause 7.15.4 of OGC API — Features — Part 1: Core. All references to the term “features” or “feature” in these requirements SHALL be replaced by the terms “resources” or “resource”, respectively.
C	Only the `executionTime` property of `Command` resources SHALL be used to determine the temporal extent evaluated against the parameter.

13.5.3. Status Filter

This filter is used to select commands based on their `statusCode` property.

REQUIREMENT 58	
IDENTIFIER	`/req/advanced-filtering/cmd-by-status`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`

A	The HTTP GET operation at an Command resources endpoint SHALL support a parameter `statusCode` with the following characteristics (using an OpenAPI 3.0 fragment):
	`name: statusCode`
	`description: |-`
	`  List of command status codes.`
	`  Only command resources whose current status matches one of the provided status codes are selected.`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  minItems: 1`
	`  items:`
	`  type: string`
	`  enum: ["PENDING", "ACCEPTED", "REJECTED", "SCHEDULED", "UPDATED", "CANCELED", "EXECUTING", "FAILED", "COMPLETED"]`
	`explode: false`

B	Only `Command` resources whose current status matches one of the specified status codes SHALL be part of the result set.

13.5.4. Sender Filter

This filter is used to select commands issued by a specific sender.

REQUIREMENT 59	
IDENTIFIER	`/req/advanced-filtering/cmd-by-sender`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`

A	The HTTP GET operation at an Command resources endpoint SHALL support a parameter `sender` with the following characteristics (using an OpenAPI 3.0 fragment):
	`name: sender`
	`description: |-`
	`  List of sender IDs.`
	`  Only command resources issued by one of the specified senders are selected.`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  minItems: 1`
	`  items:`
	`  type: string`
	`explode: false`

B	Only `Command` resources issued by one of the specified senders SHALL be part of the result set.

13.5.5. Feature of Interest Filter

REQUIREMENT 60	
IDENTIFIER	`/req/advanced-filtering/cmd-by-foi`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`
A	The HTTP GET operation at an Command resources endpoint SHALL support a parameter `foi` of type ID_List.
B	Only `Command` resources that are associated to a feature of interest that has one of the requested identifiers SHALL be part of the result set.
C	Both sampling features and domain features of interest SHALL be included in the search.

13.6. CommandStatus Query Parameters

The following query parameters are used to filter `CommandStatus` resources at a Command Status resources endpoint.

13.6.1. StatusCode Filter

REQUIREMENT 61	
IDENTIFIER	`/req/advanced-filtering/status-by-statuscode`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`

A	The HTTP GET operation at an Command Status resources endpoint SHALL support a parameter `statusCode` with the following characteristics (using an OpenAPI 3.0 fragment):
	`name: statusCode`
	`description: |-`
	`  List of command status codes.`
	`  Only command resources whose current status matches one of the provided status codes are selected.`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  minItems: 1`
	`  items:`
	`  type: string`
	`  enum: ["PENDING", "ACCEPTED", "REJECTED", "SCHEDULED", "UPDATED", "CANCELED", "EXECUTING", "FAILED", "COMPLETED"]`
	`explode: false`

B	Only `CommandStatus` resources with a status that matches one of the specified status codes SHALL be part of the result set.

13.7. SystemEvent Query Parameters

The following query parameters are used to filter `SystemEvent` resources at a System Event resources endpoint.

13.7.1. Event Type Filter

REQUIREMENT 62	
IDENTIFIER	`/req/advanced-filtering/event-by-type`
INCLUDED IN	Requirements class 6: `/req/advanced-filtering`

REQUIREMENT 62

A	The HTTP GET operation at an System Event resources endpoint SHALL support a parameter `eventType` with the following characteristics (using an OpenAPI 3.0 fragment):
	`name: eventType`
	`description: |-`
	`  List of event types.`
	`  Only event resources with a type that matches one of the provided types are selected.`
	`in: query`
	`required: false`
	`schema:`
	`  type: array`
	`  minItems: 1`
	`  items:`
	`  type: string`
	`explode: false`

B	Only `SystemEvent` resources with a type that matches one of the specified types SHALL be part of the result set.

14

REQUIREMENTS CLASS "CREATE/REPLACE/DELETE"

14 REQUIREMENTS CLASS "CREATE/REPLACE/DELETE"

14.1. Overview

REQUIREMENTS CLASS 7	
IDENTIFIER	`/req/create-replace-delete`
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.7: `/conf/create-replace-delete`
PREREQUISITE	http://www.opengis.net/spec/ogcapi-features-4/1.0/req/create-replace-delete
NORMATIVE STATEMENTS	Requirement 63: `/req/create-replace-delete/datastream` Requirement 64: `/req/create-replace-delete/datastream-update-schema` Requirement 65: `/req/create-replace-delete/datastream-delete-cascade` Requirement 66: `/req/create-replace-delete/observation` Requirement 67: `/req/create-replace-delete/observation-schema` Requirement 68: `/req/create-replace-delete/controlstream` Requirement 69: `/req/create-replace-delete/controlstream-update-schema` Requirement 70: `/req/create-replace-delete/controlstream-delete-cascade` Requirement 71: `/req/create-replace-delete/command` Requirement 72: `/req/create-replace-delete/command-schema` Requirement 73: `/req/create-replace-delete/command-status` Requirement 74: `/req/create-replace-delete/command-result` Requirement 75: `/req/create-replace-delete/feasibility`

REQUIREMENTS CLASS 7	
	Requirement 76: `/req/create-replace-delete/feasibility-status` Requirement 77: `/req/create-replace-delete/feasibility-result` Requirement 78: `/req/create-replace-delete/system-event`

The “Create/Replace/Delete” requirements class specifies how instances of the resource types defined in this Standard are created, replaced and deleted via a CS API endpoint.

All resources are created, replaced and deleted using CREATE (HTTP POST), REPLACE (HTTP PUT) and DELETE (HTTP DELETE) operations, respectively, as defined by the OGC API — Features — Part 4: Create, Replace, Update and Delete Standard.

OGC API — Features — Part 4: Create, Replace, Update and Delete uses the terms “resources endpoint” and “resource endpoint” to identify the paths where these operations are supported by the server. The following sections provide these endpoints for each resource type defined by the CS API Standard.

14.2. DataStreams

REQUIREMENT 63	
IDENTIFIER	`/req/create-replace-delete/datastream`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Datastreams Observations”
A	The server SHALL support the CREATE operation at the `DataStream` resources endpoints defined by the following URI template: • `{api_root}/systems/{sysId}/datastreams`
B	The server SHALL support the REPLACE and DELETE operations at the `DataStream` resource endpoints defined by the following URI templates: • `{api_root}/systems/{sysId}/datastreams/{id}` • `{api_root}/datastreams/{id}`
C	The `sysId` parameter is the local identifier of the `System` resource the `DataStream` is (or will be) associated to. The `id` parameter is the local identifier of the `DataStream` resource to replace or delete.

The following constraints must be implemented by the server.

REQUIREMENT 64	
IDENTIFIER	`/req/create-replace-delete/datastream-update-schema`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Datastreams Observations”
A	The server SHALL reject a REPLACE request on a `DataStream` resource that modifies the observation schema if the `DataStream` already has nested `Observation` resources. The server SHALL use HTTP status code 409 to report the error.

REQUIREMENT 65	
IDENTIFIER	`/req/create-replace-delete/datastream-delete-cascade`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Datastreams Observations”
A	By default, the server SHALL reject a DELETE request on a `DataStream` resource that has nested `Observation` resources. The server SHALL use HTTP status code 409 to report the error.
B	If the request contains the `cascade` parameter, the server SHALL accept the DELETE request and delete the `DataStream` resource as well as all its nested `Observation` resources.

NOTE 1: A schema must be provided before observations can be inserted in the datastream. The schema is provided along with the `DataStream` resource itself. Only one schema (for only one format) can be provided by a create operation for a given datastream. However, the server is allowed to automatically convert observations to/from other supported formats, as appropriate. This implies that the server can also automatically generate equivalent schemas for these other formats. Future extensions may define patterns to allow client to define multiple schemas themselves.

NOTE 2: After a datastream has been created and observations have been associated to it, the server may reject certain updates to the schema (e.g., adding or removing result fields, changing UoM, etc.). Datastream schema evolution will be addressed in more details in a future revision, but the current workaround is to create a new datastream if the schema changes.

14.3. Observations

REQUIREMENT 66	
IDENTIFIER	`/req/create-replace-delete/observation`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Datastreams Observations”
A	The server SHALL support the CREATE operation at the `Observation` resources endpoints defined by the following URI template: • `{api_root}/datastreams/{dsId}/observations`
B	The server SHALL support the REPLACE and DELETE operations at the `Observation` resource endpoints defined by the following URI templates: • `{api_root}/datastreams/{dsId}/observations/{id}` • `{api_root}/observations/{id}`
C	The `dsId` parameter is the local identifier of the `DataStream` resource the `Observation` is (or will be) associated to. The `id` parameter is the local identifier of the `Observation` resource to replace or delete.

The following constraints must be implemented by the server.

REQUIREMENT 67	
IDENTIFIER	`/req/create-replace-delete/observation-schema`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Datastreams Observations”
A	The server SHALL reject an `Observation` CREATE or REPLACE request with HTTP status code 400 if the `Observation` representation is not valid with respect to the schema provided by the parent `DataStream` resource.

14.4. Control Streams

REQUIREMENT 68	
IDENTIFIER	`/req/create-replace-delete/controlstream`

REQUIREMENT 68	
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL support the CREATE operation at the `ControlStream` resources endpoints defined by the following URI template: • `{api_root}/systems/{sysId}/controlstreams`
B	The server SHALL support the REPLACE and DELETE operations at the `ControlStream` resource endpoints defined by the following URI templates: • `{api_root}/systems/{sysId}/controlstreams/{id}` • `{api_root}/controlstreams/{id}`
C	The `sysId` parameter is the local identifier of the `System` resource the `ControlStream` is (or will be) associated to. The `id` parameter is the local identifier of the `ControlStream` resource to replace or delete.

The following constraints must be implemented by the server.

REQUIREMENT 69	
IDENTIFIER	`/req/create-replace-delete/controlstream-update-schema`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL reject a REPLACE request on a `ControlStream` resource that modifies the command schema if the `ControlStream` already has nested `Command` resources. The server SHALL use HTTP status code 409 to report the error.

REQUIREMENT 70	
IDENTIFIER	`/req/create-replace-delete/controlstream-delete-cascade`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	By default, the server SHALL reject a DELETE request on a `ControlStream` resource that has nested `Command` resources. The server SHALL use HTTP status code 409 to report the error.
B	If the request contains the `cascade` parameter, the server SHALL accept the DELETE request and delete the `ControlStream` resource as well as all its nested `Command` resources.

14.5. Commands

REQUIREMENT 71	
IDENTIFIER	`/req/create-replace-delete/command`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL support the CREATE operation at the `Command` resources endpoints defined by the following URI template: • `{api_root}/controlstreams/{csId}/commands`
B	The server SHALL support the REPLACE and DELETE operations at the `Command` resource endpoints defined by the following URI templates: • `{api_root}/controlstreams/{csId}/commands/{id}` • `{api_root}/commands/{id}`
C	The `csId` parameter is the local identifier of the `ControlStream` resource the `Command` is (or will be) associated to. The `id` parameter is the local identifier of the `Command` resource to replace or delete.

The following constraints must be implemented by the server.

REQUIREMENT 72	
IDENTIFIER	`/req/create-replace-delete/command-schema`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL reject a `Command` CREATE or REPLACE request with HTTP status code 400 if the `Command` representation is not valid with respect to the schema provided by the parent `ControlStream` resource.

NOTE: Cancelling a command is different from deleting the `Command` resource with an HTTP DELETE request. When a command is cancelled, the `Command` resource remain on the server but its status is changed to `CANCELED` (and of course the command processing should be aborted whenever possible). Command cancellation is implemented by posting a new status report with status code `CANCELED` at the command status endpoint. See requirements for `CommandStatus` resources endpoints below.

14.6. Command Status

REQUIREMENT 73	
IDENTIFIER	`/req/create-replace-delete/command-status`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL support the CREATE operation at the `CommandStatus` resources endpoints defined by the following URI template: • `{api_root}/commands/{cmdId}/status`
B	The server SHALL support the REPLACE and DELETE operations at the `CommandStatus` resource endpoints defined by the following URI template: • `{api_root}/commands/{cmdId}/status/{id}`
C	The `cmdId` parameter is the local identifier of the `Command` resource the `CommandStatus` is (or will be) associated to. The `id` parameter is the local identifier of the `CommandStatus` resource to replace or delete.

14.7. Command Results

REQUIREMENT 74	
IDENTIFIER	`/req/create-replace-delete/command-result`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL support the CREATE operation at the `CommandResult` resources endpoints defined by the following URI template: • `{api_root}/commands/{cmdId}/result`
B	The server SHALL support the REPLACE and DELETE operations at the `CommandResult` resource endpoints defined by the following URI template: • `{api_root}/commands/{cmdId}/result/{id}`
C	The `cmdId` parameter is the local identifier of the `Command` resource the `CommandResult` is (or will be) associated to.

REQUIREMENT 74	
	The `id` parameter is the local identifier of the `CommandResult` resource to replace or delete.

14.8. Feasibility

REQUIREMENT 75	
IDENTIFIER	`/req/create-replace-delete/feasibility`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Command Feasibility”
A	The server SHALL support the CREATE operation at the `Command` resources endpoints defined by the following URI template: • `{api_root}/controlstreams/{csId}/feasibility`
B	The server SHALL support the REPLACE and DELETE operations at the `Command` resource endpoints defined by the following URI templates: • `{api_root}/controlstreams/{csId}/feasibility` • `{api_root}/feasibility/{id}`
C	The `csId` parameter is the local identifier of the `ControlStream` resource the `Feasibility` is (or will be) associated to. The `id` parameter is the local identifier of the `Feasibility` resource to replace or delete.

14.9. Feasibility Status

REQUIREMENT 76	
IDENTIFIER	`/req/create-replace-delete/feasibility-status`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Command Feasibility”
A	The server SHALL support the CREATE operation at the `CommandStatus` resources endpoints defined by the following URI template:

REQUIREMENT 76	
	• `{api_root}/feasibility/{feasId}/status`
B	The server SHALL support the REPLACE and DELETE operations at the `CommandStatus` resource endpoints defined by the following URI template: • `{api_root}/feasibility/{feasId}/status/{id}`
C	The `feasId` parameter is the local identifier of the `Feasibility` resource the `CommandStatus` is (or will be) associated to. The `id` parameter is the local identifier of the `CommandStatus` resource to replace or delete.

14.10. Feasibility Results

REQUIREMENT 77	
IDENTIFIER	`/req/create-replace-delete/feasibility-result`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`
CONDITIONS	The server implements Requirements Class “Command Feasibility”
A	The server SHALL support the CREATE operation at the `CommandResult` resources endpoints defined by the following URI template: • `{api_root}/feasibility/{feasId}/result`
B	The server SHALL support the REPLACE and DELETE operations at the `CommandResult` resource endpoints defined by the following URI template: • `{api_root}/feasibility/{feasId}/result/{id}`
C	The `feasId` parameter is the local identifier of the `Feasibility` resource the `CommandResult` is (or will be) associated to. The `id` parameter is the local identifier of the `CommandResult` resource to replace or delete.

14.11. System Events

REQUIREMENT 78	
IDENTIFIER	`/req/create-replace-delete/system-event`
INCLUDED IN	Requirements class 7: `/req/create-replace-delete`

REQUIREMENT 78	
CONDITIONS	The server implements Requirements Class “System Events”
A	The server SHALL support the CREATE operation at the `SystemEvent` resources endpoints defined by the following URI template: • `{api_root}/systems/{sysId}/events`
B	The server SHALL support the REPLACE and DELETE operations at the `SystemEvent` resource endpoints defined by the following URI template: • `{api_root}/systems/{sysId}/events/{id}` • `{api_root}/systemEvents/{id}`
C	The `sysId` parameter is the local identifier of the `System` resource the `SystemEvent` is (or will be) associated to. The `id` parameter is the local identifier of the `SystemEvent` resource to replace or delete.

15

REQUIREMENTS CLASS "UPDATE"

15 REQUIREMENTS CLASS "UPDATE"

15.1. Overview

REQUIREMENTS CLASS 8	
IDENTIFIER	`/req/update`
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.8: `/conf/update`
PREREQUISITES	Requirements class 7: `/req/create-replace-delete` http://www.opengis.net/spec/ogcapi-features-4/1.0/req/update
NORMATIVE STATEMENTS	Requirement 79: `/req/update/datastream` Requirement 80: `/req/update/datastream-update-schema` Requirement 81: `/req/update/observation` Requirement 82: `/req/update/observation-schema` Requirement 83: `/req/update/controlstream` Requirement 84: `/req/update/controlstream-update-schema` Requirement 85: `/req/update/command` Requirement 86: `/req/update/command-schema` Requirement 87: `/req/update/command-status` Requirement 88: `/req/update/command-result` Requirement 89: `/req/update/feasibility` Requirement 90: `/req/update/feasibility-status` Requirement 91: `/req/update/feasibility-result` Requirement 92: `/req/update/system-event`

The "Update" requirements class specifies how instances of the resource types defined in this Standard are updated (i.e., patched) via a CS API endpoint.

All resources are updated using the UPDATE (HTTP PATCH) operation, as defined by the OGC API — Features — Part 4: Create, Replace, Update and Delete Standard.

OGC API — Features — Part 4: Create, Replace, Update and Delete uses the terms "resources endpoint" and "resource endpoint" to identify the paths where these operations are supported by the server. The following sections provide these endpoints for each resource type defined by the CS API Standard.

15.2. DataStreams

REQUIREMENT 79	
IDENTIFIER	`/req/update/datastream`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Datastreams Observations”
A	The server SHALL support the UPDATE operation at the `DataStream` resources endpoints defined by the following URI templates: • `{api_root}/systems/{sysId}/datastreams/{id}` • `{api_root}/datastreams/{id}`
B	The `sysId` parameter is the local identifier of the `System` resource the `DataStream` is associated to. The `id` parameter is the local identifier of the `DataStream` resource to update.

The following constraints must be implemented by the server.

REQUIREMENT 80	
IDENTIFIER	`/req/update/datastream-update-schema`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Datastreams Observations”
A	The server SHALL reject an UPDATE request on a `DataStream` resource that modifies the observation schema if the `DataStream` already has nested `Observation` resources.

15.3. Observations

REQUIREMENT 81	
IDENTIFIER	`/req/update/observation`

REQUIREMENT 81	
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Datastreams Observations”
A	The server SHALL support the UPDATE operation at the `Observation` resource endpoints defined by the following URI templates: • `{api_root}/datastreams/{dsId}/observations/{id}` • `{api_root}/observations/{id}`
B	The `dsId` parameter is the local identifier of the `DataStream` resource the `Observation` is associated to. The `id` parameter is the local identifier of the `Observation` resource to update.

The following constraints must be implemented by the server.

REQUIREMENT 82	
IDENTIFIER	`/req/update/observation-schema`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Datastreams Observations”
A	The server SHALL reject an `Observation` UPDATE request with HTTP status code 400 if the `Observation` representation is not valid with respect to the schema provided by the parent `DataStream` resource.

15.4. Control Streams

REQUIREMENT 83	
IDENTIFIER	`/req/update/controlstream`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL support the UPDATE operation at the `ControlStream` resource endpoints defined by the following URI templates:

REQUIREMENT 83	
	• `{api_root}/systems/{sysId}/controlstreams/{id}` • `{api_root}/controlstreams/{id}`
B	The `sysId` parameter is the local identifier of the `System` resource the `ControlStream` is associated to. The `id` parameter is the local identifier of the `ControlStream` resource to update.

The following constraints must be implemented by the server.

REQUIREMENT 84	
IDENTIFIER	`/req/update/controlstream-update-schema`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL reject an UPDATE request on a `ControlStream` resource that modifies the command schema if the `ControlStream` already has nested `Command` resources.

15.5. Commands

REQUIREMENT 85	
IDENTIFIER	`/req/update/command`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL support the UPDATE operation at the `Command` resource endpoints defined by the following URI templates: • `{api_root}/controlstreams/{csId}/commands/{id}` • `{api_root}/commands/{id}`
B	The `csId` parameter is the local identifier of the `ControlStream` resource the `Command` is associated to. The `id` parameter is the local identifier of the `Command` resource to update.

The following constraints must be implemented by the server.

REQUIREMENT 86	
IDENTIFIER	`/req/update/command-schema`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL reject an UPDATE request on a `Command` resource with HTTP status code 400 if the `Command` representation is not valid with respect to the schema provided by the parent `ControlStream` resource.

15.6. Command Status

REQUIREMENT 87	
IDENTIFIER	`/req/update/command-status`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL support the UPDATE operation at the `CommandStatus` resources endpoints defined by the following URI template: • `{api_root}/commands/{cmdId}/status/{id}`
B	The `cmdId` parameter is the local identifier of the `Command` resource the `CommandStatus` is associated to. The `id` parameter is the local identifier of the `CommandStatus` resource to update.

15.7. Command Results

REQUIREMENT 88	
IDENTIFIER	`/req/update/command-result`

REQUIREMENT 88	
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Control Streams Commands”
A	The server SHALL support the UPDATE operation at the `CommandResult` resources endpoints defined by the following URI template: • `{api_root}/commands/{cmdId}/result/{id}`
B	The `cmdId` parameter is the local identifier of the `Command` resource the `CommandResult` is associated to. The `id` parameter is the local identifier of the `CommandResult` resource to update.

15.8. Feasibility

REQUIREMENT 89	
IDENTIFIER	`/req/update/feasibility`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Command Feasibility”
A	The server SHALL support the UPDATE operation at the `Command` resource endpoints defined by the following URI templates: • `{api_root}/controlstreams/{csId}/feasibility/{id}` • `{api_root}/feasibility/{id}`
B	The `csId` parameter is the local identifier of the `ControlStream` resource the `Command` is associated to. The `id` parameter is the local identifier of the `Feasibility` resource to update.

15.9. Feasibility Status

REQUIREMENT 90	
IDENTIFIER	`/req/update/feasibility-status`

REQUIREMENT 90	
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Command Feasibility”
A	The server SHALL support the UPDATE operation at the `CommandStatus` resources endpoints defined by the following URI template: • `{api_root}/feasibility/{feasId}/status/{id}`
B	The `feasId` parameter is the local identifier of the `Feasibility` resource the `CommandStatus` is associated to. The `id` parameter is the local identifier of the `CommandStatus` resource to update.

15.10. Feasibility Results

REQUIREMENT 91	
IDENTIFIER	`/req/update/feasibility-result`
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “Command Feasibility”
A	The server SHALL support the UPDATE operation at the `CommandResult` resources endpoints defined by the following URI template: • `{api_root}/feasibility/{feasId}/result/{id}`
B	The `feasId` parameter is the local identifier of the `Feasibility` resource the `CommandResult` is associated to. The `id` parameter is the local identifier of the `CommandResult` resource to update.

15.11. System Events

REQUIREMENT 92	
IDENTIFIER	`/req/update/system-event`

REQUIREMENT 92	
INCLUDED IN	Requirements class 8: `/req/update`
CONDITIONS	The server implements Requirements Class “System Events”
A	The server SHALL support the UPDATE operation at the `SystemEvent` resources endpoints defined by the following URI template: • `{api_root}/systems/{sysId}/events/{id}` • `{api_root}/systemEvents/{id}`
B	The `sysId` parameter is the local identifier of the `System` resource the `SystemEvent` is associated to. The `id` parameter is the local identifier of the `SystemEvent` resource to update.

16

REQUIREMENTS CLASSES FOR ENCODINGS

16 REQUIREMENTS CLASSES FOR ENCODINGS

16.1. Requirements Class "JSON Encoding"

16.1.1. Overview

REQUIREMENTS CLASS 9	
IDENTIFIER	`/req/json`
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.9: `/conf/json`
PREREQUISITE	http://www.opengis.net/spec/SWE/3.0/req/json-record-components
NORMATIVE STATEMENTS	Requirement 93: `/req/json/mediatype-read` Requirement 94: `/req/json/mediatype-write` Requirement 95: `/req/json/datastream-schema` Requirement 96: `/req/json/obsschema-schema` Requirement 97: `/req/json/observation-schema` Requirement 98: `/req/json/observation-constraints` Requirement 99: `/req/json/controlstream-schema` Requirement 100: `/req/json/commandschema-schema` Requirement 101: `/req/json/command-schema` Requirement 102: `/req/json/command-constraints` Requirement 103: `/req/json/commandstatus-schema` Requirement 104: `/req/json/commandresult-schema` Requirement 105: `/req/json/commandresult-constraints`

REQUIREMENTS CLASS 9	
	Requirement 106: `/req/json/systemevent-schema`

This requirements class defines general JSON encodings for all resource types defined in part 2.

16.1.2. Media Type

The media type used to advertise support for this encoding is `application/json`.

REQUIREMENT 93	
IDENTIFIER	`/req/json/mediatype-read`
INCLUDED IN	Requirements class 9: `/req/json`
A	The server SHALL accept resource retrieval (read) requests with media type `application/json` for all resource types whose representation is specified in this requirements class.
B	The response to such request SHALL be encoded as specified in the clause corresponding to the resource type.

REQUIREMENT 94	
IDENTIFIER	`/req/json/mediatype-write`
INCLUDED IN	Requirements class 9: `/req/json`
CONDITIONS	The server implements Requirements Class “Create/Replace/Delete”.
A	The server SHALL accept resource insertion (write) requests with media type `application/json` for all resource types whose representation is specified in this requirements class.
B	The resource representation provided in the request SHALL be encoded as specified in the clause corresponding to the resource type.

16.1.3. DataStream Representation

REQUIREMENT 95	
IDENTIFIER	`/req/json/datastream-schema`
INCLUDED IN	Requirements class 9: `/req/json`
A	A JSON document containing a single `DataStream` resource SHALL be valid against the JSON schema dataStream.json.
B	A JSON document containing a collection of `DataStream` resources SHALL be valid against the JSON schema dataStreamCollection.json.

Example — A Datastream in JSON format: This is a simple datastream with a single observed property.

```
{
  "id": "958tf25kjm2f6",
  "name": "Indoor Thermometer 001 - Living Room Temperature",
  "outputName": "temp",
  "system@link": {
    "href": "https://data.example.org/api/systems/123",
    "uid": "urn:x-ogc:systems:001"
  },
  "featureOfInterest@link": {
    "href": "https://data.example.org/api/collections/buildings/items/754",
    "title": "My House"
  },
  "samplingFeature@link": {
    "href": "https://data.example.org/api/samplingFeatures/4478",
    "title": "Thermometer Sampling Point"
  },
  "phenomenonTime": [
    "2020-06-29T14:32:00Z",
    "2022-06-29T19:37:00Z"
  ],
  "resultTime": [
    "2020-06-29T14:32:00Z",
    "2012-06-29T19:37:00Z"
  ],
  "observedProperties": [
    {
      "definition": "http://mmisw.org/ont/cf/parameter/air_temperature",
      "label": "Room Temperature",
      "description": "Ambient air temperature measured inside the room"
    }
  ],
  "resultType": "measure",
  "formats": [
    "application/json",
    "application/swe+json",
    "application/swe+csv",
    "application/x-protobuf"
  ],
  "live": true,
  "links": [
    {
      "rel" : "observations",
      "href" : "https://data.example.org/api/datastreams/958tf25kjm2f6/
observations",
```

```
      "type" : "application/json"
    }
  ]
}
```

16.1.4. Observation Schema Representation

When using the `application/json` media type for observations, two separate schemas are provided to further describe the content of the observation `result` and `parameters` properties (the `parameters` schema is optional). Both schemas are provided as SWE Common data component tree in JSON format.

REQUIREMENT 96	
IDENTIFIER	`/req/json/obsschema-schema`
INCLUDED IN	Requirements class 9: `/req/json`
A	The `Observation Schema` resource for media type `application/json` SHALL be valid against the JSON schema observationSchemaJson.json.

Example — Example Observation Schemas for the JSON format: This is an example schema for scalar observations:

```
{
  "obsFormat": "application/json",
  "resultSchema": {
    "name": "temp",
    "type": "Quantity",
    "definition": "http://mmisw.org/ont/cf/parameter/air_temperature",
    "label": "Room Temperature",
    "description": "Ambient air temperature measured inside the room",
    "uom": {
      "code": "Cel"
    },
    "nilValues": [
      { "reason": "http://www.opengis.net/def/nil/OGC/0/missing", "value": "NaN"
},
      { "reason": "http://www.opengis.net/def/nil/OGC/0/BelowDetectionRange",
"value": "-Infinity" },
      { "reason": "http://www.opengis.net/def/nil/OGC/0/AboveDetectionRange",
"value": "+Infinity" }
    ]
  }
}
```

This is an example schema for vector observations:

```
{
  "obsFormat": "application/json",
  "resultSchema": {
    "name": "location",
    "type": "Vector",
    "label": "Platform Location",
```

```
    "definition": "http://sensorml.com/ont/swe/property/Location",
    "referenceFrame": "http://www.opengis.net/def/crs/EPSG/0/4979",
    "coordinates": [
      {
        "name": "lat",
        "type": "Quantity",
        "definition": "http://sensorml.com/ont/swe/property/GeodeticLatitude",
        "axisID": "Lat",
        "label": "Geodetic Latitude",
        "uom": {
          "code": "deg"
        }
      },
      {
        "name": "lon",
        "type": "Quantity",
        "definition": "http://sensorml.com/ont/swe/property/Longitude",
        "axisID": "Lon",
        "label": "Longitude",
        "uom": {
          "code": "deg"
        }
      },
      {
        "name": "h",
        "type": "Quantity",
        "definition": "http://sensorml.com/ont/swe/property/
HeightAboveEllipsoid",
        "axisID": "h",
        "label": "Ellipsoidal Height",
        "uom": {
          "code": "m"
        }
      }
    ]
  }
}
```

This third example has an out-of-band PNG image as the result:

```
{
    "obsFormat": "application/json",
    "resultLink": {
        "mediaType": "image/png"
    }
  }
```

NOTE: All other observation properties are the same for all datastreams and thus described in the static schema provided in Clause 16.1.5.

16.1.5. Observation Representation

REQUIREMENT 97

IDENTIFIER `/req/json/observation-schema`

REQUIREMENT 97	
INCLUDED IN	Requirements class 9: `/req/json`
A	A JSON document containing a single `Observation` resource SHALL be valid against the JSON schema observation.json.
B	A JSON document containing a collection of `Observation` resources SHALL be valid against the JSON schema observationCollection.json.

REQUIREMENT 98	
IDENTIFIER	`/req/json/observation-constraints`
INCLUDED IN	Requirements class 9: `/req/json`
STATEMENT	The following constraints apply to `Observation` resources:
A	The value of the `phenomenonTime` and `resultTime` properties SHALL be expressed in the UTC time scale, with an optional time offset.
B	The value of the `result` property SHALL be encoded according to the schema of the parent `DataStream`. The schema is provided by the `resultSchema` property of the schema resource.
C	The value of the `parameters` property SHALL be encoded according to the `parametersSchema` of the parent `DataStream`.
D	See observationSchemaJson.json.

Example — Observations in JSON format: This is a simple observation with a scalar observed property, associated to the datastream of the example above.

```
{
  "id": "1h6pmb3ntfmogfppknk9aefpvs",
  "datastream@id": "958tf25kjm2f6",
  "phenomenonTime": "2021-03-15T04:53:34Z",
  "resultTime": "2021-03-15T04:53:34Z",
  "result": 23.5
}
```

This second observation has a vector result type:

```
{
  "id": "1125alnna75hafppknk9aefpvs",
  "datastream@id": "1vf8i5ois38u8",
  "phenomenonTime": "2021-03-15T04:53:34Z",
  "resultTime": "2021-03-15T04:53:34Z",
  "result": {
    "lat": -86.5861,
    "lon": 34.7304,
    "alt": 183
  }
```

```
}
```

This third observation has PNG image as a result type that is a PNG image encoded inline as a `data:` URL

```
{
  "id": "fefaig45w46v5186d6w",
  "datastream@id": "f44f85rrt",
  "foi@id": "55f48g48th",
  "phenomenonTime": "2023-04-03T18:45:23Z",
  "resultTime": "2023-04-03T18:45:23Z",
  "result@link": {
    "href": "",
    "title": "Inline PNG image",
    "type": "image/png"
  }
}
```

16.1.6. ControlStream Representation

REQUIREMENT 99	
IDENTIFIER	`/req/json/controlstream-schema`
INCLUDED IN	Requirements class 9: `/req/json`
A	A JSON document containing a single `ControlStream` resource SHALL be valid against the JSON schema controlStream.json.
B	A JSON document containing a collection of `ControlStream` resources SHALL be valid against the JSON schema controlStreamCollection.json.

Example — A Control Stream in JSON format: This is a simple control stream for a camera accepting PTZ commands in JSON format.

```
{
  "id": "hf62t0dotfd5k",
  "name": "Garage Video Camera 001 - PTZ Control",
  "inputName": "ptz",
  "system@link": {
    "href": "https://data.example.org/api/systems/4722256",
    "uid": "urn:x-ogc:systems:CAM001",
    "title": "Garage Video Camera 001"
  },
  "issueTime": [
    "2012-06-29T14:32:34Z",
    "2012-06-29T14:37:34Z"
  ],
  "executionTime": [
    "2012-06-29T14:32:34Z",
    "2012-06-29T14:37:34Z"
  ],
  "controlledProperties": [
    {
```

```
      "definition": "http://sensorml.com/ont/swe/property/PanAngle",
      "label": "Pan Angle"
    },
    {
      "definition": "http://sensorml.com/ont/swe/property/TiltAngle",
      "label": "Tilt Angle"
    },
    {
      "definition": "http://sensorml.com/ont/swe/property/ZoomFactor",
      "label": "Zoom Factor"
    }
  ],
  "formats": [
    "application/json"
  ],
  "live": true,
  "async": false,
  "links": [
    {
      "rel": "commands",
      "href": "https://data.example.org/api/controls/hf62t0dotfd5k/commands"
    }
  ]
}
```


16.1.7. Command Schema Representation

When using the `application/json` media type for commands, two separate schemas are provided to further describe the content of the `parameters` and `result` properties (the `result` schema is optional). Both schemas are provided as SWE Common data component tree in JSON format.

REQUIREMENT 100	
IDENTIFIER	`/req/json/commandschema-schema`
INCLUDED IN	Requirements class 9: `/req/json`
A	The `Command Schema` resource for media type `application/json` SHALL be valid against the JSON schema commandSchemaJson.json.

Example — Example Command Schemas for the JSON format: This is an example schema for PTZ commands:


```
{
  "commandFormat": "application/json",
  "parametersSchema": {
    "type": "DataRecord",
    "fields": [
      {
        "name": "pan",
        "type": "Quantity",
        "definition": "http://sensorml.com/ont/swe/property/PanAngle",
        "label": "Pan Angle",
```

```
        "description": "Rotation of the camera around its vertical axis (i.e.,
causing the image to translate along its horizontal axis)",
        "uom": {
          "code": "deg"
        }
      },
      {
        "name": "tilt",
        "type": "Quantity",
        "definition": "http://sensorml.com/ont/swe/property/PanAngle",
        "label": "Pan Angle",
        "description": "Rotation of the camera around its horizontal axis (i.e.,
causing the image to translate along its vertical axis)",
        "uom": {
          "code": "deg"
        }
      },
      {
        "name": "zoom",
        "type": "Quantity",
        "definition": "http://sensorml.com/ont/swe/property/ZoomFactor",
        "label": "Zoom Factor",
        "description": "Amount of zoom, 0 being the highest FOV and 100 being
the lowest",
        "uom": {
          "code": "%"
        }
      }
    ]
  }
}
```

NOTE: All other command properties are the same for all control streams and thus described in the static schema provided in Clause 16.1.8.

16.1.8. Command Representation

REQUIREMENT 101	
IDENTIFIER	`/req/json/command-schema`
INCLUDED IN	Requirements class 9: `/req/json`
A	A JSON document containing a single `Command` resource SHALL be valid against the JSON schema command.json.
B	A JSON document containing a collection of `Command` resources SHALL be valid against the JSON schema commandCollection.json.

REQUIREMENT 102	
IDENTIFIER	`/req/json/command-constraints`
INCLUDED IN	Requirements class 9: `/req/json`
STATEMENT	The following constraints apply to `Command` resources:
A	The value of the `issueTime` and `executionTime` properties SHALL be expressed in the UTC time scale, with an optional time offset.
B	The value of the `parameters` property SHALL be encoded according to the schema of the parent `ControlStream`. The schema is provided by the `parametersSchema` property of the schema resource.
C	See commandSchemaJson.json.

Example — Command in JSON format: This is an example command used to task a PTZ camera, encoded in JSON format:

```
{
  "id": "1125alnna75hafppknk9aefpvs",
  "controlstream@id": "hf62t0dotfd5k",
  "sender": "user01",
  "issueTime": "2021-03-15T04:53:34.248Z",
  "executionTime": [
    "2021-03-15T04:53:34.543Z",
    "2021-03-15T04:53:36.021Z"
  ],
  "currentStatus": "COMPLETED",
  "parameters": {
    "pan": -10.0,
    "tilt": 23.0,
    "zoom": 0.4
  }
}
```

16.1.9. Command Status Representation

REQUIREMENT 103	
IDENTIFIER	`/req/json/commandstatus-schema`
INCLUDED IN	Requirements class 9: `/req/json`
A	A JSON document containing a single `CommandStatus` resource SHALL be valid against the JSON schema commandStatus.json.

REQUIREMENT 103	
B	A JSON document containing a collection of `CommandStatus` resources SHALL be valid against the JSON schema commandStatusCollection.json.

Example — Command Status in JSON format: These are example command status reports, encoded in JSON format:

```
{
  "id": "rlg2905142qs5uvish4vktotffds2iss8aa0a00",
  "command@id": "1125alnna75hafppknk9aefpvs",
  "reportTime": "2021-03-15T04:53:34.348Z",
  "statusCode": "ACCEPTED"
}

{
  "id": "155rufq7aplr8id10839fc8d6u0ulqfu1bjfumo",
  "command@id": "1125alnna75hafppknk9aefpvs",
  "reportTime": "2021-03-15T04:53:36.021Z",
  "statusCode": "COMPLETED",
  "message": "Camera moved to new position"
}
```

16.1.10. Command Result Representation

REQUIREMENT 104	
IDENTIFIER	`/req/json/commandresult-schema`
INCLUDED IN	Requirements class 9: `/req/json`
A	A JSON document containing a single `CommandResult` resource SHALL be valid against the JSON schema commandResult.json.
B	A JSON document containing a collection of `CommandResult` resources SHALL be valid against the JSON schema commandResultCollection.json.

REQUIREMENT 105	
IDENTIFIER	`/req/json/commandresult-constraints`
INCLUDED IN	Requirements class 9: `/req/json`
A	If a `CommandResult` resource includes inline data, the content of the `data` property SHALL be encoded according to the schema provided by the parent `ControlStream`:
B	For regular commands, the schema is provided by the the `resultSchema` property of the schema resource.

REQUIREMENT 105	
C	For feasibility requests (i.e., commands received on a feasibility channel), the schema is provided by the the `feasibilityResultSchema` property of the schema resource.
D	See commandSchemaJson.json.

Example — Command Result in JSON format: These are example command results, encoded in JSON format:

```
{
  "data": {
    "mean": "10.51",
    "stdev": "1.23"
  }
}

{
  "observation@link": {
    "href": "https://data.example.org/api/observations/
gss45sdf413s387g49445ssdf55?f=json",
    "title": "Satellite Image",
    "type": "application/json"
  }
}

{
  "datastream@link": {
    "href": "https://data.example.org/api/datastreams/445ssdf55",
    "title": "Plume Simulation Data",
    "type": "application/json"
  }
}
```

16.1.11. System Event Representation

REQUIREMENT 106	
IDENTIFIER	`/req/json/systemevent-schema`
INCLUDED IN	Requirements class 9: `/req/json`
A	A JSON document containing a single `SystemEvent` resource SHALL be valid against the JSON schema systemEvent.json.
B	A JSON document containing a collection of `SystemEvent` resources SHALL be valid against the JSON schema systemEventCollection.json.

16.2. Requirements Class “SWE Common JSON Encoding”

REQUIREMENTS CLASS 10	
IDENTIFIER	`/req/swecommon-json`
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.10: `/conf/swecommon-json`
PREREQUISITE	http://www.opengis.net/spec/SWE/3.0/req/json-encoding-rules
NORMATIVE STATEMENTS	Requirement 107: `/req/swecommon-json/mediatype-read` Requirement 108: `/req/swecommon-json/mediatype-write` Requirement 109: `/req/swecommon-json/obsschema-schema` Requirement 110: `/req/swecommon-json/obsschema-mapping` Requirement 111: `/req/swecommon-json/observation-encoding` Requirement 112: `/req/swecommon-json/cmdschema-schema` Requirement 113: `/req/swecommon-json/cmdschema-mapping` Requirement 114: `/req/swecommon-json/command-encoding`

16.2.1. Overview

This requirements class defines JSON encodings for `Observation` and `Command` resources based on SWE Common 3.0.

16.2.2. Media Type

NOTE: Implementations should use **`application/vnd.ogc.swe+json`** as a preliminary media type until the SWE Common 3.0 Standard is stable to avoid confusing future implementations accessing JSON documents from draft versions of the Standard. The media type `application/`

`swe+json` will be registered for SWE Common JSON encoding, if and once this Standard is approved by the OGC Members. This note will be removed before publishing this Standard.

The media type used when using the SWE Common JSON encoding is **`application/swe+json`**.

REQUIREMENT 107	
IDENTIFIER	`/req/swecommon-json/mediatype-read`
INCLUDED IN	Requirements class 10: `/req/swecommon-json`
A	The server SHALL accept resource retrieval (read) requests with media type `application/swe+json` for all resource types whose representation is specified in this requirements class.
B	The response to such request SHALL be encoded as specified in the clause corresponding to the resource type.

REQUIREMENT 108	
IDENTIFIER	`/req/swecommon-json/mediatype-write`
INCLUDED IN	Requirements class 10: `/req/swecommon-json`
CONDITIONS	The server implements Requirements Class “Create/Replace/Delete”.
A	The server SHALL accept resource insertion (write) requests with media type `application/swe+json` for all resource types whose representation is specified in this requirements class.
B	The resource representation provided in the request SHALL be encoded as specified in the clause corresponding to the resource type.

16.2.3. Observation Schema Representation

The observation schema for the `application/swe+json` media type is a SWE Common data component tree provided in JSON format.

REQUIREMENT 109	
IDENTIFIER	`/req/swecommon-json/obsschema-schema`
INCLUDED IN	Requirements class 10: `/req/swecommon-json`

REQUIREMENT 109	
A	The `Observation Schema` resource for media type `application/swe+json` SHALL be valid against the JSON schema observationSchemaSwe.json.
B	The `encoding` property SHALL be set to a `JSONEncoding` object.

REQUIREMENT 110	
IDENTIFIER	`/req/swecommon-json/obsschema-mapping`
INCLUDED IN	Requirements class 10: `/req/swecommon-json`

A	The `recordSchema` property SHALL include at least one `Time` component corresponding to either `resultTime` or `phenomenonTime`. This `Time` component SHALL be identified using one of the following URIs as the `definition` property.
	For `phenomenonTime`:
	`http://www.w3.org/ns/sosa/phenomenonTime`, or
	`http://www.opengis.net/def/property/OGC/0/SamplingTime`
	For `resultTime`:
	`http://www.w3.org/ns/sosa/resultTime`

B	If the `recordSchema` property includes a reference to a sampling feature, a `Text` component SHALL be used. The component SHALL be identified using the following URI as the `definition` property: `http://www.w3.org/ns/sosa/FeatureOfInterest` The value of the component SHALL be the local identifier of the `SamplingFeature` resource.

16.2.4. Observation Representation

REQUIREMENT 111	
IDENTIFIER	`/req/swecommon-json/observation-encoding`
INCLUDED IN	Requirements class 10: `/req/swecommon-json`
A	`Observation` resources SHALL be encoded according to the schema provided by the parent `DataStream`, using the encoding rules defined in Clause 10.2: Requirements Class: JSON Encoding Rules of SWE Common 3.0.

16.2.5. Command Schema Representation

The command schema for the `application/swe+json` media type is a SWE Common data component tree provided in JSON format.

REQUIREMENT 112	
IDENTIFIER	`/req/swecommon-json/cmdschema-schema`
INCLUDED IN	Requirements class 10: `/req/swecommon-json`
A	The `Command Schema` resource for media type `application/swe+json` SHALL be valid against the JSON schema commandSchemaSwe.json.
B	The `encoding` property SHALL be set to a `JSONEncoding` object.

REQUIREMENT 113	
IDENTIFIER	`/req/swecommon-json/cmdschema-mapping`
INCLUDED IN	Requirements class 10: `/req/swecommon-json`
A	If the `recordSchema` property includes a timestamp to be mapped to the `issueTime` property of the `CommandResource`, a `Time` component SHALL be used. The component SHALL be identified using the following URI as the `definition` property: `http://www.opengis.net/def/property/OGC/0/IssueTime`
B	If the `recordSchema` property includes a reference to a sampling feature, a `Text` component SHALL be used. The component SHALL be identified using the following URI as the `definition` property: `http://www.w3.org/ns/sosa/FeatureOfInterest` The value of the component SHALL be the local identifier of the `SamplingFeature` resource.

16.2.6. Command Representation

REQUIREMENT 114	
IDENTIFIER	`/req/swecommon-json/command-encoding`
INCLUDED IN	Requirements class 10: `/req/swecommon-json`

REQUIREMENT 114	
A	`Command` resources SHALL be encoded according to the schema provided by the parent `ControlStream`, using the encoding rules defined in Clause 10.2: Requirements Class: JSON Encoding Rules of SWE Common 3.0.

16.3. Requirements Class “SWE Common Text Encoding”

REQUIREMENTS CLASS 11	
IDENTIFIER	`/req/swecommon-text`
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.11: `/conf/swecommon-text`
PREREQUISITE	http://www.opengis.net/spec/SWE/3.0/req/text-encoding-rules
NORMATIVE STATEMENTS	Requirement 115: `/req/swecommon-text/mediatype-read` Requirement 116: `/req/swecommon-text/mediatype-write` Requirement 117: `/req/swecommon-text/obsschema-schema` Requirement 118: `/req/swecommon-text/obsschema-mapping` Requirement 119: `/req/swecommon-text/observation-encoding` Requirement 120: `/req/swecommon-text/cmdschema-schema` Requirement 121: `/req/swecommon-text/cmdschema-mapping` Requirement 122: `/req/swecommon-text/command-encoding`

16.3.1. Overview

This requirements class defines text encodings (delimiter separated values, or DSV) for `Observation` and `Command` resources based on the Text Encoding defined in the SWE Common 3.0 Standard.

16.3.2. Media Type

NOTE: Implementations should use **`application/vnd.ogc.swe+text`** as a preliminary media type until the SWE Common 3.0 Standard is stable to avoid confusing future implementations accessing JSON documents from draft versions of the Standard. The media type `application/swe+text` will be registered for SWE Common Text encoding, if and once this Standard is approved by the OGC Members. This note will be removed before publishing this Standard.

The media type used when using the SWE Common Text encoding is **`application/swe+text`**.

REQUIREMENT 115	
IDENTIFIER	`/req/swecommon-text/mediatype-read`
INCLUDED IN	Requirements class 11: `/req/swecommon-text`
A	The server SHALL accept resource retrieval (read) requests with media type `application/swe+text` for all resource types whose representation is specified in this requirements class.
B	The response to such request SHALL be encoded as specified in the clause corresponding to the resource type.

REQUIREMENT 116	
IDENTIFIER	`/req/swecommon-text/mediatype-write`
INCLUDED IN	Requirements class 11: `/req/swecommon-text`
CONDITIONS	The server implements Requirements Class “Create/Replace/Delete”.
A	The server SHALL accept resource insertion (write) requests with media type `application/swe+text` for all resource types whose representation is specified in this requirements class.
B	The resource representation provided in the request SHALL be encoded as specified in the clause corresponding to the resource type.

16.3.3. Observation Schema Representation

The observation schema for the `application/swe+text` media type is a SWE Common data component tree provided in JSON format.

REQUIREMENT 117	
IDENTIFIER	/req/swecommon-text/obsschema-schema
INCLUDED IN	Requirements class 11: /req/swecommon-text
A	The Observation Schema resource for media type application/swe+text SHALL be valid against the JSON schema observationSchemaSwe.json.
B	The encoding property SHALL be set to a TextEncoding object.

REQUIREMENT 118	
IDENTIFIER	/req/swecommon-text/obsschema-mapping
INCLUDED IN	Requirements class 11: /req/swecommon-text
STATEMENT	The recordSchema property SHALL fulfill Requirement 110: /req/swecommon-json/obsschema-mapping.

16.3.4. Observation Representation

REQUIREMENT 119	
IDENTIFIER	/req/swecommon-text/observation-encoding
INCLUDED IN	Requirements class 11: /req/swecommon-text
A	Observation resources SHALL be encoded according to the schema provided by the parent DataStream, using the encoding rules defined in Clause 10.3: Requirements Class: Text Encoding Rules of SWE Common 3.0.

16.3.5. Command Schema Representation

The command schema for the application/swe+text media type is a SWE Common data component tree provided in JSON format.

REQUIREMENT 120	
IDENTIFIER	/req/swecommon-text/cmdschema-schema

REQUIREMENT 120	
INCLUDED IN	Requirements class 11: `/req/swecommon-text`
A	The `Command Schema` resource for media type `application/swe+text` SHALL be valid against the JSON schema commandSchemaSwe.json.
B	The `encoding` property SHALL be set to a `TextEncoding` object.

REQUIREMENT 121	
IDENTIFIER	`/req/swecommon-text/cmdschema-mapping`
INCLUDED IN	Requirements class 11: `/req/swecommon-text`
STATEMENT	The `recordSchema` property SHALL fulfill Requirement 113: `/req/swecommon-json/cmdschema-mapping`.

16.3.6. Command Representation

REQUIREMENT 122	
IDENTIFIER	`/req/swecommon-text/command-encoding`
INCLUDED IN	Requirements class 11: `/req/swecommon-text`
A	`Command` resources SHALL be encoded according to the schema provided by the parent `ControlStream`, using the encoding rules defined in Clause 10.3: Requirements Class: Text Encoding Rules of SWE Common 3.0.

16.4. Requirements Class “SWE Common Binary Encoding”

REQUIREMENTS CLASS 12	
IDENTIFIER	`/req/swecommon-binary`

REQUIREMENTS CLASS 12	
TARGET TYPE	Web API
CONFORMANCE CLASS	Conformance class A.12: `/conf/swecommon-binary`
PREREQUISITE	http://www.opengis.net/spec/SWE/3.0/req/binary-encoding-rules
NORMATIVE STATEMENTS	Requirement 123: `/req/swecommon-binary/mediatype-read` Requirement 124: `/req/swecommon-binary/mediatype-write` Requirement 125: `/req/swecommon-binary/obsschema-schema` Requirement 126: `/req/swecommon-binary/obsschema-mapping` Requirement 127: `/req/swecommon-binary/observation-encoding` Requirement 128: `/req/swecommon-binary/cmdschema-schema` Requirement 129: `/req/swecommon-binary/cmdschema-mapping` Requirement 130: `/req/swecommon-binary/command-encoding`

16.4.1. Overview

This requirements class defines binary encodings of `Observation` and `Command` resources based on the Binary Encoding defined in the SWE Common 3.0 Standard.

The main objective of this encoding is better data size efficiency than text or JSON, thus allowing for:

- Transfer of observations and commands over very low power / low bandwidth network links (e.g., LoRa, etc.); and
- Transfer high bandwidth data sets such as raster data (e.g., video, LiDAR, etc.).

For even better efficiency, this encoding can be combined with a transport protocol such as MQTT.

16.4.2. Media Type

NOTE: Implementations should use **`application/vnd.ogc.swe+binary`** as a preliminary media type until the SWE Common 3.0 Standard is stable to avoid confusing future implementations accessing JSON documents from draft versions of the Standard. The media type `application/`

`swe+binary` will be registered for SWE Common binary encoding, if and once this Standard is approved by the OGC Members. This note will be removed before publishing this Standard.

The media type used when using the SWE Common Text encoding is **`application/swe+binary`**.

REQUIREMENT 123	
IDENTIFIER	`/req/swecommon-binary/mediatype-read`
INCLUDED IN	Requirements class 12: `/req/swecommon-binary`
A	The server SHALL accept resource retrieval (read) requests with media type `application/swe+binary` for all resource types whose representation is specified in this requirements class.
B	The response to such request SHALL be encoded as specified in the clause corresponding to the resource type.

REQUIREMENT 124	
IDENTIFIER	`/req/swecommon-binary/mediatype-write`
INCLUDED IN	Requirements class 12: `/req/swecommon-binary`
CONDITIONS	The server implements Requirements Class “Create/Replace/Delete”.
A	The server SHALL accept resource insertion (write) requests with media type `application/swe+binary` for all resource types whose representation is specified in this requirements class.
B	The resource representation provided in the request SHALL be encoded as specified in the clause corresponding to the resource type.

16.4.3. Observation Schema Representation

The observation schema for the `application/swe+binary` media type is a SWE Common data component tree provided in JSON format.

REQUIREMENT 125	
IDENTIFIER	`/req/swecommon-binary/obsschema-schema`
INCLUDED IN	Requirements class 12: `/req/swecommon-binary`

REQUIREMENT 125	
A	The `Observation Schema` resource for media type `application/swe+binary` SHALL be valid against the JSON schema observationSchemaSwe.json.
B	The `encoding` property SHALL be set to a `BinaryEncoding` object.

REQUIREMENT 126	
IDENTIFIER	`/req/swecommon-binary/obsschema-mapping`
INCLUDED IN	Requirements class 12: `/req/swecommon-binary`
STATEMENT	The `recordSchema` property SHALL fulfill Requirement 110: `/req/swecommon-json/obsschema-mapping`.

16.4.4. Observation Representation

REQUIREMENT 127	
IDENTIFIER	`/req/swecommon-binary/observation-encoding`
INCLUDED IN	Requirements class 12: `/req/swecommon-binary`
A	`Observation` resources SHALL be encoded according to the schema provided by the parent `DataStream`, using the encoding rules defined in Clause 10.4: Requirements Class: Binary Encoding Rules of SWE Common 3.0.

16.4.5. Command Schema Representation

The command schema for the `application/swe+binary` media type is a SWE Common data component tree provided in JSON format.

REQUIREMENT 128	
IDENTIFIER	`/req/swecommon-binary/cmdschema-schema`
INCLUDED IN	Requirements class 12: `/req/swecommon-binary`

REQUIREMENT 128	
A	The `Command Schema` resource for media type `application/swe+binary` SHALL be valid against the JSON schema commandSchemaSwe.json.
B	The `encoding` property SHALL be set to a `BinaryEncoding` object.

REQUIREMENT 129	
IDENTIFIER	`/req/swecommon-binary/cmdschema-mapping`
INCLUDED IN	Requirements class 12: `/req/swecommon-binary`
STATEMENT	The `recordSchema` property SHALL fulfill Requirement 113: `/req/swecommon-json/cmdschema-mapping`.

16.4.6. Command Representation

REQUIREMENT 130	
IDENTIFIER	`/req/swecommon-binary/command-encoding`
INCLUDED IN	Requirements class 12: `/req/swecommon-binary`
A	`Command` resources SHALL be encoded according to the schema provided by the parent `ControlStream`, using the encoding rules defined in Clause 10.4: Requirements Class: Binary Encoding Rules of SWE Common 3.0.

A

ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE

ANNEX A (NORMATIVE) CONFORMANCE CLASS ABSTRACT TEST SUITE

A.1. Conformance Class “Common”

CONFORMANCE CLASS A.1	
IDENTIFIER	`/conf/api-common`
REQUIREMENTS CLASS	Requirements class 1: `/req/api-common`
PREREQUISITE	http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.1: `/conf/api-common/resources` Abstract test A.2: `/conf/api-common/resource-collection`

ABSTRACT TEST A.1	
IDENTIFIER	`/conf/api-common/resources`
REQUIREMENT	Requirement 1: `/req/api-common/resources`
TEST PURPOSE	No test required for this requirement as it is tested in other classes.

ABSTRACT TEST A.2	
IDENTIFIER	`/conf/api-common/resource-collection`
REQUIREMENT	Requirement 2: `/req/api-common/resource-collection`
TEST PURPOSE	Verify that resource collections are implemented like feature collections.
TEST METHOD	1. For each resource collection available on the server with the property `itemType` that is NOT set to `feature`: a) Execute the following tests from OGC API — Features — Part 1: Core: • /conf/core/fc-md-links • /conf/core/fc-md-items • /conf/core/fc-md-items-links • /conf/core/fc-md-extent • /conf/core/sfc-md-op • /conf/core/sfc-md-success • /conf/core/fc-op • /conf/core/fc-limit-definition • /conf/core/fc-limit-response • /conf/core/query-param-invalid • /conf/core/query-param-unknown • /conf/core/fc-links • /conf/core/fc-timeStamp • /conf/core/fc-numberMatched • /conf/core/fc-numberReturned • /conf/core/f-op • /conf/core/f-success • /conf/core/f-links

A.2. Conformance Class “Datastreams & Observations”

CONFORMANCE CLASS A.2	
IDENTIFIER	`/conf/datastream`

CONFORMANCE CLASS A.2	
REQUIREMENTS CLASS	Requirements class 2: `/req/datastream`
PREREQUISITE	Conformance class A.1: `/conf/api-common`
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.3: `/conf/datastream/sf-ref-from-datastream` Abstract test A.4: `/conf/datastream/foi-ref-from-datastream` Abstract test A.5: `/conf/datastream/canonical-url` Abstract test A.6: `/conf/datastream/resources-endpoint` Abstract test A.7: `/conf/datastream/canonical-endpoint` Abstract test A.8: `/conf/datastream/ref-from-system` Abstract test A.9: `/conf/datastream/ref-from-deployment` Abstract test A.10: `/conf/datastream/collections` Abstract test A.11: `/conf/datastream/schema-op` Abstract test A.12: `/conf/datastream/obs-canonical-url` Abstract test A.13: `/conf/datastream/obs-resources-endpoint` Abstract test A.14: `/conf/datastream/obs-canonical-endpoint` Abstract test A.15: `/conf/datastream/obs-ref-from-datastream` Abstract test A.16: `/conf/datastream/obs-collections`

ABSTRACT TEST A.3	
IDENTIFIER	`/conf/datastream/sf-ref-from-datastream`
REQUIREMENT	Requirement 3: `/req/datastream/sf-ref-from-datastream`
TEST PURPOSE	Validate that `Sampling Features` associated to a given datastream are available as sub-resources.
TEST METHOD	1. Retrieve all `DataStream` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=datastreams`.

ABSTRACT TEST A.3

2. For each `DataStream` resource in the response:
   a) Validate that the server implements an Sampling Features resources endpoint at path `{api_root}/datastreams/{dsId}/samplingFeatures` using test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/sf/resources-endpoint, where `dsId` is the local ID of the `DataStream` resource.

ABSTRACT TEST A.4

IDENTIFIER	`/conf/datastream/foi-ref-from-datastream`
REQUIREMENT	Requirement 4: `/req/datastream/foi-ref-from-datastream`
TEST PURPOSE	Validate that `Features of Interest` associated to a given datastream are available as sub-resources.
TEST METHOD	1. Retrieve all `DataStream` resources by executing test {part1-spec}/conf/api-common/canonical-resources with parameter `resource-type=datastreams`. 2. For each `DataStream` resource in the response: a) Issue an HTTP GET request at path `{api_root}/datastreams/{dsId}/featuresOfInterest`, where `dsId` is the local ID of the `DataStream` resource. b) Validate that a document was returned with a status code 200. c) Iterate through the list of resources in the response, following `next` links as appropriate. d) If the response content type is `application/geo+json`, validate the response using the GeoJSON schema.

ABSTRACT TEST A.5

IDENTIFIER	`/conf/datastream/canonical-url`
REQUIREMENT	Requirement 5: `/req/datastream/canonical-url`
TEST PURPOSE	Validate that every `DataStream` resource is accessible via its canonical URL.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `DataStream`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. For each item, check that a link with relation type `canonical` is included. 3. Dereference this link and validate that a document is returned with a status code 200. 4. Check that the returned document has the same content as the resource originally included in the collection items (except for the canonical link).

ABSTRACT TEST A.6

IDENTIFIER	`/conf/datastream/resources-endpoint`
REQUIREMENT	Requirement 6: `/req/datastream/resources-endpoint`
TEST PURPOSE	Validate that the server implements a `DataStream` resources endpoint correctly. *This is a parameterized test that requires the endpoint URL as a parameter*
TEST METHOD	1. Issue an HTTP GET request to the endpoint URL. 2. Validate that a document was returned with a status code 200. 3. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_datastream-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.7

IDENTIFIER	`/conf/datastream/canonical-endpoint`
REQUIREMENT	Requirement 7: `/req/datastream/canonical-endpoint`
TEST PURPOSE	Validate that the server exposes the canonical `DataStream` resources endpoint.
TEST METHOD	Validate that the server implements a DataStream resources endpoint at path `{api_root}/datastreams` using test _conf_datastream_resources-endpoint.

ABSTRACT TEST A.8

IDENTIFIER	`/conf/datastream/ref-from-system`
REQUIREMENT	Requirement 8: `/req/datastream/ref-from-system`
TEST PURPOSE	Validate that `DataStream` resources associated to a `System` are available as sub-resources.
TEST METHOD	1. Retrieve all `System` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=systems`. 2. For each `System` resource in the response: a) Validate that the server implements a DataStream resources endpoint at path `{api_root}/systems/{sysId}/datastreams` using test _conf_datastream_resources-endpoint, where `sysId` is the local ID of the `System` resource.

ABSTRACT TEST A.9

IDENTIFIER	`/conf/datastream/ref-from-deployment`
REQUIREMENT	Requirement 9: `/req/datastream/ref-from-deployment`
TEST PURPOSE	Validate that `DataStream` resources associated to a `Deployment` are available as sub-resources.
TEST METHOD	1. Retrieve all `Deployment` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=deployments`. 2. For each `Deployment` resource in the response: a) Validate that the server implements a DataStream resources endpoint at path `{api_root}/deployments/{depId}/datastreams` using test _conf_datastream_resources-endpoint, where `depId` is the local ID of the `Deployment` resource.

ABSTRACT TEST A.10

IDENTIFIER	`/conf/datastream/collections`
REQUIREMENT	Requirement 10: `/req/datastream/collections`
TEST PURPOSE	Validate that `DataStream` collections are tagged with the proper item type.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `DataStream`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_datastream-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.11

IDENTIFIER	`/conf/datastream/schema-op`
REQUIREMENT	Requirement 11: `/req/datastream/schema-op`
TEST PURPOSE	Validate that every `DataStream` resource has a schema sub-resource.

ABSTRACT TEST A.11

TEST METHOD

1. Retrieve all `DataStream` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=datastreams`.
2. For each `DataStream` resource in the response:
   a) Retrieve the list of supported observation formats listed in the `DataStream` resource.
   b) For each supported observation format:
   step:
   Issue an HTTP GET request at path `{api_root}/datastreams/{dsId}/schema?obsFormat={format}`, where `dsId` is the local ID of the `DataStream` resource, and `format` is one of the supported formats.

step:	Validate that a document was returned with a status code 200.

ABSTRACT TEST A.12

IDENTIFIER	`/conf/datastream/obs-canonical-url`
REQUIREMENT	Requirement 12: `/req/datastream/obs-canonical-url`
TEST PURPOSE	Validate that every `Observation` resource is accessible via its canonical URL.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `Observation`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. For each item, check that a link with relation type `canonical` is included. 3. Dereference this link and validate that a document is returned with a status code 200. 4. Check that the returned document has the same content as the resource originally included in the collection items (except for the canonical link).

ABSTRACT TEST A.13

IDENTIFIER	`/conf/datastream/obs-resources-endpoint`
REQUIREMENT	Requirement 13: `/req/datastream/obs-resources-endpoint`
TEST PURPOSE	Validate that the server implements a `Observation` resources endpoint correctly. *This is a parameterized test that requires the endpoint URL as a parameter*
TEST METHOD	1. Issue an HTTP GET request to the endpoint URL. 2. Validate that a document was returned with a status code 200. 3. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header.

ABSTRACT TEST A.13

a) If the response content type is `application/json`, execute test _conf_json_observation-schema.

b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.14

IDENTIFIER	`/conf/datastream/obs-canonical-endpoint`
REQUIREMENT	Requirement 14: `/req/datastream/obs-canonical-endpoint`
TEST PURPOSE	Validate that the server exposes the canonical `Observation` resources endpoint.
TEST METHOD	Validate that the server implements an Observation resources endpoint at path `{api_root}/observations` using test _conf_datastream_obs-resources-endpoint.

ABSTRACT TEST A.15

IDENTIFIER	`/conf/datastream/obs-ref-from-datastream`
REQUIREMENT	Requirement 15: `/req/datastream/obs-ref-from-datastream`
TEST PURPOSE	Validate that `Observation` resources associated to a `DataStream` are available as sub-resources.
TEST METHOD	1. Retrieve all `DataStream` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=datastreams`. 2. For each `DataStream` resource in the response: a) Validate that the server implements an Observation resources endpoint at path `{api_root}/datastreams/{dsId}/observations` using test _conf_datastream_obs-resources-endpoint, where `dsId` is the local ID of the `DataStream` resource.

ABSTRACT TEST A.16

IDENTIFIER	`/conf/datastream/obs-collections`
REQUIREMENT	Requirement 16: `/req/datastream/obs-collections`
TEST PURPOSE	Validate that `Observation` collections are tagged with the proper item type.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `Observation`:

ABSTRACT TEST A.16	
	1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_observation-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

A.3. Conformance Class "Control Streams & Commands"

CONFORMANCE CLASS A.3	
IDENTIFIER	`/conf/controlstream`
REQUIREMENTS CLASS	Requirements class 3: `/req/controlstream`
PREREQUISITE	Conformance class A.1: `/conf/api-common`
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.17: `/conf/controlstream/sf-ref-from-controlstream` Abstract test A.18: `/conf/controlstream/foi-ref-from-controlstream` Abstract test A.19: `/conf/controlstream/canonical-url` Abstract test A.20: `/conf/controlstream/resources-endpoint` Abstract test A.21: `/conf/controlstream/canonical-endpoint` Abstract test A.22: `/conf/controlstream/ref-from-system` Abstract test A.23: `/conf/controlstream/ref-from-deployment` Abstract test A.24: `/conf/controlstream/collections` Abstract test A.25: `/conf/controlstream/schema-op` Abstract test A.26: `/conf/controlstream/cmd-canonical-url` Abstract test A.27: `/conf/controlstream/cmd-resources-endpoint`

CONFORMANCE CLASS A.3	
	Abstract test A.28: /conf/controlstream/cmd-canonical-endpoint Abstract test A.29: /conf/controlstream/cmd-ref-from-controlstream Abstract test A.30: /conf/controlstream/cmd-collections Abstract test A.31: /conf/controlstream/status-resources-endpoint Abstract test A.32: /conf/controlstream/command-status-endpoint Abstract test A.33: /conf/controlstream/result-resources-endpoint Abstract test A.34: /conf/controlstream/command-result-endpoint

ABSTRACT TEST A.17	
IDENTIFIER	/conf/controlstream/sf-ref-from-controlstream
REQUIREMENT	Requirement 17: /req/controlstream/sf-ref-from-controlstream
TEST PURPOSE	Validate that Sampling Features associated to a given control stream are available as sub-resources.
TEST METHOD	1. Retrieve all ControlStream resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter resource-type=controlstreams. 2. For each ControlStream resource in the response: a) Validate that the server implements an Sampling Features resources endpoint at path {api_root}/controlstreams/{dsId}/samplingFeatures using test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/sf/resources-endpoint, where dsId is the local ID of the ControlStream resource.

ABSTRACT TEST A.18	
IDENTIFIER	/conf/controlstream/foi-ref-from-controlstream
REQUIREMENT	Requirement 18: /req/controlstream/foi-ref-from-controlstream
TEST PURPOSE	Validate that Features of Interest associated to a given control stream are available as sub-resources.
TEST METHOD	1. Retrieve all controlstream resources by executing test {part1-spec}/conf/api-common/canonical-resources with parameter resource-type=controlstreams. 2. For each ControlStream resource in the response:

ABSTRACT TEST A.18

a) Issue an HTTP GET request at path `{api_root}/controlstreams/{dsId}/featuresOfInterest`, where `dsId` is the local ID of the `ControlStream` resource.

b) Validate that a document was returned with a status code 200.

c) Iterate through the list of resources in the response, following `next` links as appropriate.

d) If the response content type is `application/geo+json`, validate the response using the GeoJSON schema.

ABSTRACT TEST A.19

IDENTIFIER	`/conf/controlstream/canonical-url`
REQUIREMENT	Requirement 19: `/req/controlstream/canonical-url`
TEST PURPOSE	Validate that every `ControlStream` resource is accessible via its canonical URL.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `ControlStream`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. For each item, check that a link with relation type `canonical` is included. 3. Dereference this link and validate that a document is returned with a status code 200. 4. Check that the returned document has the same content as the resource originally included in the collection items (except for the canonical link).

ABSTRACT TEST A.20

IDENTIFIER	`/conf/controlstream/resources-endpoint`
REQUIREMENT	Requirement 20: `/req/controlstream/resources-endpoint`
TEST PURPOSE	Validate that the server implements a `ControlStream` resources endpoint correctly. *This is a parameterized test that requires the endpoint URL as a parameter*
TEST METHOD	1. Issue an HTTP GET request to the endpoint URL. 2. Validate that a document was returned with a status code 200. 3. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_controlstream-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.21

IDENTIFIER	`/conf/controlstream/canonical-endpoint`
REQUIREMENT	Requirement 21: `/req/controlstream/canonical-endpoint`
TEST PURPOSE	Validate that the server exposes the canonical `ControlStream` resources endpoint.
TEST METHOD	Validate that the server implements a ControlStream resources endpoint at path `{api_root}/controlstreams` using test _conf_controlstream_resources-endpoint.

ABSTRACT TEST A.22

IDENTIFIER	`/conf/controlstream/ref-from-system`
REQUIREMENT	Requirement 22: `/req/controlstream/ref-from-system`
TEST PURPOSE	Validate that `ControlStream` resources associated to a `System` are available as sub-resources.
TEST METHOD	1. Retrieve all `System` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=systems`. 2. For each `System` resource in the response: a) Validate that the server implements a ControlStream resources endpoint at path `{api_root}/systems/{sysId}/controlstreams` using test _conf_controlstream_resources-endpoint, where `sysId` is the local ID of the `System` resource.

ABSTRACT TEST A.23

IDENTIFIER	`/conf/controlstream/ref-from-deployment`
REQUIREMENT	Requirement 23: `/req/controlstream/ref-from-deployment`
TEST PURPOSE	Validate that `ControlStream` resources associated to a `Deployment` are available as sub-resources.
TEST METHOD	1. Retrieve all `Deployment` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=deployments`. 2. For each `Deployment` resource in the response: a) Validate that the server implements a ControlStream resources endpoint at path `{api_root}/deployments/{depId}/controlstreams` using test _conf_controlstream_resources-endpoint, where `depId` is the local ID of the `Deployment` resource.

ABSTRACT TEST A.24

IDENTIFIER	`/conf/controlstream/collections`
REQUIREMENT	Requirement 24: `/req/controlstream/collections`
TEST PURPOSE	Validate that `ControlStream` collections are tagged with the proper item type.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `ControlStream`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_ controlstream-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.25

IDENTIFIER	`/conf/controlstream/schema-op`
REQUIREMENT	Requirement 25: `/req/controlstream/schema-op`
TEST PURPOSE	Validate that every `ControlStream` resource has a schema sub-resource.
TEST METHOD	1. Retrieve all `ControlStream` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=controlstreams`. 2. For each `ControlStream` resource in the response: a) Retrieve the list of supported command formats listed in the `ControlStream` resource. b) For each supported command format: step: Issue an HTTP GET request at path `{api_root}/controlstreams/{dsId}/schema?cmdFormat={format}`, where `dsId` is the local ID of the `ControlStream` resource, and `format` is one of the supported formats. step: Validate that a document was returned with a status code 200.

ABSTRACT TEST A.26

IDENTIFIER	`/conf/controlstream/cmd-canonical-url`
REQUIREMENT	Requirement 26: `/req/controlstream/cmd-canonical-url`

ABSTRACT TEST A.26

TEST PURPOSE	Validate that every `Command` resource is accessible via its canonical URL.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `Command`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. For each item, check that a link with relation type `canonical` is included. 3. Dereference this link and validate that a document is returned with a status code 200. 4. Check that the returned document has the same content as the resource originally included in the collection items (except for the canonical link).

ABSTRACT TEST A.27

IDENTIFIER	`/conf/controlstream/cmd-resources-endpoint`
REQUIREMENT	Requirement 27: `/req/controlstream/cmd-resources-endpoint`
TEST PURPOSE	Validate that the server implements a `Command` resources endpoint correctly. *This is a parameterized test that requires the endpoint URL as a parameter*
TEST METHOD	1. Issue an HTTP GET request to the endpoint URL. 2. Validate that a document was returned with a status code 200. 3. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_command-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.28

IDENTIFIER	`/conf/controlstream/cmd-canonical-endpoint`
REQUIREMENT	Requirement 28: `/req/controlstream/cmd-canonical-endpoint`
TEST PURPOSE	Validate that the server exposes the canonical `Command` resources endpoint.
TEST METHOD	Validate that the server implements a Command resources endpoint at path `{api_root}/commands` using test _conf_controlstream_cmd-resources-endpoint.

ABSTRACT TEST A.29

IDENTIFIER	`/conf/controlstream/cmd-ref-from-controlstream`
REQUIREMENT	Requirement 29: `/req/controlstream/cmd-ref-from-controlstream`
TEST PURPOSE	Validate that `Command` resources associated to a `ControlStream` are available as sub-resources.
TEST METHOD	1. Retrieve all `ControlStream` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=controlstreams`. 2. For each `ControlStream` resource in the response: a) Validate that the server implements a Command resources endpoint at path `{api_root}/controlstreams/{dsId}/commands` using test _conf_controlstream_cmd-resources-endpoint, where `dsId` is the local ID of the `ControlStream` resource.

ABSTRACT TEST A.30

IDENTIFIER	`/conf/controlstream/cmd-collections`
REQUIREMENT	Requirement 30: `/req/controlstream/cmd-collections`
TEST PURPOSE	Validate that `Command` collections are tagged with the proper item type.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `Command`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_command-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.31

IDENTIFIER	`/conf/controlstream/status-resources-endpoint`
REQUIREMENT	Requirement 31: `/req/controlstream/status-resources-endpoint`
TEST PURPOSE	Validate that the server implements a `CommandStatus` resources endpoint correctly. *This is a parameterized test that requires the endpoint URL as a parameter*
TEST METHOD	1. Issue an HTTP GET request to the endpoint URL. 2. Validate that a document was returned with a status code 200.

ABSTRACT TEST A.31

3. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header.
   a) If the response content type is `application/json`, execute test _conf_json_commandstatus-schema.
   b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.32

IDENTIFIER	`/conf/controlstream/command-status-endpoint`
REQUIREMENT	Requirement 32: `/req/controlstream/command-status-endpoint`
TEST PURPOSE	Validate that every `Command` resource has a status endpoint
TEST METHOD	1. Retrieve all `Command` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=commands`. 2. For each `Command` resource in the response: a) Validate that the server implements a Command Status resources endpoint at path `{api_root}/commands/{cmdId}/status` using test _conf_controlstream_status-resources-endpoint, where `cmdId` is the local ID of the `Command` resource.

ABSTRACT TEST A.33

IDENTIFIER	`/conf/controlstream/result-resources-endpoint`
REQUIREMENT	Requirement 33: `/req/controlstream/result-resources-endpoint`
TEST PURPOSE	Validate that the server implements a `CommandResult` resources endpoint correctly. *This is a parameterized test that requires the endpoint URL as a parameter*
TEST METHOD	1. Issue an HTTP GET request to the endpoint URL. 2. Validate that a document was returned with a status code 200. 3. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_commandresult-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.34	
IDENTIFIER	`/conf/controlstream/command-result-endpoint`
REQUIREMENT	Requirement 34: `/req/controlstream/command-result-endpoint`
TEST PURPOSE	Validate that every `Command` resource has a result endpoint
TEST METHOD	1. Retrieve all `Command` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=commands`. 2. For each `Command` resource in the response: a) Validate that the server implements a Command Result resources endpoint at path `{api_root}/commands/{cmdId}/result` using test _conf_controlstream_result-resources-endpoint, where `cmdId` is the local ID of the `Command` resource.

A.4. Conformance Class “Command Feasibility”

CONFORMANCE CLASS A.4	
IDENTIFIER	`/conf/feasibility`
REQUIREMENTS CLASS	Requirements class 4: `/req/feasibility`
PREREQUISITE	Conformance class A.3: `/conf/controlstream`
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.35: `/conf/feasibility/canonical-url` Abstract test A.36: `/conf/feasibility/ref-from-controlstream` Abstract test A.37: `/conf/feasibility/status-endpoint` Abstract test A.38: `/conf/feasibility/result-endpoint` Abstract test A.39: `/conf/feasibility/collections`

ABSTRACT TEST A.35

IDENTIFIER	`/conf/feasibility/canonical-url`
REQUIREMENT	Requirement 35: `/req/feasibility/canonical-url`
TEST PURPOSE	Validate that every `Command` resource is accessible via its canonical URL.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `Command`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. For each item, check that a link with relation type `canonical` is included. 3. Dereference this link and validate that a document is returned with a status code 200. 4. Check that the returned document has the same content as the resource originally included in the collection items (except for the canonical link).

ABSTRACT TEST A.36

IDENTIFIER	`/conf/feasibility/ref-from-controlstream`
REQUIREMENT	Requirement 36: `/req/feasibility/ref-from-controlstream`
TEST PURPOSE	Validate that `Command` resources associated to a `ControlStream` are available as sub-resources.
TEST METHOD	1. Retrieve all `ControlStream` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=controlstreams`. 2. For each `ControlStream` resource in the response: a) Validate that the server implements a Command resources endpoint at path `{api_root}/controlstreams/{dsId}/commands` using test _conf_controlstream_cmd-resources-endpoint, where `dsId` is the local ID of the `ControlStream` resource.

ABSTRACT TEST A.37

IDENTIFIER	`/conf/feasibility/status-endpoint`
REQUIREMENT	Requirement 37: `/req/feasibility/status-endpoint`
TEST PURPOSE	Validate that every `Feasibility` resource has a status endpoint
TEST METHOD	1. Retrieve all `Feasibility` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=feasibility`. 2. For each `Feasibility` resource in the response:

ABSTRACT TEST A.37	
	a) Validate that the server implements a Command Status resources endpoint at path `{api_root}/feasibility/{cmdId}/status` using test _conf_controlstream_status-resources-endpoint, where `cmdId` is the local ID of the `Feasibility` resource.

ABSTRACT TEST A.38	
IDENTIFIER	`/conf/feasibility/result-endpoint`
REQUIREMENT	Requirement 38: `/req/feasibility/result-endpoint`
TEST PURPOSE	Validate that every `Feasibility` resource has a result endpoint
TEST METHOD	1. Retrieve all `Feasibility` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=feasibility`. 2. For each `Feasibility` resource in the response: a) Validate that the server implements a Command Result resources endpoint at path `{api_root}/feasibility/{cmdId}/result` using test _conf_controlstream_result-resources-endpoint, where `cmdId` is the local ID of the `Feasibility` resource.

ABSTRACT TEST A.39	
IDENTIFIER	`/conf/feasibility/collections`
REQUIREMENT	Requirement 39: `/req/feasibility/collections`
TEST PURPOSE	Validate that `Feasibility` collections are tagged with the proper item type.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `Feasibility`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_command-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

A.5. Conformance Class “System Events”

CONFORMANCE CLASS A.5

IDENTIFIER	`/conf/system-event`
REQUIREMENTS CLASS	Requirements class 5: `/req/system-event`
PREREQUISITES	Conformance class A.1: `/conf/api-common` http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/system
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.40: `/conf/system-event/canonical-url` Abstract test A.41: `/conf/system-event/resources-endpoint` Abstract test A.42: `/conf/system-event/canonical-endpoint` Abstract test A.43: `/conf/system-event/ref-from-system` Abstract test A.44: `/conf/system-event/collections`

ABSTRACT TEST A.40

IDENTIFIER	`/conf/system-event/canonical-url`
REQUIREMENT	Requirement 40: `/req/system-event/canonical-url`
TEST PURPOSE	Validate that every `ControlStream` resource is accessible via its canonical URL.
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `ControlStream`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. For each item, check that a link with relation type `canonical` is included. 3. Dereference this link and validate that a document is returned with a status code 200. 4. Check that the returned document has the same content as the resource originally included in the collection items (except for the canonical link).

ABSTRACT TEST A.41

IDENTIFIER	`/conf/system-event/resources-endpoint`
REQUIREMENT	Requirement 41: `/req/system-event/resources-endpoint`
TEST PURPOSE	Validate that the server implements a `SystemEvent` resources endpoint correctly. *This is a parameterized test that requires the endpoint URL as a parameter*
TEST METHOD	1. Issue an HTTP GET request to the endpoint URL. 2. Validate that a document was returned with a status code 200.

ABSTRACT TEST A.41

3. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header.
   a) If the response content type is `application/json`, execute test _conf_json_systemevent-schema.
   b) For other response content types not supported by the testing engine, issue a warning and skip this test.

ABSTRACT TEST A.42

IDENTIFIER	`/conf/system-event/canonical-endpoint`
REQUIREMENT	Requirement 42: `/req/system-event/canonical-endpoint`
TEST PURPOSE	Validate that the server exposes the canonical `SystemEvent` resources endpoint.
TEST METHOD	Validate that the server implements a System Event resources endpoint at path `{api_root}/systemEvents` using test _conf_controlstream_resources-endpoint.

ABSTRACT TEST A.43

IDENTIFIER	`/conf/system-event/ref-from-system`
REQUIREMENT	Requirement 43: `/req/system-event/ref-from-system`
TEST PURPOSE	Validate that `SystemEvent` resources associated to a `System` are available as sub-resources.
TEST METHOD	1. Retrieve all `System` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=systems`. 2. For each `System` resource in the response: a) Validate that the server implements a System Event resources endpoint at path `{api_root}/systems/{sysId}/systemEvents` using test _conf_system-event_resources-endpoint, where `sysId` is the local ID of the `System` resource.

ABSTRACT TEST A.44

IDENTIFIER	`/conf/system-event/collections`
REQUIREMENT	Requirement 44: `/req/system-event/collections`
TEST PURPOSE	Validate that `SystemEvent` collections are tagged with the proper item type.

ABSTRACT TEST A.44	
TEST METHOD	For every collection advertised by the server with the `itemType` property set to `SystemEvent`: 1. Retrieve the collection items as described in test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/collection-items. 2. Validate that the contents of the returned document conform to the media type reported by the response `Content-Type` header. a) If the response content type is `application/json`, execute test _conf_json_systemevent-schema. b) For other response content types not supported by the testing engine, issue a warning and skip this test.

A.6. Conformance Class "Advanced Filtering"

CONFORMANCE CLASS A.6	
IDENTIFIER	`/conf/advanced-filtering`
REQUIREMENTS CLASS	Requirements class 6: `/req/advanced-filtering`
PREREQUISITE	Conformance class A.1: `/conf/api-common`
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.45: `/conf/advanced-filtering/datastream-by-phenomenontime` Abstract test A.46: `/conf/advanced-filtering/datastream-by-resulttime` Abstract test A.47: `/conf/advanced-filtering/datastream-by-obsprop` Abstract test A.48: `/conf/advanced-filtering/datastream-by-foi` Abstract test A.49: `/conf/advanced-filtering/obs-by-phenomenontime` Abstract test A.50: `/conf/advanced-filtering/obs-by-resulttime` Abstract test A.51: `/conf/advanced-filtering/obs-by-foi` Abstract test A.52: `/conf/advanced-filtering/controlstream-by-issuetime` Abstract test A.53: `/conf/advanced-filtering/controlstream-by-exectime` Abstract test A.54: `/conf/advanced-filtering/controlstream-by-controlprop`

CONFORMANCE CLASS A.6

	Abstract test A.55: /conf/advanced-filtering/controlstream-by-foi Abstract test A.56: /conf/advanced-filtering/cmd-by-issuetime Abstract test A.57: /conf/advanced-filtering/cmd-by-exectime Abstract test A.58: /conf/advanced-filtering/cmd-by-status Abstract test A.59: /conf/advanced-filtering/cmd-by-sender Abstract test A.60: /conf/advanced-filtering/cmd-by-foi Abstract test A.61: /conf/advanced-filtering/status-by-statuscode Abstract test A.62: /conf/advanced-filtering/event-by-type

ABSTRACT TEST A.45

IDENTIFIER	/conf/advanced-filtering/datastream-by-phenomenontime
REQUIREMENT	Requirement 45: /req/advanced-filtering/datastream-by-phenomenontime
TEST PURPOSE	Validate that the phenomenonTime query parameter is processed correctly.
TEST METHOD	1. Issue an HTTP GET request at URL {api_root}/datastreams?phenomenonTime={datetime} where {datetime} is a time instant or period (see the requirement for the exact syntax of the parameter). 2. Validate the response using the steps described in test _conf_datastream_resources-endpoint. 3. For each DataStream resource in the response: a) Retrieve its phenomenonTime property. b) Verify that the value of the property intersects the time specified in the request.

ABSTRACT TEST A.46

IDENTIFIER	/conf/advanced-filtering/datastream-by-resulttime
REQUIREMENT	Requirement 46: /req/advanced-filtering/datastream-by-resulttime
TEST PURPOSE	Validate that the resultTime query parameter is processed correctly.

ABSTRACT TEST A.46

TEST METHOD

1. Issue an HTTP GET request at URL `{api_root}/datastreams?resultTime={datetime}` where `{datetime}` is a time instant or period (see the requirement for the exact syntax of the parameter).
2. Validate the response using the steps described in test _conf_datastream_resources-endpoint.
3. For each `DataStream` resource in the response:
   a) Retrieve its `resultTime` property.
   b) Verify that the value of the property intersects the time specified in the request.

ABSTRACT TEST A.47

IDENTIFIER `/conf/advanced-filtering/datastream-by-obsprop`

REQUIREMENT Requirement 47: `/req/advanced-filtering/datastream-by-obsprop`

TEST PURPOSE Validate that the `observedProperty` query parameter is processed correctly.

TEST METHOD

1. Issue an HTTP GET request at URL `{api_root}/datastreams?observedProperty={idList}` where `{idList}` is a list of one or more local IDs of `Property` resources. See test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test _conf_datastream_resources-endpoint.
3. For each `DataStream` resource in the response:
   a) Retrieve all observed properties listed in the `DataStream` resource.
   b) Verify that at least one of the collected properties has one of the identifiers included in `{idList}`.
4. Repeat the previous steps with the `observedProperty` parameter set to a list of one or more URIs identifying observable properties.

ABSTRACT TEST A.48

IDENTIFIER `/conf/advanced-filtering/datastream-by-foi`

REQUIREMENT Requirement 48: `/req/advanced-filtering/datastream-by-foi`

TEST PURPOSE Validate that the `foi` query parameter is processed correctly.

TEST METHOD

1. Issue an HTTP GET request at URL `{api_root}/datastreams?foi={idList}` where `{idList}` is a list of one or more local IDs of `Sampling Feature` or `Feature` resources. See test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/advanced-filtering/id-list-schema

ABSTRACT TEST A.48

2. Validate the response using the steps described in test _conf_datastream_resources-endpoint.

3. For each `DataStream` resource in the response:
   a) Retrieve the datastreams's sampling features by issuing an HTTP GET request at `{api_root}/datastreams/{dsId}/samplingFeatures`.
   b) For each `Sampling Feature` resource in the returned collection:
      1. Follow the `sampleOf` links to retrieve the target features, recursively. If a link does not resolve or the link media type is not supported by the testing engine, use the link target as the identifier of the feature.
   c) Verify that at least one of the collected features has one of the identifiers included in `{idList}`.

4. Repeat the previous steps with the `foi` parameter set to a list of one or more UIDs of `Feature` resources.

ABSTRACT TEST A.49

IDENTIFIER `/conf/advanced-filtering/obs-by-phenomenontime`

REQUIREMENT Requirement 49: `/req/advanced-filtering/obs-by-phenomenontime`

TEST PURPOSE Validate that the `phenomenonTime` query parameter is processed correctly.

TEST METHOD

1. Issue an HTTP GET request at URL `{api_root}/observations?phenomenonTime={datetime}` where `{datetime}` is a time instant or period (see the requirement for the exact syntax of the parameter).

2. Validate the response using the steps described in test _conf_datastream_obs-resources-endpoint.

3. For each `Observation` resource in the response:
   a) Retrieve its `phenomenonTime` property.
   b) Verify that the value of the property intersects the time specified in the request.

4. Repeat the steps above for every observation resources endpoint nested under a `DataStream` resource, that is at endpoints `{api_root}/datastreams/{dsId}/observations` where `dsId` is the local ID of a `DataStream` resource.

ABSTRACT TEST A.50

IDENTIFIER `/conf/advanced-filtering/obs-by-resulttime`

REQUIREMENT Requirement 50: `/req/advanced-filtering/obs-by-resulttime`

TEST PURPOSE Validate that the `resultTime` query parameter is processed correctly.

ABSTRACT TEST A.50

TEST METHOD

1. Issue an HTTP GET request at URL `{api_root}/observations?resultTime={datetime}` where `{datetime}` is a time instant or period (see the requirement for the exact syntax of the parameter).
2. Validate the response using the steps described in test _conf_datastream_obs-resources-endpoint.
3. For each `Observation` resource in the response:
   a) Retrieve its `resultTime` property.
   b) Verify that the value of the property intersects the time specified in the request.
4. Repeat the steps above for every observation resources endpoint nested under a `DataStream` resource, that is at endpoints `{api_root}/datastreams/{dsId}/observations` where `dsId` is the local ID of a `DataStream` resource.

ABSTRACT TEST A.51

IDENTIFIER `/conf/advanced-filtering/obs-by-foi`

REQUIREMENT Requirement 51: `/req/advanced-filtering/obs-by-foi`

TEST PURPOSE Validate that the `foi` query parameter is processed correctly.

TEST METHOD

1. Issue an HTTP GET request at URL `{api_root}/observations?foi={idList}` where `{idList}` is a list of one or more local IDs of `Sampling Feature` or `Feature` resources. See test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/advanced-filtering/id-list-schema
2. Validate the response using the steps described in test _conf_datastream_obs-resources-endpoint.
3. For each `Observation` resource in the response:
   a) Retrieve its `samplingFeature` property.
   b) Follow the `sampleOf` links to retrieve the sampled features, recursively. If a link does not resolve or the link media type is not supported by the testing engine, use the link target as the identifier of the feature.
   c) Verify that at least one of the collected features has one of the identifiers included in `{idList}`.
4. Repeat the previous steps with the `foi` parameter set to a list of one or more UIDs of `Feature` resources.
5. Repeat the steps above for every observation resources endpoint nested under a `DataStream` resource, that is at endpoints `{api_root}/datastreams/{dsId}/observations` where `dsId` is the local ID of a `DataStream` resource.

ABSTRACT TEST A.52

IDENTIFIER `/conf/advanced-filtering/controlstream-by-issuetime`

ABSTRACT TEST A.52

REQUIREMENT	Requirement 52: `/req/advanced-filtering/controlstream-by-issuetime`
TEST PURPOSE	Validate that the `issueTime` query parameter is processed correctly.
TEST METHOD	1. Issue an HTTP GET request at URL `{api_root}/controlstreams?issueTime={datetime}` where `{datetime}` is a time instant or period (see the requirement for the exact syntax of the parameter). 2. Validate the response using the steps described in test _conf_controlstream_resources-endpoint. 3. For each `ControlStream` resource in the response: a) Retrieve its `issueTime` property. b) Verify that the value of the property intersects the time specified in the request.

ABSTRACT TEST A.53

IDENTIFIER	`/conf/advanced-filtering/controlstream-by-exectime`
REQUIREMENT	Requirement 53: `/req/advanced-filtering/controlstream-by-exectime`
TEST PURPOSE	Validate that the `executionTime` query parameter is processed correctly.
TEST METHOD	1. Issue an HTTP GET request at URL `{api_root}/controlstreams?executionTime={datetime}` where `{datetime}` is a time instant or period (see the requirement for the exact syntax of the parameter). 2. Validate the response using the steps described in test _conf_controlstream_resources-endpoint. 3. For each `ControlStream` resource in the response: a) Retrieve its `executionTime` property. b) Verify that the value of the property intersects the time specified in the request.

ABSTRACT TEST A.54

IDENTIFIER	`/conf/advanced-filtering/controlstream-by-controlprop`
REQUIREMENT	Requirement 54: `/req/advanced-filtering/controlstream-by-controlprop`
TEST PURPOSE	Validate that the `controlledProperty` query parameter is processed correctly.
TEST METHOD	1. Issue an HTTP GET request at URL `{api_root}/controlstreams?controlledProperty={idList}` where `{idList}` is a list of one or more local IDs of `Property` resources.

ABSTRACT TEST A.54

See test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/advanced-filtering/id-list-schema

2. Validate the response using the steps described in test _conf_controlstream_resources-endpoint.

3. For each `ControlStream` resource in the response:
   a) Retrieve all controlled properties listed in the `ControlStream` resource.

   b) Verify that at least one of the collected properties has one of the identifiers included in `{idList}`.

4. Repeat the previous steps with the `controlledProperty` parameter set to a list of one or more URIs identifying controllable properties.

ABSTRACT TEST A.55

IDENTIFIER `/conf/advanced-filtering/controlstream-by-foi`

REQUIREMENT Requirement 55: `/req/advanced-filtering/controlstream-by-foi`

TEST PURPOSE Validate that the `foi` query parameter is processed correctly.

TEST METHOD

1. Issue an HTTP GET request at URL `{api_root}/controlstreams?foi={idList}` where `{idList}` is a list of one or more local IDs of `Sampling Feature` or `Feature` resources. See test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/advanced-filtering/id-list-schema

2. Validate the response using the steps described in test _conf_controlstream_resources-endpoint.

3. For each `ControlStream` resource in the response:
   a) Retrieve the control streams's sampling features by issuing an HTTP GET request at `{api_root}/controlstreams/{dsId}/samplingFeatures`.

   b) For each `Sampling Feature` resource in the returned collection:
      1. Follow the `sampleOf` links to retrieve the target features, recursively. If a link does not resolve or the link media type is not supported by the testing engine, use the link target as the identifier of the feature.

   c) Verify that at least one of the collected features has one of the identifiers included in `{idList}`.

4. Repeat the previous steps with the `foi` parameter set to a list of one or more UIDs of `Feature` resources.

ABSTRACT TEST A.56

IDENTIFIER `/conf/advanced-filtering/cmd-by-issuetime`

REQUIREMENT Requirement 56: `/req/advanced-filtering/cmd-by-issuetime`

ABSTRACT TEST A.56

TEST PURPOSE Validate that the `issueTime` query parameter is processed correctly.

TEST METHOD

1. Issue an HTTP GET request at URL `{api_root}/commands?issueTime={datetime}` where `{datetime}` is a time instant or period (see the requirement for the exact syntax of the parameter).
2. Validate the response using the steps described in test _conf_controlstream_cmd-resources-endpoint.
3. For each `Command` resource in the response:
   a) Retrieve its `issueTime` property.
   b) Verify that the value of the property intersects the time specified in the request.
4. Repeat the steps above for every command resources endpoint nested under a `ControlStream` resource, that is at endpoints `{api_root}/controlstreams/{dsId}/commands` where `dsId` is the local ID of a `ControlStream` resource.

ABSTRACT TEST A.57

IDENTIFIER `/conf/advanced-filtering/cmd-by-exectime`

REQUIREMENT Requirement 57: `/req/advanced-filtering/cmd-by-exectime`

TEST PURPOSE Validate that the `executionTime` query parameter is processed correctly.

TEST METHOD

1. Issue an HTTP GET request at URL `{api_root}/commands?executionTime={datetime}` where `{datetime}` is a time instant or period (see the requirement for the exact syntax of the parameter).
2. Validate the response using the steps described in test _conf_controlstream_cmd-resources-endpoint.
3. For each `Command` resource in the response:
   a) Retrieve its `executionTime` property.
   b) Verify that the value of the property intersects the time specified in the request.
4. Repeat the steps above for every command resources endpoint nested under a `ControlStream` resource, that is at endpoints `{api_root}/controlstreams/{dsId}/commands` where `dsId` is the local ID of a `ControlStream` resource.

ABSTRACT TEST A.58

IDENTIFIER `/conf/advanced-filtering/cmd-by-status`

REQUIREMENT Requirement 58: `/req/advanced-filtering/cmd-by-status`

TEST PURPOSE Validate that the `statusCode` query parameter is processed correctly.

ABSTRACT TEST A.58

TEST METHOD	1. Issue an HTTP GET request at URL `{api_root}/commands?statusCode={idList}` where `{idList}` is a list of one or more status codes (see requirement for the possible values). 2. Validate the response using the steps described in test _conf_controlstream_cmd-resources-endpoint. 3. For each `Command` resource in the response: a) Retrieve its `currentStatus` property b) Verify that the value of the property is equal to one of the status codes listed in the request. 4. Repeat the steps above for every command resources endpoint nested under a `ControlStream` resource, that is at endpoints `{api_root}/controlstreams/{dsId}/commands` where `dsId` is the local ID of a `ControlStream` resource.

ABSTRACT TEST A.59

IDENTIFIER	`/conf/advanced-filtering/cmd-by-sender`
REQUIREMENT	Requirement 59: `/req/advanced-filtering/cmd-by-sender`
TEST PURPOSE	Validate that the `sender` query parameter is processed correctly.
TEST METHOD	1. Issue an HTTP GET request at URL `{api_root}/commands?sender={idList}` where `{idList}` is a list of one or more sender IDs. 2. Validate the response using the steps described in test _conf_controlstream_cmd-resources-endpoint. 3. For each `Command` resource in the response: a) Retrieve its `sender` property b) Verify that the value of the property is equal to one of the IDs listed in the request. 4. Repeat the steps above for every command resources endpoint nested under a `ControlStream` resource, that is at endpoints `{api_root}/controlstreams/{dsId}/commands` where `dsId` is the local ID of a `ControlStream` resource.

ABSTRACT TEST A.60

IDENTIFIER	`/conf/advanced-filtering/cmd-by-foi`
REQUIREMENT	Requirement 60: `/req/advanced-filtering/cmd-by-foi`
TEST PURPOSE	Validate that the `foi` query parameter is processed correctly.
TEST METHOD	1. Issue an HTTP GET request at URL `{api_root}/commands?foi={idList}` where `{idList}` is a list of one or more local IDs of `Sampling Feature` or `Feature` resources.

ABSTRACT TEST A.60

	See test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/advanced-filtering/id-list-schema 2. Validate the response using the steps described in test _conf_controlstream_cmd-resources-endpoint. 3. For each `Command` resource in the response: a) Retrieve its `samplingFeature` property. b) Follow the `sampleOf` links to retrieve the sampled features, recursively. If a link does not resolve or the link media type is not supported by the testing engine, use the link target as the identifier of the feature. c) Verify that at least one of the collected features has one of the identifiers included in `{idList}`. 4. Repeat the previous steps with the `foi` parameter set to a list of one or more UIDs of `Feature` resources. 5. Repeat the steps above for every commands resources endpoint nested under a `ControlStream` resource, that is at endpoints `{api_root}/controlstreams/{dsId}/commands` where `dsId` is the local ID of a `ControlStream` resource.

ABSTRACT TEST A.61

IDENTIFIER	`/conf/advanced-filtering/status-by-statuscode`
REQUIREMENT	Requirement 61: `/req/advanced-filtering/status-by-statuscode`
TEST PURPOSE	Validate that the `statusCode` query parameter is processed correctly.
TEST METHOD	Retrieve all `Command` resources by executing test http://www.opengis.net/spec/ogcapi-connectedsystems-1/1.0/conf/api-common/canonical-resources with parameter `resource-type=commands`, then for or every `Command` resource: 1. Issue an HTTP GET request at URL `{api_root}/commands/{cmdId}/status?statusCode={idList}` where `{idList}` is a list of one or more status codes (see requirement for the possible values). 2. Validate the response using the steps described in test _conf_controlstream_status-resources-endpoint. 3. For each `Command` resource in the response: a) Retrieve its `currentStatus` property b) Verify that the value of the property is equal to one of the status codes listed in the request.

ABSTRACT TEST A.62

IDENTIFIER	`/conf/advanced-filtering/event-by-type`

ABSTRACT TEST A.62	
REQUIREMENT	Requirement 62: `/req/advanced-filtering/event-by-type`
TEST PURPOSE	Validate that the `eventType` query parameter is processed correctly.
TEST METHOD	1. Issue an HTTP GET request at URL `{api_root}/systemevents?eventType={type}` where `{type}` is a list of one or more event types. 2. Validate the response using the steps described in test _conf_system-event_resources-endpoint. 3. For each `SystemEvent` resource in the response: a) Retrieve its `type` property b) Verify that the value of the property is equal to one of the types listed in the request. 4. Repeat the steps above for every system event resources endpoint nested under a `System` resource, that is at endpoints `{api_root}/systems/{sysId}/events` where `sysId` is the local ID of a `System` resource.

A.7. Conformance Class “Create/Replace/Delete”

CONFORMANCE CLASS A.7	
IDENTIFIER	`/conf/create-replace-delete`
REQUIREMENTS CLASS	Requirements class 7: `/req/create-replace-delete`
PREREQUISITE	http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.63: `/conf/create-replace-delete/datastream` Abstract test A.64: `/conf/create-replace-delete/datastream-update-schema` Abstract test A.65: `/conf/create-replace-delete/datastream-delete-cascade` Abstract test A.66: `/conf/create-replace-delete/observation` Abstract test A.67: `/conf/create-replace-delete/observation-schema` Abstract test A.68: `/conf/create-replace-delete/controlstream` Abstract test A.69: `/conf/create-replace-delete/controlstream-update-schema`

CONFORMANCE CLASS A.7

	Abstract test A.70: /conf/create-replace-delete/controlstream-delete-cascade Abstract test A.71: /conf/create-replace-delete/command Abstract test A.72: /conf/create-replace-delete/command-schema Abstract test A.73: /conf/create-replace-delete/command-status Abstract test A.74: /conf/create-replace-delete/command-result Abstract test A.75: /conf/create-replace-delete/feasibility Abstract test A.76: /conf/create-replace-delete/feasibility-status Abstract test A.77: /conf/create-replace-delete/feasibility-result Abstract test A.78: /conf/create-replace-delete/system-event

ABSTRACT TEST A.63

IDENTIFIER	/conf/create-replace-delete/datastream
REQUIREMENT	Requirement 63: /req/create-replace-delete/datastream
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for DataStream resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint {api_root}/systems/{sysId}/datastreams (for CREATE) b) At resource endpoint {api_root}/systems/{sysId}/datastreams/{id} (for REPLACE and DELETE) c) At resource endpoint {api_root}/datastreams/{id} (for REPLACE and DELETE)

ABSTRACT TEST A.64

IDENTIFIER	/conf/create-replace-delete/datastream-update-schema
REQUIREMENT	Requirement 64: /req/create-replace-delete/datastream-update-schema
TEST PURPOSE	Validate that the server rejects DataStream REPLACE requests with incompatible schemas.
TEST METHOD	1. Given a DataStream resource with ID dsId that has associated observations:

ABSTRACT TEST A.64

a) Issue an HTTP PUT request at URL `{api_root}/datastreams/{dsId}` with a different observation schema.

b) Verify that the server responds with an error code 409.

ABSTRACT TEST A.65

IDENTIFIER	`/conf/create-replace-delete/datastream-delete-cascade`
REQUIREMENT	Requirement 65: `/req/create-replace-delete/datastream-delete-cascade`
TEST PURPOSE	Validate that the server implements the `cascade` query parameter correctly.
TEST METHOD	1. Given a `DataStream` resource with ID `dsId` that has observations: a) Issue an HTTP DELETE request at URL `{api_root}/datastreams/{dsId}?cascade=false`. b) Verify that the server responds with an error code 409. c) Issue an HTTP DELETE request at URL `{api_root}/datastreams/{dsId}?cascade=true`. d) Verify that the datastream and all its observations have been deleted.

ABSTRACT TEST A.66

IDENTIFIER	`/conf/create-replace-delete/observation`
REQUIREMENT	Requirement 66: `/req/create-replace-delete/observation`
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for `Observation` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint `{api_root}/datastreams/{dsId}/observations` (for CREATE) b) At resource endpoint `{api_root}/datastreams/{dsId}/observations/{id}` (for REPLACE and DELETE) c) At resource endpoint `{api_root}/observations/{id}` (for REPLACE and DELETE)

ABSTRACT TEST A.67

IDENTIFIER	`/conf/create-replace-delete/observation-schema`

ABSTRACT TEST A.67

REQUIREMENT	Requirement 67: `/req/create-replace-delete/observation-schema`
TEST PURPOSE	Validate that the server rejects observations with incompatible schemas.
TEST METHOD	1. Given a `DataStream` resource with ID `dsId`: a) Issue an HTTP CREATE request at URL `{api_root}/datastreams/{dsId}/observations` with an observation whose result structure is incompatible with the observation schema registered with the datastream. b) Verify that the server responds with an error code 400.

ABSTRACT TEST A.68

IDENTIFIER	`/conf/create-replace-delete/controlstream`
REQUIREMENT	Requirement 68: `/req/create-replace-delete/controlstream`
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for `ControlStream` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint `{api_root}/systems/{sysId}/controlstreams` (for CREATE) b) At resource endpoint `{api_root}/systems/{sysId}/controlstreams/{id}` (for REPLACE and DELETE) c) At resource endpoint `{api_root}/controlstreams/{id}` (for REPLACE and DELETE)

ABSTRACT TEST A.69

IDENTIFIER	`/conf/create-replace-delete/controlstream-update-schema`
REQUIREMENT	Requirement 69: `/req/create-replace-delete/controlstream-update-schema`
TEST PURPOSE	Validate that the server rejects `ControlStream` REPLACE requests with incompatible schemas.
TEST METHOD	1. Given a `ControlStream` resource with ID `dsId` that has associated commands: a) Issue an HTTP PUT request at URL `{api_root}/controlstreams/{dsId}` with a different observation schema. b) Verify that the server responds with an error code 409.

ABSTRACT TEST A.70

IDENTIFIER	`/conf/create-replace-delete/controlstream-delete-cascade`
REQUIREMENT	Requirement 70: `/req/create-replace-delete/controlstream-delete-cascade`
TEST PURPOSE	Validate that the server implements the `cascade` query parameter correctly.
TEST METHOD	1. Given a `ControlStream` resource with ID `dsId` that has commands: a) Issue an HTTP DELETE request at URL `{api_root}/controlstreams/{dsId}?cascade=false`. b) Verify that the server responds with an error code 409. c) Issue an HTTP DELETE request at URL `{api_root}/controlstreams/{dsId}?cascade=true`. d) Verify that the control stream and all its commands have been deleted.

ABSTRACT TEST A.71

IDENTIFIER	`/conf/create-replace-delete/command`
REQUIREMENT	Requirement 71: `/req/create-replace-delete/command`
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for `Command` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint `{api_root}/controlstreams/{dsId}/commands` (for CREATE) b) At resource endpoint `{api_root}/controlstreams/{dsId}/commands/{id}` (for REPLACE and DELETE) c) At resource endpoint `{api_root}/commands/{id}` (for REPLACE and DELETE)

ABSTRACT TEST A.72

IDENTIFIER	`/conf/create-replace-delete/command-schema`
REQUIREMENT	Requirement 72: `/req/create-replace-delete/command-schema`
TEST PURPOSE	Validate that the server rejects commands with incompatible schemas.
TEST METHOD	1. Given a `ControlStream` resource with ID `dsId`:

ABSTRACT TEST A.72

a) Issue an HTTP CREATE request at URL `{api_root}/controlstreams/{dsId}/commands` with a command whose result structure is incompatible with the command schema registered with the control stream.

b) Verify that the server responds with an error code 400.

ABSTRACT TEST A.73

IDENTIFIER	`/conf/create-replace-delete/command-status`
REQUIREMENT	Requirement 73: `/req/create-replace-delete/command-status`
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for `CommandStatus` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint `{api_root}/commands/{cmdId}/status` (for CREATE) b) At resource endpoint `{api_root}/commands/{cmdId}/status/{id}` (for REPLACE and DELETE)

ABSTRACT TEST A.74

IDENTIFIER	`/conf/create-replace-delete/command-result`
REQUIREMENT	Requirement 74: `/req/create-replace-delete/command-result`
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for `CommandResult` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint `{api_root}/commands/{cmdId}/result` (for CREATE) b) At resource endpoint `{api_root}/commands/{cmdId}/result/{id}` (for REPLACE and DELETE)

ABSTRACT TEST A.75

IDENTIFIER	`/conf/create-replace-delete/feasibility`
REQUIREMENT	Requirement 75: `/req/create-replace-delete/feasibility`
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for `Feasibility` resources.

ABSTRACT TEST A.75

TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint `{api_root}/controlstreams/{dsId}/feasibility` (for CREATE) b) At resource endpoint `{api_root}/controlstreams/{dsId}/feasibility/{id}` (for REPLACE and DELETE) c) At resource endpoint `{api_root}/feasibility/{id}` (for REPLACE and DELETE)

ABSTRACT TEST A.76

IDENTIFIER	`/conf/create-replace-delete/feasibility-status`
REQUIREMENT	Requirement 76: `/req/create-replace-delete/feasibility-status`
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for feasibility status.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint `{api_root}/feasibility/{cmdId}/status` (for CREATE) b) At resource endpoint `{api_root}/feasibility/{cmdId}/status/{id}` (for REPLACE and DELETE)

ABSTRACT TEST A.77

IDENTIFIER	`/conf/create-replace-delete/feasibility-result`
REQUIREMENT	Requirement 77: `/req/create-replace-delete/feasibility-result`
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for feasibility result.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint `{api_root}/feasibility/{cmdId}/result` (for CREATE) b) At resource endpoint `{api_root}/feasibility/{cmdId}/result/{id}` (for REPLACE and DELETE)

ABSTRACT TEST A.78

IDENTIFIER	`/conf/create-replace-delete/system-event`

ABSTRACT TEST A.78	
REQUIREMENT	Requirement 78: `/req/create-replace-delete/system-event`
TEST PURPOSE	Validate that the server implements CREATE/REPLACE/DELETE operations correctly for `SystemEvent` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/create-replace-delete at the following endpoints: a) At resources endpoint `{api_root}/systems/{sysId}/events` (for CREATE) b) At resource endpoint `{api_root}/systems/{sysId}/events/{id}` (for REPLACE and DELETE) c) At resource endpoint `{api_root}/systemEvents/{id}` (for REPLACE and DELETE)

A.8. Conformance Class “Update”

CONFORMANCE CLASS A.8	
IDENTIFIER	`/conf/update`
REQUIREMENTS CLASS	Requirements class 8: `/req/update`
PREREQUISITES	Conformance class A.7: `/conf/create-replace-delete` http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.79: `/conf/update/datastream` Abstract test A.80: `/conf/update/datastream-update-schema` Abstract test A.81: `/conf/update/observation` Abstract test A.82: `/conf/update/observation-schema` Abstract test A.83: `/conf/update/controlstream` Abstract test A.84: `/conf/update/controlstream-update-schema` Abstract test A.85: `/conf/update/command` Abstract test A.86: `/conf/update/command-schema` Abstract test A.87: `/conf/update/command-status` Abstract test A.88: `/conf/update/command-result` Abstract test A.89: `/conf/update/feasibility` Abstract test A.90: `/conf/update/feasibility-status` Abstract test A.91: `/conf/update/feasibility-result` Abstract test A.92: `/conf/update/system-event`

ABSTRACT TEST A.79

IDENTIFIER	`/conf/update/datastream`
REQUIREMENT	Requirement 79: `/req/update/datastream`
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for `DataStream` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints: a) At resource endpoint `{api_root}/systems/{sysId}/datastreams/{id}` b) At resource endpoint `{api_root}/datastreams/{id}`

ABSTRACT TEST A.80

IDENTIFIER	`/conf/update/datastream-update-schema`
REQUIREMENT	Requirement 80: `/req/update/datastream-update-schema`
TEST PURPOSE	Validate that the server rejects `DataStream` UPDATE requests with incompatible schemas.
TEST METHOD	1. Given a `DataStream` resource with ID `dsId` that has associated observations: a) Issue HTTP UPDATE request at URL `{api_root}/datastreams/{dsId}` with a different observation schema. b) Verify that the server responds with an error code 409.

ABSTRACT TEST A.81

IDENTIFIER	`/conf/update/observation`
REQUIREMENT	Requirement 81: `/req/update/observation`
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for `Observation` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints: a) At resource endpoint `{api_root}/datastreams/{dsId}/observations/{id}` b) At resource endpoint `{api_root}/observations/{id}`

ABSTRACT TEST A.82

IDENTIFIER	`/conf/update/observation-schema`

ABSTRACT TEST A.82

REQUIREMENT	Requirement 82: `/req/update/observation-schema`
TEST PURPOSE	Validate that the server rejects observations with incompatible schemas.
TEST METHOD	1. Given a `DataStream` resource with ID `dsId`: a) Issue an HTTP PATCH request at URL `{api_root}/datastreams/{dsId}/observations/{id}` changing the observation's result to something incompatible with the observation schema registered with the datastream. b) Verify that the server responds with an error code 400.

ABSTRACT TEST A.83

IDENTIFIER	`/conf/update/controlstream`
REQUIREMENT	Requirement 83: `/req/update/controlstream`
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for `ControlStream` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints: a) At resource endpoint `{api_root}/systems/{sysId}/controlstreams/{id}` b) At resource endpoint `{api_root}/controlstreams/{id}`

ABSTRACT TEST A.84

IDENTIFIER	`/conf/update/controlstream-update-schema`
REQUIREMENT	Requirement 84: `/req/update/controlstream-update-schema`
TEST PURPOSE	Validate that the server rejects `ControlStream` UPDATE requests with incompatible schemas.
TEST METHOD	1. Given a `ControlStream` resource with ID `dsId` that has associated observations: a) Issue HTTP UPDATE request at URL `{api_root}/controlstreams/{dsId}` with a different command schema. b) Verify that the server responds with an error code 409.

ABSTRACT TEST A.85

IDENTIFIER	`/conf/update/command`

ABSTRACT TEST A.85

REQUIREMENT	Requirement 85: `/req/update/command`
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for `Command` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints: a) At resource endpoint `{api_root}/controlstreams/{dsId}/commands/{id}` b) At resource endpoint `{api_root}/commands/{id}`

ABSTRACT TEST A.86

IDENTIFIER	`/conf/update/command-schema`
REQUIREMENT	Requirement 86: `/req/update/command-schema`
TEST PURPOSE	Validate that the server rejects commands with incompatible schemas.
TEST METHOD	1. Given a `ControlStream` resource with ID `dsId`: a) Issue an HTTP PATCH request at URL `{api_root}/controlstreams/{dsId}/commands/{id}` changing the command's parameters to something incompatible with the command schema registered with the control stream. b) Verify that the server responds with an error code 400.

ABSTRACT TEST A.87

IDENTIFIER	`/conf/update/command-status`
REQUIREMENT	Requirement 87: `/req/update/command-status`
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for `CommandStatus` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints: a) At resource endpoint `{api_root}/commands/{cmdId}/status/{id}`

ABSTRACT TEST A.88

IDENTIFIER	`/conf/update/command-result`
REQUIREMENT	Requirement 88: `/req/update/command-result`

ABSTRACT TEST A.88

TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for `CommandResult` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints: a) At resource endpoint `{api_root}/commands/{cmdId}/result/{id}`

ABSTRACT TEST A.89

IDENTIFIER	`/conf/update/feasibility`
REQUIREMENT	Requirement 89: `/req/update/feasibility`
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for `Feasibility` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints: a) At resource endpoint `{api_root}/controlstreams/{dsId}/feasibility/{id}` b) At resource endpoint `{api_root}/feasibility/{id}`

ABSTRACT TEST A.90

IDENTIFIER	`/conf/update/feasibility-status`
REQUIREMENT	Requirement 90: `/req/update/feasibility-status`
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for feasibility status resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints: a) At resource endpoint `{api_root}/feasibility/{cmdId}/status/{id}`

ABSTRACT TEST A.91

IDENTIFIER	`/conf/update/feasibility-result`
REQUIREMENT	Requirement 91: `/req/update/feasibility-result`
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for feasibility result resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints:

ABSTRACT TEST A.91
a) At resource endpoint `{api_root}/feasibility/{cmdId}/result/{id}`

ABSTRACT TEST A.92	
IDENTIFIER	`/conf/update/system-event`
REQUIREMENT	Requirement 92: `/req/update/system-event`
TEST PURPOSE	Validate that the server implements the UPDATE operation correctly for `SystemEvent` resources.
TEST METHOD	1. Execute all tests from conformance class http://www.opengis.net/spec/ogcapi-features-4/1.0/conf/update at the following endpoints: a) At resource endpoint `{api_root}/systems/{sysId}/events/{id}` b) At resource endpoint `{api_root}/systemEvents/{id}`

A.9. Conformance Class "JSON Encoding"

CONFORMANCE CLASS A.9	
IDENTIFIER	`/conf/json`
REQUIREMENTS CLASS	Requirements class 9: `/req/json`
PREREQUISITE	http://www.opengis.net/spec/SWE/3.0/conf/json-record-components
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.93: `/conf/json/mediatype-read` Abstract test A.94: `/conf/json/mediatype-write` Abstract test A.95: `/conf/json/datastream-schema` Abstract test A.96: `/conf/json/obsschema-schema` Abstract test A.97: `/conf/json/observation-schema` Abstract test A.98: `/conf/json/observation-constraints` Abstract test A.99: `/conf/json/controlstream-schema`

CONFORMANCE CLASS A.9

Abstract test A.100: `/conf/json/commandschema-schema`
Abstract test A.101: `/conf/json/command-schema`
Abstract test A.102: `/conf/json/command-constraints`
Abstract test A.103: `/conf/json/commandstatus-schema`
Abstract test A.104: `/conf/json/commandresult-schema`
Abstract test A.105: `/conf/json/commandresult-constraints`
Abstract test A.106: `/conf/json/systemevent-schema`

ABSTRACT TEST A.93

IDENTIFIER	`/conf/json/mediatype-read`
REQUIREMENT	Requirement 93: `/req/json/mediatype-read`
TEST PURPOSE	Verify that the server supports the JSON format on retrieval operations.
TEST METHOD	1. For each supported conformance class: a) Request resources from the specified resources endpoint with media type `application/json`. b) Verify that the server responds with HTTP code 200. c) Verify that the `Content-Type` header of the response is set to `application/json`. d) Verify that the response is properly encoded as JSON.

ABSTRACT TEST A.94

IDENTIFIER	`/conf/json/mediatype-write`
REQUIREMENT	Requirement 94: `/req/json/mediatype-write`
TEST PURPOSE	Verify that the server advertises support for the JSON format on transactional operations.
TEST METHOD	1. For each supported conformance class: a) Verify that server advertises support for media type `application/json` in the API definition for CREATE or REPLACE operations, for the specified resources endpoint.

ABSTRACT TEST A.95

IDENTIFIER	`/conf/json/datastream-schema`
REQUIREMENT	Requirement 95: `/req/json/datastream-schema`
TEST PURPOSE	Validate that the JSON representation of `DataStream` resources is valid.
TEST METHOD	1. Request a single `DataStream` resource. a) Issue an HTTP GET request at `{api_root}/datastreams/{id}` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200. c) Validate the document against the schema dataStream.json using a JSON Schema validator. 2. Request multiple `DataStream` resources. a) Issue an HTTP GET request at `{api_root}/datastreams` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200. c) Validate the document against the schema dataStreamCollection.json using a JSON Schema validator. d) Repeat the steps above for nested `DataStream` resources endpoints `{api_root}/systems/{sysId}/datastreams`.

ABSTRACT TEST A.96

IDENTIFIER	`/conf/json/obsschema-schema`
REQUIREMENT	Requirement 96: `/req/json/obsschema-schema`
TEST PURPOSE	Validate that the JSON representation of observation schema resources is valid.
TEST METHOD	For every `DataStream` resource: 1. Issue an HTTP GET request at `{api_root}/datastreams/{id}/schema?obsFormat=application/json`. 2. Validate the document against the schema observationSchemaJson.json using a JSON Schema validator.

ABSTRACT TEST A.97

IDENTIFIER	`/conf/json/observation-schema`
REQUIREMENT	Requirement 97: `/req/json/observation-schema`

ABSTRACT TEST A.97

TEST PURPOSE	Validate that the JSON representation of `Observation` resources is valid.
TEST METHOD	1. Request a single `Observation` resource. a) Issue an HTTP GET request at `{api_root}/observations/{id}` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200. c) Validate the document against the schema observation.json using a JSON Schema validator. 2. Request multiple `Observation` resources. a) Issue an HTTP GET request at `{api_root}/observations` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200. c) Validate the document against the schema observationCollection.json using a JSON Schema validator. d) For each observation in the response, validate it with test _conf_json_observation-constraints e) Repeat the steps above for nested `Observation` resources endpoints `{api_root}/datastreams/{dsId}/observations`.

ABSTRACT TEST A.98

IDENTIFIER	`/conf/json/observation-constraints`
REQUIREMENT	Requirement 98: `/req/json/observation-constraints`
TEST PURPOSE	Validate that `Observation` result and parameters are encoded properly.
TEST METHOD	1. Retrieve the schema from the parent `DataStream` resource. 2. Validate that the `Observation` result is valid according to the `resultSchema`. 3. Validate that the `Observation` parameters, if any, are valid according to the `parametersSchema`.

ABSTRACT TEST A.99

IDENTIFIER	`/conf/json/controlstream-schema`
REQUIREMENT	Requirement 99: `/req/json/controlstream-schema`
TEST PURPOSE	Validate that the JSON representation of `ControlStream` resources is valid.

ABSTRACT TEST A.99

TEST METHOD

1. Request a single `ControlStream` resource.
   a) Issue an HTTP GET request at `{api_root}/controlstreams/{id}` with the `Accept` header set to `application/json`.
   b) Validate that a document was returned with a status code 200.
   c) Validate the document against the schema controlStream.json using a JSON Schema validator.
2. Request multiple `ControlStream` resources.
   a) Issue an HTTP GET request at `{api_root}/controlstreams` with the `Accept` header set to `application/json`.
   b) Validate that a document was returned with a status code 200.
   c) Validate the document against the schema controlStreamCollection.json using a JSON Schema validator.
   d) Repeat the steps above for nested `ControlStream` resources endpoints `{api_root}/systems/{sysId}/controlstreams`.

ABSTRACT TEST A.100

IDENTIFIER	`/conf/json/commandschema-schema`
REQUIREMENT	Requirement 100: `/req/json/commandschema-schema`
TEST PURPOSE	Validate that the JSON representation of command schema resources is valid.
TEST METHOD	For every `ControlStream` resource: 1. Issue an HTTP GET request at `{api_root}/controlstreams/{id}/schema?cmdFormat=application/json`. 2. Validate the document against the schema commandSchemaJson.json using a JSON Schema validator.

ABSTRACT TEST A.101

IDENTIFIER	`/conf/json/command-schema`
REQUIREMENT	Requirement 101: `/req/json/command-schema`
TEST PURPOSE	Validate that the JSON representation of `Command` resources is valid.
TEST METHOD	1. Request a single `Command` resource. a) Issue an HTTP GET request at `{api_root}/commands/{id}` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200.

ABSTRACT TEST A.101

c) Validate the document against the schema command.json using a JSON Schema validator.

2. Request multiple `Command` resources.
   a) Issue an HTTP GET request at `{api_root}/commands` with the `Accept` header set to `application/json`.
   b) Validate that a document was returned with a status code 200.
   c) Validate the document against the schema commandCollection.json using a JSON Schema validator.
   d) Repeat the steps above for nested `Command` resources endpoints `{api_root}/controlstreams/{dsId}/commands`.

ABSTRACT TEST A.102

IDENTIFIER	`/conf/json/command-constraints`
REQUIREMENT	Requirement 102: `/req/json/command-constraints`
TEST PURPOSE	Validate that `Command` parameters are encoded properly.
TEST METHOD	1. Retrieve the schema from the parent `ControlStream` resource. 2. Validate that the `Command` parameters are valid according to the `parametersSchema`.

ABSTRACT TEST A.103

IDENTIFIER	`/conf/json/commandstatus-schema`
REQUIREMENT	Requirement 103: `/req/json/commandstatus-schema`
TEST PURPOSE	Validate that the JSON representation of `CommandStatus` resources is valid.
TEST METHOD	1. Request a single `CommandStatus` resource. a) Issue an HTTP GET request at `{api_root}/commands/{cmdId}/status/{id}` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200. c) Validate the document against the schema commandStatus.json using a JSON Schema validator. 2. Request multiple `CommandStatus` resources. a) Issue an HTTP GET request at `{api_root}/commands/{cmdId}/status` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200.

ABSTRACT TEST A.103

c) Validate the document against the schema commandStatusCollection.json using a JSON Schema validator.

ABSTRACT TEST A.104

IDENTIFIER	`/conf/json/commandresult-schema`
REQUIREMENT	Requirement 104: `/req/json/commandresult-schema`
TEST PURPOSE	Validate that the JSON representation of `CommandResult` resources is valid.
TEST METHOD	1. Request a single `CommandResult` resource. a) Issue an HTTP GET request at `{api_root}/commands/{cmdId}/result/{id}` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200. c) Validate the document against the schema commandResult.json using a JSON Schema validator. 2. Request multiple `CommandResult` resources. a) Issue an HTTP GET request at `{api_root}/commands/{cmdId}/result` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200. c) Validate the document against the schema commandResultCollection.json using a JSON Schema validator.

ABSTRACT TEST A.105

IDENTIFIER	`/conf/json/commandresult-constraints`
REQUIREMENT	Requirement 105: `/req/json/commandresult-constraints`
TEST PURPOSE	Validate that `CommandResult` results are encoded properly.
TEST METHOD	1. Retrieve the schema from the parent `ControlStream` resource. 2. Validate that the `CommandResult` `result` field is valid according to the `resultSchema`.

ABSTRACT TEST A.106

IDENTIFIER	`/conf/json/systemevent-schema`
REQUIREMENT	Requirement 106: `/req/json/systemevent-schema`

ABSTRACT TEST A.106	
TEST PURPOSE	Validate that the JSON representation of `SystemEvent` resources is valid.
TEST METHOD	1. Request a single `SystemEvent` resource. a) Issue an HTTP GET request at `{api_root}/systemEvents/{id}` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200. c) Validate the document against the schema systemEvent.json using a JSON Schema validator. 2. Request multiple `SystemEvent` resources. a) Issue an HTTP GET request at `{api_root}/systemEvents` with the `Accept` header set to `application/json`. b) Validate that a document was returned with a status code 200. c) Validate the document against the schema systemEventCollection.json using a JSON Schema validator. d) Repeat the steps above for nested `SystemEvent` resources endpoints `{api_root}/systems/{sysId}/events`.

A.10. Conformance Class "SWE Common JSON Encoding"

CONFORMANCE CLASS A.10	
IDENTIFIER	`/conf/swecommon-json`
REQUIREMENTS CLASS	Requirements class 10: `/req/swecommon-json`
PREREQUISITE	http://www.opengis.net/spec/SWE/3.0/conf/json-encoding-rules
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.107: `/conf/swecommon-json/mediatype-read` Abstract test A.108: `/conf/swecommon-json/mediatype-write` Abstract test A.109: `/conf/swecommon-json/obsschema-schema` Abstract test A.110: `/conf/swecommon-json/obsschema-mapping`

CONFORMANCE CLASS A.10

	Abstract test A.111: `/conf/swecommon-json/observation-encoding` Abstract test A.112: `/conf/swecommon-json/cmdschema-schema` Abstract test A.113: `/conf/swecommon-json/cmdschema-mapping` Abstract test A.114: `/conf/swecommon-json/command-encoding`

ABSTRACT TEST A.107

IDENTIFIER	`/conf/swecommon-json/mediatype-read`
REQUIREMENT	Requirement 107: `/req/swecommon-json/mediatype-read`
TEST PURPOSE	Verify that the server supports the SWE Common JSON format on retrieval operations.
TEST METHOD	1. For at least one of the `Observation` and/or `Command` resources endpoint: a) Verify that the server advertises support for media type `application/swe+json` in the API definition for retrieval operations. b) Request resources from the resources endpoint with media type `application/swe+json`. c) Verify that the server responds with HTTP code 200. d) Verify that the `Content-Type` header of the response is set to `application/swe+json`. e) Verify that the response is properly encoded as JSON.

ABSTRACT TEST A.108

IDENTIFIER	`/conf/swecommon-json/mediatype-write`
REQUIREMENT	Requirement 108: `/req/swecommon-json/mediatype-write`
TEST PURPOSE	Verify that the server advertises support for the SWE Common JSON format on transactional operations.
TEST METHOD	1. For at least one of the `Observation` and/or `Command` resources endpoint: a) Verify that the server advertises support for media type `application/swe+json` in the API definition for CREATE or REPLACE operations.

ABSTRACT TEST A.109

IDENTIFIER	`/conf/swecommon-json/obsschema-schema`
REQUIREMENT	Requirement 109: `/req/swecommon-json/obsschema-schema`
TEST PURPOSE	Validate that the JSON representation of observation schema resources is valid.
TEST METHOD	For every `DataStream` resource: 1. Issue an HTTP GET request at `{api_root}/datastreams/{id}/schema?obsFormat=application/swe+json`. 2. Validate the document against the schema observationSchemaSwe.json using a JSON Schema validator. 3. Validate that the SWE Common encoding is set to `JSONEncoding`. 4. Validate the schema using test _conf_swecommon-json_obsschema-mapping

ABSTRACT TEST A.110

IDENTIFIER	`/conf/swecommon-json/obsschema-mapping`
REQUIREMENT	Requirement 110: `/req/swecommon-json/obsschema-mapping`
TEST PURPOSE	Verify that the mandatory fields are present in the schema.
TEST METHOD	1. Scan the schema and validate that at least one Time data component is present 2. Validate the the definition field of the Time component is one of: • `http://www.w3.org/ns/sosa/phenomenonTime` • `http://www.opengis.net/def/property/OGC/0/SamplingTime` • `http://www.w3.org/ns/sosa/resultTime`

ABSTRACT TEST A.111

IDENTIFIER	`/conf/swecommon-json/observation-encoding`
REQUIREMENT	Requirement 111: `/req/swecommon-json/observation-encoding`
TEST PURPOSE	Validate that the JSON representation of `Observation` resources is valid.
TEST METHOD	1. For every `DataStream` that advertises support for the `application/swe+json` format: a) Issue an HTTP GET request at `{api_root}/datastreams/{dsId}/observations` with the `Accept` header set to `application/swe+json`. b) Validate that a document was returned with a status code 200.

ABSTRACT TEST A.111

c) Validate the response using a SWE Common validator implementing the JSON encoding rules.

ABSTRACT TEST A.112

IDENTIFIER	`/conf/swecommon-json/cmdschema-schema`
REQUIREMENT	Requirement 112: `/req/swecommon-json/cmdschema-schema`
TEST PURPOSE	Validate that the JSON representation of command schema resources is valid.
TEST METHOD	For every `ControlStream` resource: 1. Issue an HTTP GET request at `{api_root}/controlstreams/{id}/schema?cmdFormat=application/swe+json`. 2. Validate the document against the schema commandSchemaSwe.json using a JSON Schema validator. 3. Validate that the SWE Common encoding is set to `JSONEncoding`. 4. Validate the schema using test _conf_swecommon-json_cmdschema-mapping

ABSTRACT TEST A.113

IDENTIFIER	`/conf/swecommon-json/cmdschema-mapping`
REQUIREMENT	Requirement 113: `/req/swecommon-json/cmdschema-mapping`
TEST PURPOSE	Verify that the mandatory fields are present in the schema.
TEST METHOD	1. Scan the schema and validate that at least one Time data component is present 2. Validate the the definition field of the Time component is one of: • `http://www.opengis.net/def/property/OGC/0/IssueTime`

ABSTRACT TEST A.114

IDENTIFIER	`/conf/swecommon-json/command-encoding`
REQUIREMENT	Requirement 114: `/req/swecommon-json/command-encoding`
TEST PURPOSE	Validate that the JSON representation of `Command` resources is valid.
TEST METHOD	1. For every `ControlStream` that advertises support for the `application/swe+json` format:

ABSTRACT TEST A.114	
	a) Issue an HTTP GET request at `{api_root}/controlstreams/{dsId}/commands` with the `Accept` header set to `application/swe+json`. b) Validate that a document was returned with a status code 200. c) Validate the response using a SWE Common validator implementing the JSON encoding rules.

A.11. Conformance Class "SWE Common Text Encoding"

CONFORMANCE CLASS A.11	
IDENTIFIER	`/conf/swecommon-text`
REQUIREMENTS CLASS	Requirements class 11: `/req/swecommon-text`
PREREQUISITE	http://www.opengis.net/spec/SWE/3.0/conf/text-encoding-rules
TARGET TYPE	Web API
CONFORMANCE TESTS	Abstract test A.115: `/conf/swecommon-text/mediatype-read` Abstract test A.116: `/conf/swecommon-text/mediatype-write` Abstract test A.117: `/conf/swecommon-text/obsschema-schema` Abstract test A.118: `/conf/swecommon-text/obsschema-mapping` Abstract test A.119: `/conf/swecommon-text/observation-encoding` Abstract test A.120: `/conf/swecommon-text/cmdschema-schema` Abstract test A.121: `/conf/swecommon-text/cmdschema-mapping` Abstract test A.122: `/conf/swecommon-text/command-encoding`

ABSTRACT TEST A.115	
IDENTIFIER	`/conf/swecommon-text/mediatype-read`

ABSTRACT TEST A.115

REQUIREMENT	Requirement 115: `/req/swecommon-text/mediatype-read`
TEST PURPOSE	Verify that the server supports the SWE Common Text format on retrieval operations.
TEST METHOD	1. For at least one of the `Observation` and/or `Command` resources endpoint: a) Verify that the server advertises support for media type `application/swe+binary` in the API definition for retrieval operations. b) Request resources from the resources endpoint with media type `application/swe+text`. c) Verify that the server responds with HTTP code 200. d) Verify that the `Content-Type` header of the response is set to `application/swe+text`.

ABSTRACT TEST A.116

IDENTIFIER	`/conf/swecommon-text/mediatype-write`
REQUIREMENT	Requirement 116: `/req/swecommon-text/mediatype-write`
TEST PURPOSE	Verify that the server advertises support for the SWE Common Text format on transactional operations.
TEST METHOD	1. For at least one of the `Observation` and/or `Command` resources endpoint: a) Verify that the server advertises support for media type `application/swe+text` in the API definition for CREATE or REPLACE operations.

ABSTRACT TEST A.117

IDENTIFIER	`/conf/swecommon-text/obsschema-schema`
REQUIREMENT	Requirement 117: `/req/swecommon-text/obsschema-schema`
TEST PURPOSE	Validate that the JSON representation of observation schema resources is valid.
TEST METHOD	For every `DataStream` resource: 1. Issue an HTTP GET request at `{api_root}/datastreams/{id}/schema?obsFormat=application/swe+text`. 2. Validate the document against the schema observationSchemaSwe.json using a JSON Schema validator. 3. Validate that the SWE Common encoding is set to `TextEncoding`. 4. Validate the schema using test _conf_swecommon-text_obsschema-mapping

ABSTRACT TEST A.118

IDENTIFIER	`/conf/swecommon-text/obsschema-mapping`
REQUIREMENT	Requirement 118: `/req/swecommon-text/obsschema-mapping`
TEST PURPOSE	Verify that the mandatory fields are present in the schema.
TEST METHOD	Execute test _conf_swecommon-json_obsschema-mapping

ABSTRACT TEST A.119

IDENTIFIER	`/conf/swecommon-text/observation-encoding`
REQUIREMENT	Requirement 119: `/req/swecommon-text/observation-encoding`
TEST PURPOSE	Validate that the Text (DSV) representation of `Observation` resources is valid.
TEST METHOD	1. For every `DataStream` that advertises support for the `application/swe+text` format: a) Issue an HTTP GET request at `{api_root}/datastreams/{dsId}/observations` with the `Accept` header set to `application/swe+text`. b) Validate that a document was returned with a status code 200. c) Validate the response using a SWE Common validator implementing the Text encoding rules.

ABSTRACT TEST A.120

IDENTIFIER	`/conf/swecommon-text/cmdschema-schema`
REQUIREMENT	Requirement 120: `/req/swecommon-text/cmdschema-schema`
TEST PURPOSE	Validate that the JSON representation of command schema resources is valid.
TEST METHOD	For every `ControlStream` resource: 1. Issue an HTTP GET request at `{api_root}/controlstreams/{id}/schema?cmdFormat=application/swe+text`. 2. Validate the document against the schema commandSchemaSwe.json using a JSON Schema validator. 3. Validate that the SWE Common encoding is set to `TextEncoding`. 4. Validate the schema using test _conf_swecommon-text_cmdschema-mapping

ABSTRACT TEST A.121	
IDENTIFIER	`/conf/swecommon-text/cmdschema-mapping`
REQUIREMENT	Requirement 121: `/req/swecommon-text/cmdschema-mapping`
TEST PURPOSE	Verify that the mandatory fields are present in the schema.
TEST METHOD	Execute test _conf_swecommon-json_cmdschema-mapping

ABSTRACT TEST A.122	
IDENTIFIER	`/conf/swecommon-text/command-encoding`
REQUIREMENT	Requirement 122: `/req/swecommon-text/command-encoding`
TEST PURPOSE	Validate that the Text (DSV) representation of `Command` resources is valid.
TEST METHOD	1. For every `ControlStream` that advertises support for the `application/swe+text` format: a) Issue an HTTP GET request at `{api_root}/controlstreams/{dsId}/commands` with the `Accept` header set to `application/swe+text`. b) Validate that a document was returned with a status code 200. c) Validate the response using a SWE Common validator implementing the Text encoding rules.

A.12. Conformance Class “SWE Common Binary Encoding”

CONFORMANCE CLASS A.12	
IDENTIFIER	`/conf/swecommon-binary`
REQUIREMENTS CLASS	Requirements class 12: `/req/swecommon-binary`
PREREQUISITE	http://www.opengis.net/spec/SWE/3.0/conf/binary-encoding-rules
TARGET TYPE	Web API

CONFORMANCE CLASS A.12

CONFORMANCE TESTS	Abstract test A.123: `/conf/swecommon-binary/mediatype-read` Abstract test A.124: `/conf/swecommon-binary/mediatype-write` Abstract test A.125: `/conf/swecommon-binary/obsschema-schema` Abstract test A.126: `/conf/swecommon-binary/obsschema-mapping` Abstract test A.127: `/conf/swecommon-binary/observation-encoding` Abstract test A.128: `/conf/swecommon-binary/cmdschema-schema` Abstract test A.129: `/conf/swecommon-binary/cmdschema-mapping` Abstract test A.130: `/conf/swecommon-binary/command-encoding`

ABSTRACT TEST A.123

IDENTIFIER	`/conf/swecommon-binary/mediatype-read`
REQUIREMENT	Requirement 123: `/req/swecommon-binary/mediatype-read`
TEST PURPOSE	Verify that the server supports the SWE Common Binary format on retrieval operations.
TEST METHOD	1. For at least one of the `Observation` and/or `Command` resources endpoint: a) Verify that the server advertises support for media type `application/swe+binary` in the API definition for retrieval operations. b) Request resources from the resources endpoint with media type `application/swe+binary`. c) Verify that the server responds with HTTP code 200. d) Verify that the `Content-Type` header of the response is set to `application/swe+binary`.

ABSTRACT TEST A.124

IDENTIFIER	`/conf/swecommon-binary/mediatype-write`
REQUIREMENT	Requirement 124: `/req/swecommon-binary/mediatype-write`
TEST PURPOSE	Verify that the server advertises support for the SWE Common Binary format on transactional operations.

ABSTRACT TEST A.124

TEST METHOD	1. For at least one of the `Observation` and/or `Command` resources endpoint: a) Verify that the server advertises support for media type `application/swe+binary` in the API definition for CREATE or REPLACE operations.

ABSTRACT TEST A.125

IDENTIFIER	`/conf/swecommon-binary/obsschema-schema`
REQUIREMENT	Requirement 125: `/req/swecommon-binary/obsschema-schema`
TEST PURPOSE	Validate that the JSON representation of observation schema resources is valid.
TEST METHOD	For every `DataStream` resource: 1. Issue an HTTP GET request at `{api_root}/datastreams/{id}/schema?obsFormat=application/swe+binary`. 2. Validate the document against the schema observationSchemaSwe.json using a JSON Schema validator. 3. Validate that the SWE Common encoding is set to `BinaryEncoding`. 4. Validate the schema using test _conf_swecommon-binary_obsschema-mapping

ABSTRACT TEST A.126

IDENTIFIER	`/conf/swecommon-binary/obsschema-mapping`
REQUIREMENT	Requirement 126: `/req/swecommon-binary/obsschema-mapping`
TEST PURPOSE	Verify that the mandatory fields are present in the schema.
TEST METHOD	Execute test _conf_swecommon-json_obsschema-mapping

ABSTRACT TEST A.127

IDENTIFIER	`/conf/swecommon-binary/observation-encoding`
REQUIREMENT	Requirement 127: `/req/swecommon-binary/observation-encoding`
TEST PURPOSE	Validate that the binary representation of `Observation` resources is valid.
TEST METHOD	1. For every `DataStream` that advertises support for the `application/swe+binary` format:

ABSTRACT TEST A.127

a) Issue an HTTP GET request at `{api_root}/datastreams/{dsId}/observations` with the `Accept` header set to `application/swe+binary`.

b) Validate that a document was returned with a status code 200.

c) Validate the response using a SWE Common validator implementing the Text encoding rules.

ABSTRACT TEST A.128

IDENTIFIER	`/conf/swecommon-binary/cmdschema-schema`
REQUIREMENT	Requirement 128: `/req/swecommon-binary/cmdschema-schema`
TEST PURPOSE	Validate that the JSON representation of command schema resources is valid.
TEST METHOD	For every `ControlStream` resource: 1. Issue an HTTP GET request at `{api_root}/controlstreams/{id}/schema?cmdFormat=application/swe+binary`. 2. Validate the document against the schema commandSchemaSwe.json using a JSON Schema validator. 3. Validate that the SWE Common encoding is set to `BinaryEncoding`. 4. Validate the schema using test _conf_swecommon-binary_cmdschema-mapping

ABSTRACT TEST A.129

IDENTIFIER	`/conf/swecommon-binary/cmdschema-mapping`
REQUIREMENT	Requirement 129: `/req/swecommon-binary/cmdschema-mapping`
TEST PURPOSE	Verify that the mandatory fields are present in the schema.
TEST METHOD	Execute test _conf_swecommon-json_cmdschema-mapping

ABSTRACT TEST A.130

IDENTIFIER	`/conf/swecommon-binary/command-encoding`
REQUIREMENT	Requirement 130: `/req/swecommon-binary/command-encoding`

ABSTRACT TEST A.130

TEST PURPOSE Validate that the binary representation of `Command` resources is valid.

TEST METHOD

1. For every `ControlStream` that advertises support for the `application/swe+binary` format:
   a) Issue an HTTP GET request at `{api_root}/controlstreams/{dsId}/commands` with the `Accept` header set to `application/swe+binary`.
   b) Validate that a document was returned with a status code 200.
   c) Validate the response using a SWE Common validator implementing the Text encoding rules.

B

ANNEX B (INFORMATIVE) EXAMPLES

ANNEX B (INFORMATIVE) EXAMPLES

More JSON examples are available in the project's GitHub repository at:

https://schemas.opengis.net/ogcapi/connected-systems/part2/1.0/openapi/examples

C

ANNEX C (INFORMATIVE) RELATIONSHIP WITH OTHER OGC/ISO STANDARDS (INFORMATIVE)

ANNEX C (INFORMATIVE) RELATIONSHIP WITH OTHER OGC/ISO STANDARDS (INFORMATIVE)

See OGC API — Connected Systems — Part 1, Annex C, for a description of relationships with other Standards.

D

ANNEX D (INFORMATIVE) REVISION HISTORY

ANNEX D (INFORMATIVE) REVISION HISTORY

DATE	RELEASE	EDITOR	PRIMARY CLAUSES MODIFIED	DESCRIPTION
2023-01-10	1.0 draft	Alex Robin	All	Initial draft version
2023-04-21	1.0 draft	Alex Robin	All	Migration to Metanorma
2024-06-10	1.0 draft	Alex Robin	All	Added missing sections, alignment with Part 1
2024-09-10	1.0 draft	Alex Robin	All	Added ATS
2025-03-18	1.0 draft	Christian Autermann	All	Incorporated feedback from public comments

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Simon Cox: OGC 10-004r3, *Topic 20 — Observations and Measurements*. Open Geospatial Consortium (2013). http://www.opengis.net/doc/as/om/2.0.

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

[3] Mark Burgoyne, David Blodgett, Charles Heazel, Chris Little: OGC 19-086r6, *OGC API — Environmental Data Retrieval Standard*. Open Geospatial Consortium (2023). http://www.opengis.net/doc/IS/ogcapi-edr-1/1.1.0.

[4] Taehoon Kim, Kyoung-Sook Kim, Mahmoud SAKR, Martin Desruisseaux: OGC 22-003r3, *OGC API — Moving Features — Part 1: Core*. Open Geospatial Consortium (2024). http://www.opengis.net/doc/IS/ogcapi-movingfeatures-1/1.0.

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

[6] Steve Liang, Tania Khalafbeigi: OGC 17-079r1, *OGC SensorThings API Part 2 – Tasking Core*. Open Geospatial Consortium (2019). http://www.opengis.net/doc/IS/sensorthings-part2-TaskingCore/1.0.0.

[7] Arne Bröring, Christoph Stasch, Johannes Echterhoff: OGC 12-006, *OGC® Sensor Observation Service Interface Standard*. Open Geospatial Consortium (2012). http://www.opengis.net/doc/IS/SOS/2.0.0.

[8] Ingo Simonis, Johannes Echterhoff: OGC 09-000, *OGC® Sensor Planning Service Implementation Standard*. Open Geospatial Consortium (2011). https://portal.ogc.org/files/?artifact_id=38478.

[9] QUDT Quantity Kinds, Version 2.1. https://www.qudt.org/doc/DOC_VOCAB-QUANTITY-KINDS.html